Automated Incident Reporting (AIR) [3]

The automated incident reporting (AIR) system allows you to measure overall system availability for the following products:

- Transmission Control Protocol/Internet Protocol (TCP/IP)
- Network Queuing System (NQS)
- Online tapes
- UNICOS kernel
- Unified Resource Manager (URM)

Warning: The AIR feature is not part of a Cray ML-Safe configuration. This section **does not** contain any further warnings or information pertaining to the use of a Cray ML-Safe configuration of the UNICOS system.

3.1 AIR Components Overview

The AIR system consists of four main components, as follows:

- Configuration file
- Coordinator daemon
- Monitoring functions
- Report generator

3.1.1 AIR Configuration File

The AIR configuration file, `/usr/air/config_file`, contains definitions for all the configurable aspects of the AIR system, written in a simple configuration language syntax. All AIR system components refer to this file at initiation for information. Scanning, printing, validation, and translation routines manage the processing of the data in the file.

Caution: AIR configuration file, `/usr/air/config_file`, can be maintained through the UNICOS Installation Menu System (installation tool). If the installation tool is used to maintain this file, it should never be edited manually.

3.1.2 AIR Coordinator Daemon

The AIR coordinator daemon, `aird`, executes configured functions at the specified rates and enacts the return code processing cues.

The coordinator translates the configuration file into a work list consisting of functions associated with each monitored product. The coordinator keeps a running clock, executing the functions with rates indicating that they should be executed. When the coordinator is not executing functions, it waits for function completion. If there are no functions for which to wait, the coordinator sleeps until the next time a function is configured to be executed.

3.1.3 AIR Monitoring Functions

The AIR monitoring functions are product verification processes; these functions can be either shell scripts or executable binaries. The implementation of the functions for each monitored product follows a hierarchical philosophy. Several functions are specified for each monitored product, and they are differentiated by the cost of the resources they use and the aspects of the product that they test. For example, a function that verifies that a process exists on the system would be low cost, and, thus, could be executed more frequently than a function that required a tape mount. However, the higher-cost function provides a better assurance of product verification. These issues must be balanced by the rate specification in the configuration file and be configured for each site's specific needs.

3.1.4 AIR Report Generator

The AIR report generator collects information, such as the AIR configuration and monitoring function event records from the coordinator log file, and presents product availability and summary information in a text format.

The coordinator reads input from the configuration file and translates that data into a series of event functions for periodic execution. The results of each function's executions are processed on completion, and pertinent information is written to the coordinator's log file. Periodically, reports can be generated by executing the report generation procedures, as follows:

Command	Description
`airprconf`	Prints AIR configuration file contents from the configuration headers in the `aird` binary log file (see `airprconf`(8)).
`airsum`	Generates availability summary reports based on the `aird`(8) binary log file (see `airsum`(8))
`airtsum`	Generates detailed AIR reports based on the coordinator binary log file (see `airtsum`(8))
`airdet`	Generates detailed AIR reports based on the `aird` binary log file (see `airdet`(8))

3.2 Initiation and Administration

The `aird`(8) process is initiated automatically at system boot time. `aird` is listed in the system `/etc/daemons` file and is started in the same manner as the other system daemons. The `/usr/air/bin/start_air` script contains the sequence used to start the `aird` process; use this script if you need to start `aird` manually. (See `start_air`(8) for more information.)

For systems running AIR, the following scripts are available (to be executed periodically by `cron`) to perform useful, daily functions:

Shell script	Description
`mvfiles`	Moves the AIR log files to data directories, resetting the log files. If you do not execute this script periodically in your system, the binary log file written to by `aird` will grow quite large.
`airdchk`	Monitors `aird`. This script uses the `airexist`(8) command to verify that `aird` exists on the system. Mail is sent to root if `aird` is not running.

These two scripts are located in the `/usr/air/bin` directory. The following example shows `crontab` entries for `mvfiles` and `airdchk`:

```
#
#    AIR utilities for periodic execution by cron.
#    Execute aird checking test every fifteen minutes.
#    Move the log files every Sunday.
#
15 * * * * /usr/air/bin/airdchk
0 0 * * 0 /usr/air/bin/mvfiles
```

3.3 AIR Configuration

The AIR configuration file contains definitions for all configurable aspects of the AIR system. The `aird` process reads the configuration file and translates the contents into monitoring functions that are executed periodically. In addition to initiating its internal processing worklist, `aird` also sets any environment variables specified in the configuration file.

AIR configuration file can be maintained by using the UNICOS Installation Menu System (installation tool). For completeness, the following sections describe both the installation tool menus for AIR and the configuration file itself.

The AIR configuration file is composed of statements written in the AIR configuration language. This section explains the configuration language in detail along with the corresponding installation tool menus, examines the default configuration file, `/usr/air/config_file`, and investigates tuning and validating the configuration file.

3.3.1 Basic Syntactic Rules

The AIR configuration language is composed of a defined set of keywords and their associated arguments. Each line of the configuration file is blank (white space or a new line), a comment (containing a #, text and/or white space), or a keyword and its associated arguments. A comment is permitted on a keyword line.

The following basic syntactic rules apply:

- The configuration language can contain only printable ASCII characters; the parser exits with an error if it finds an unprintable, non-ASCII character in the configuration file.
- Keywords are uppercase names, and user values are lowercase names.
- Noncomment lines begin with a keyword followed by the appropriate arguments.
- Comments in the file are designated like shell comments, beginning with the # character and continuing until an end-of-line is encountered.

- Only one keyword may be present on a line, and it must begin with the first nonwhite character on that line. The `CONFIG` keyword must be the first keyword, and the `ENDCONFIG` keyword must be the last keyword.
- The maximum line length is 4096 characters.
- Legal separators are white space, tabs, colons, commas, and semicolons.
- All path names specified as arguments to keywords must be the full path name of the file. (AIR validation routines ensure that all path names begin with a `/`.)
- All rates specified as arguments to keywords are interpreted as minutes by default. For example, a specification of 300 is interpreted as 5 hours. The `aird -C` option lets you change the conversion factor of the specified arguments (see `aird`(8)).

3.3.2 Configuration Keywords

This section contains lists of available keywords and associated arguments. Refer to the configuration file on your system or to the AIR configuration menu in the installation tool while examining these sections, noting the location and value of each keyword and its arguments. The keywords are discussed in the order they appear in the released configuration file.

3.3.2.1 File Delineation Keywords

The keywords described in the following sections define the beginning and end of the configuration file.

3.3.2.1.1 `CONFIG` *name*

The `CONFIG` *name* keyword marks the beginning of the configuration specification. Only comment or blank lines are allowed before a `CONFIG` line. The *name* argument is the name of the configuration, which is any string.

3.3.2.1.2 `ENDCONFIG` *name*

The `ENDCONFIG` *name* keyword marks the end of the configuration specification. Only comment or blank lines are allowed after an `ENDCONFIG` line. The *name* argument is the name of the configuration, which is any string, but must match the *name* specified with the `CONFIG` keyword in the file.

3.3.2.2 Basic Operational Keywords

In your configuration file, the keywords described in the following sections appear immediately after the `CONFIG` keyword and before a `PRODUCT` or `FUNCTION` keyword specification. These keywords define the basic operational configuration for the AIR system.

3.3.2.2.1 Installation Tool

The operational keywords correspond to the following installation tool menu:

```
M-> Configure system ->
        M-> AIR configuration ->
                M-> Coordinator setup ->

    AIR Coordinator setup

S->  AIR daemon binary log file name          /usr/spool/air/logs/blog
     Monitoring function execution directory  /usr/air/test
     AIR daemon heartbeat rate                15
     AIR daemon ASCII log file name           /usr/spool/air/logs/coord.log
     AIR daemon debugging level               0
     AIR daemon information log level         0
```

3.3.2.2.2 `COORD_LOG` *file*

The `COORD_LOG` *file* keyword specifies the absolute path name to the `aird` ASCII log file. This log file is used for debugging purposes only. Any information needed by the report generators is logged into `aird`'s binary log file. The `COORD_LOGLEV` keyword specifies the number of messages written to this ASCII log.

3.3.2.2.3 `COORD_TESTDIR` *dir*

The `COORD_TESTDIR` *dir* keyword specifies the absolute path name to the directory where the configured monitoring functions are executed.

3.3.2.2.4 `COORD_HBEAT` *rate*

The `COORD_HBEAT` *rate* keyword specifies the rate at which `aird` should log its own heartbeat record into its binary log file. The report generators use this heartbeat record and the configured rate when determining AIR system availability.

3.3.2.2.5 COORD_DEBUG *level*

The COORD_DEBUG *level* keyword specifies the number of diagnostic messages that should be logged to the aird ASCII log file (the location of which is set by using the COORD_LOG keyword). The *level* argument is a number 0 through 20; however, because this keyword is used for debugging purposes only, it is recommended that *level* usually be set to 0.

3.3.2.2.6 COORD_BLOG *file*

The COORD_BLOG *file* keyword specifies the absolute path name to aird 's binary log file. The report generators use this log file when determining the availability of the monitored products. Refer to Section 3.5, page 174, for more information on the contents and use of this file.

3.3.2.2.7 COORD_LOGLEV *level*

The COORD_LOGLEV *level* keyword specifies the number of general informational messages that should be logged to the aird ASCII log file. The *level* argument is a number 0 through 20; however, because this keyword is used for debugging purposes only, it is recommended that *level* usually be set to 0.

3.3.2.2.8 TYPE *tag types*

The TYPE *tag type* keywords are configured in the following installation tool menu:

```
M-> Configure system ->
     M-> AIR configuration ->
          M-> Return tags and types setup ->
```

```
             AIR Return tags and types setup

     Tag Name    Tag Type          Type 1      Type 2       Type 3
     ----------  ----------------  ----------  ----------   ------
E->  PASSED      PROD_AVAILABLE
     FAILED      PROD_UNAVAILABLE
     CHANGED     PROD_AVAILABLE    WARN_ADMIN
     NODEVICE    PROD_AVAILABLE    WARN_ADMIN  WARN_OPS
     NORESERVE   PROD_AVAILABLE    WARN_ADMIN
     TIMEOFDAY   PROD_AVAILABLE
     NOMOUNT     PROD_UNAVAILABLE  WARN_OPS    WARN_ADMIN
     BADWRITE    PROD_UNAVAILABLE
```

```
BADJOB      PROD_UNAVAILABLE
TIMEDOUT    PROD_AVAILABLE    WARN_ADMIN
QSUBFAILED  PROD_UNAVAILABLE
AUDITERROR  PROD_AVAILABLE    WARN_ADMIN
TCPFAILED   PROD_UNAVAILABLE
UDPFAILED   PROD_UNAVAILABLE
ICMPFAILED  PROD_UNAVAILABLE
```

The TYPE *tag types* keyword defines return tags and their associated types. The tags are set as environment variables. aird and the monitoring functions use the tags to communicate. The monitoring functions use the tags as return values. The report generators use the types to report various aspects of the system availability. The tags are also used in the MESSAGE and RETURN keyword arguments, and additional text and subsequent actions are assigned to the tag.

Other than two required tags, PASSED and FAILED, the tag argument assignment is arbitrary; however, the tags defined in the configuration file must match the expected return values for the configured monitoring functions. In other words, for every expected return value in the monitoring functions, a TYPE keyword definition with that return value listed as the tag argument must exist.

The report generators use the PROD_AVAILABLE and PROD_UNAVAILABLE types extensively when determining product availability.

The following types are allowed:

```
PROD_AVAILABLE   PROD_UNAVAILABLE   PROD_WARNING   SEND_MAIL
SPR              OIR                WARN_OPS       WARN_ADMIN
WARN_USERS       10%                20%            30%
40%              50%                60%            70%
80%              90%                91%            92%
93%              94%                95%            96%
97%              98%                99%            100%
```

3.3.2.3 Monitored Products Keywords

The keywords in the following sections appear immediately following the TYPES keyword definitions in your configuration file. These keywords specify the products to be monitored by the AIR system and configure the monitoring functions to be used for each specified product. An explanation of how products and functions are configured in the installation tool follows the description of the keywords.

3.3.2.3.1 `PRODUCT` *name status*

The `PRODUCT` *name status* keyword marks the beginning of a product definition and is always paired with the `ENDPRODUCT` keyword. You can define multiple products in a configuration file (within the `CONFIG` and `ENDCONFIG` keyword pair); however, you cannot nest product specifications within other product specifications. You can define single or multiple monitoring functions within each `PRODUCT` / `ENDPRODUCT` pair. The *name* argument, which is indicated in the report generators output, is an arbitrary string but it must be unique within the configuration. The *status* argument indicates whether a product is active (`ON`) or inactive (`OFF`). A product that is inactive is still part of the configuration, but no functions defined within that product are executed.

3.3.2.3.2 `ENDPRODUCT` *name*

The `ENDPRODUCT` *name* keyword marks the end of a specific product specification. This keyword is always paired with the `PRODUCT` keyword.

3.3.2.3.3 `MESSAGE` *tag message*

The `MESSAGE` *tag message* keyword specifies a text message to be associated with the specified tag. As described in the `TYPES` keyword definition, the return tag indicates a status returned by a monitoring function after executing. `aird` sets these return tags to environment variables prior to the execution of the monitoring functions. The report generators use the specified *message* text when reporting availability statistics. The *tag* argument must be one of the tags defined in a `TYPE` keyword specification. The *message* is any arbitrary string. The entire line is limited to 4096 characters.

3.3.2.3.4 `FUNCTION` *name status*

The `FUNCTION` *name status* keyword marks the beginning of a function definition and is always paired with an `ENDFUNCTION` keyword. You can define multiple functions for a product (within a `PRODUCT` / `ENDPRODUCT` keyword pair); however, you cannot nest function specifications within other function specifications. The *name* argument which is indicated in the report generators output, must be unique within a product specification, but does not need to be unique within the configuration. The *status* argument indicates whether a function is active (`ON`) or disabled (`OFF`).

3.3.2.3.5 `ENDFUNCTION` *name*

The `ENDFUNCTION` *name* keyword marks the end of a `FUNCTION` specification. This keyword is always paired with a `FUNCTION` keyword.

3.3.2.4 Monitoring Function Specification

The keywords described in the following sections appear within a `FUNCTION` / `ENDFUNCTION` keyword pair. These keywords are required for a complete monitoring function specification.

3.3.2.4.1 `RATE` *rate*

The `RATE` *rate* keyword specifies the frequency with which `aird` executes the monitoring function. The *rate* argument is interpreted as number of minutes. If the function is meant to be an action type of routine rather than a monitoring function (for example, a function to restart a daemon when a `FAILED` status has been returned to a monitoring function), the argument for `RATE` should be set to `NONE`. This value prevents the function from being executed periodically while allowing it to be executed from other functions. Refer to Section 3.4, page 160, for more information about rate specification on the monitoring functions.

3.3.2.4.2 `EXECUTE` *file*

The `EXECUTE` *file* keyword specifies the absolute path name of the monitoring function that is to be executed.

3.3.2.4.3 `LOGFILE` *file*

The `LOGFILE` *file* keyword specifies the absolute path name of the file in which the function output is placed. If there is no output of interest from the specific monitoring function, set this argument to `NONE`.

3.3.2.4.4 `TIMEOUT` *time*

The `TIMEOUT` *time* keyword specifies the length of time that `aird` should wait for the return of the function. If this time is exceeded, `aird` kills the monitoring function and logs the abnormal termination in its binary log file. The *time* argument can be set to `NONE` to indicate that the function should never be timed out by `aird`.

3.3.2.4.5 `RETURN` *tag value action*

The `RETURN` *tag value action* keyword specifies the return values for the monitoring function. The return *tag* is a tag previously defined in a `TYPE` keyword specification and associated with text from the `MESSAGE` keyword specification. The *value* argument, to which the tag set in the environment is assigned, is an integer between 0 and 200. The *action* argument specifies the name of another function to be executed on the return of the specified tag value from the monitoring function.

The function identified in the action argument must also be defined using the function specifications within the current product specification. The action can also be set to `NONE`, indicating that no further action should be taken on the return of that tag from the monitoring function.

3.3.2.5 Installation Tool Configuration

The configuration of products and their monitoring functions span three interconnected menus in the installation tool:

```
M-> Configure system ->
      M-> AIR configuration ->
          M-> Product enable ->
          M-> Product functions ->
          M-> Function return configuration ->
```

Each of these menus is described below.

3.3.2.5.1 Product Enable Menu

This first menu describes the products and whether they are to be monitored. The menu consists of two fields: the name of the product, and whether it is enabled.

An example of the product enable menu follows:

```
              AIR Product enable
      Product Name     Product Enabled
      ------------     ---------------
E->   disk-integ       YES
      nqs              YES
      tapes            YES
      msgdaemon        YES
      tcp              YES
      urm              YES
```

The menu is used to determine the `PRODUCT` and `ENDPRODUCT` keywords in the configuration file.

3.3.2.5.2 Product Functions Menu

This menu contains seven fields that are used to determine the `FUNCTION`, `ENDFUNCTION`, `RATE`, `EXECUTE`, `LOGFILE`, and `TIMEOUT` keywords in the configuration file.

Here is a short example of the product functions menu, focusing on the URM product:

```
                                 AIR Product functions

    Prod Name Funct Name Enabled Rate  Command Log File                Timeout
     --------- ---------- ------- ----   ----------------------------   -------
E-> urm       function   YES     25    /usr/air/test/urm/urm.funct     NONE
    urm       response   YES     20    /usr/air/test/urm/urm.response  NONE
    urm       existence  YES      10    /usr/air/test/urm/urm.exist     NONE
    urm       restart    NO      NONE  /usr/air/test/urm/urm.restart   NONE
```

The `Prod Name` field associates these functions under a given product (there must be a product with this name in the `Product enable->` menu).

The `Funct Name` field (`FUNCTION`) names the function while the `Enabled` field indicates whether this monitoring function is on or off. The `Rate` field (`RATE`) indicates the time in minutes between executions of the monitoring command set in the `Command` field (`EXECUTE`). The `Log File` field (`LOGFILE`) is the name of a file where the output is to be placed (blank means no log file) and the `Timeout` field (`TIMEOUT`) indicates the maximum amount of time the monitoring function can take.

3.3.2.5.3 Function Return Configuration

This menu contains six fields that define the `RETURN` and `MESSAGE` keywords. An example showing the function return configuration for the URM product is as follows:

```
                           AIR Function return configuration

    Product Name Function Name Return Name  Return Value  Action   Return Message
    ------------ ------------- -----------  ------------  ------   --------------
E-> urm          function      PASSED       0                      Test Passed
    urm          function      FAILED       1                      Test Failed
    urm         response     FAILED      1                    Test Failed
    urm         response     PASSED      0                    Test Passed
    urm         existence    FAILED      1           restart  Test Failed
    urm         existence    PASSED      0                    Test Passed
    urm         restart      FAILED      1                    Test Failed
    urm         1restart     PASSED      0                    Test Passed
```

The `Product Name` and `Function Name` fields are used to associate the return information with a given product (must be a product by this name in the `Product enable->` menu) and function (must be a function by this name for this product defined in the `Product functions ->` menu).

Each function under a product must have a minimum of two return values, named `PASSED` and `FAILED`. These names correspond to an exit status (in this case `0` and `1` respectively) and to a return message text.

Depending on the return value for a function, an action can be performed. This action is specified by the `Action` field and corresponds to the name of another function defined for this product. In the example above, if a `FAILED` status is returned by the `existence` function of the `urm` product (exit status of `1` from `/usr/air/test/urm/urm.exist`), the `restart` function is then started by the `aird` process.

The `restart` function is a special function, since it is not used for monitoring purposes but rather for restarting the URM daemon. Since the restart function should only be started when the `existence` test fails, the `restart` function is configured with a `Rate` (`RATE`) of `NONE`. This indicates that this product is not to be started periodically. Note also that in the above example, the `restart` function is turned off. Therefore, if the `existence` function were to fail, the `restart` function would not be run because it is disabled.

3.3.3 Return Tags

This section contains an overview and summary of the `TYPE`, `MESSAGE`, and `RETURN` keywords and the associated arguments.

The `TYPE` keyword defines return tags and their associated types. The `TYPE` *tag* argument defines a variable to be used as an exit status by a monitoring function. `aird` sets this variable in the environment when it initially processes the configuration file. The *types* associated with each `TYPE` tag are associated with the environment variables that serve as exit statuses for the monitoring functions. The report generators use these types, associated with the tags, in their processing. Only the `PROD_AVAILABLE` and `PROD_UNAVAILABLE` types are used by the report generators at this time for availability determination.

The report generators also use the MESSAGE keyword, which associates a text message with a tag.

The RETURN keyword assigns the tag an environment variable and indicates whether further action should be taken following the return of a monitoring function.

3.3.4 Sample Configuration File

This section contains a sample configuration file and an interpretation of the contents. Read this section if you are unsure of the correct interpretation or if you want to check your understanding. Refer to this sample file as you read the text following the file.

```
# Start of Configuration File Generated by printcf on Thu Feb 21 15:31:40 1991
#
#       :%s./usr./sn1101/soft/os/crs
#
CONFIG  kernel_test_version
        #
        #       Define Coordinator logfile
        #
        COORD_LOG       /usr/spool/air/logs/coord.log
        #
        #       test directory
        #
        COORD_TESTDIR       /usr/air/test
        #
        #       Define Coordinator Heart Beat
        #
        COORD_HBEAT     10
        #
        #       Define Debug level
        #
        COORD_DEBUG     0
        #
        #       Define Binary output file name
        #
        COORD_BLOG      /usr/spool/air/logs/blog
        #
        #       Define ASCII logging level
        #
        COORD_LOGLEV    0
```

```
#
#       Define TYPES
#
TYPE    PASSED      PROD_AVAILABLE
TYPE    FAILED      PROD_UNAVAILABLE
TYPE    CHANGED     PROD_AVAILABLE WARN_ADMIN
TYPE    NODEVICE    PROD_AVAILABLE WARN_ADMIN WARN_OPS
TYPE    NORESERVE   PROD_AVAILABLE WARN_ADMIN
TYPE    NOMOUNT     PROD_UNAVAILABLE WARN_OPS WARN_ADMIN
TYPE    BADWRITE    PROD_UNAVAILABLE
TYPE    BADJOB      PROD_UNAVAILABLE
TYPE    QSUBFAILED  PROD_UNAVAILABLE
TYPE    AUDITERROR  PROD_AVAILABLE WARN_ADMIN
TYPE    TCPFAILED   PROD_UNAVAILABLE
TYPE    UDPFAILED   PROD_UNAVAILABLE
TYPE    ICMPFAILED  PROD_UNAVAILABLE
#
#       Define product disk-integ
#
PRODUCT disk-integ  ON
        MESSAGE FAILED  Test Failed
        MESSAGE PASSED  Test Passed
        #
        #       Define Function response of Product disk-integ
        #
        FUNCTION        response      ON
                RATE    1
                EXECUTE /usr/air/test/kern/kern.response
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     response
        #
        #       Define Function existence of Product disk-integ
        #
        FUNCTION        existence      ON
                RATE    15
                EXECUTE /usr/air/test/kern/kern.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
                ENDFUNCTION     existence
        ENDPRODUCT      disk-integ
       #
        #        #       Define product nqs
        #
        PRODUCT nqs  ON
                MESSAGE PASSED  Test Passed
                MESSAGE FAILED  Test Failed
                MESSAGE QSUBFAILED  Qsub failed during functional test.
                MESSAGE BADJOB  Returned job did not contain expected output
                #
                #       Define Function functional of Product nqs
                #
                FUNCTION        function      ON
                        RATE    15
                        EXECUTE /usr/air/test/nqs/nqs.funct
                        LOGFILE NONE
                        TIMEOUT                 10
                        RETURN  PASSED  0       NONE
                        RETURN  FAILED  1       NONE
                        RETURN  QSUBFAILED  2   NONE
                        RETURN  BADJOB      3   NONE
                ENDFUNCTION     function
                #
                #       Define Function response of Product nqs
                #
                FUNCTION        response      ON
                        RATE    10
                        EXECUTE /usr/air/test/nqs/nqs.response
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     response
                #
                #       Define Function existence of Product nqs
                #
                FUNCTION        existence      ON
                        RATE    5
                        EXECUTE /usr/air/test/nqs/nqs.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     existence
                #
                #       Define Function netexist of Product nqs
                #
                FUNCTION        netexist      ON
                        RATE    5
                        EXECUTE /usr/air/test/nqs/nqsnet.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     netexist
        ENDPRODUCT      nqs
        #
        #       Define product tapes
        #
        PRODUCT tapes  ON
                MESSAGE PASSED  Test Passed
                MESSAGE FAILED  Test Failed
                MESSAGE BADWRITE  Write to tape failed
                MESSAGE NOMOUNT   Mount of tape failed
                MESSAGE NORESERVE Reserve of tape failed
                MESSAGE NODEVICE  No devices available at start of test.
                #
                #       Define Function functional of Product tapes
                #
                FUNCTION        function      OFF
                        RATE    60
                        EXECUTE /usr/air/test/tapes/tape.funct
                        LOGFILE NONE
                        TIMEOUT                 10
                        RETURN  PASSED     0       NONE
                        RETURN  FAILED     1       NONE
                        RETURN  BADWRITE   2       NONE
                        RETURN  NOMOUNT    3       NONE
                        RETURN  NORESERVE  4       NONE
                        RETURN  NODEVICE   5       NONE
                ENDFUNCTION     function
                #
                #       Define Function response of Product tapes
                #
                FUNCTION        response      ON
```

```
                RATE    10
                EXECUTE /usr/air/test/tapes/tape.response
                LOGFILE NONE
                TIMEOUT                 1
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     response
        #
        #       Define Function existence of Product tapes
        #
        FUNCTION        existence      ON
                RATE    5
                EXECUTE /usr/air/test/tapes/tape.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     existence
        #
        #       Define Function avrexist of Product tapes
        #
        FUNCTION        avrexist      ON
                RATE    5
                EXECUTE /usr/air/test/tapes/tapeavr.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     avrexist
ENDPRODUCT      tapes
#
#       Define product msgdaemon
#
PRODUCT msgdaemon  ON
        MESSAGE PASSED  Test Passed
        MESSAGE FAILED  Test Failed
        #
        #       Define Function response of Product msgdaemon
        #
        FUNCTION        response      ON
                RATE    10
                EXECUTE /usr/air/test/msgd/msgd.response
                LOGFILE NONE
                TIMEOUT                 1
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     response
        #
        #       Define Function existence of Product msgdaemon
        #
        FUNCTION        existence      ON
                RATE    5
                EXECUTE /usr/air/test/msgd/msgd.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     existence
ENDPRODUCT      msgdaemon
#
#       Define product tcp
#
PRODUCT tcp  ON
        MESSAGE FAILED  Test Failed
        MESSAGE PASSED  Test Passed
        MESSAGE TCPFAILED Transmission Control Protocol failure
        MESSAGE UDPFAILED User Datagram Protocol failure
        MESSAGE ICMPFAILED Control Message Protocol failure
        #
        #       Define Function functional of Product tcp
        #
        FUNCTION        function      ON
                RATE    10
                EXECUTE /usr/air/test/tcp/tcp.funct
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  PASSED     0    NONE
                RETURN  FAILED     1    NONE
                RETURN  ICMPFAILED 2    NONE
                RETURN  UDPFAILED  3    NONE
                RETURN  TCPFAILED  4    NONE
        ENDFUNCTION     function
        #
        #       Define Function existence of Product tcp
        #
        FUNCTION        existence      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcp.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     existence
        #
        #       Define Function gatedexist of Product tcp
        #
        FUNCTION        gatedexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcpgated.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     gatedexist
        #
        #       Define Function lpdexist of Product tcp
        #
        FUNCTION        lpdexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcplpd.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     lpdexist
        #
        #       Define Function namedexist of Product tcp
        #
        FUNCTION        namedexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcpnamed.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     namedexist
        #
        #       Define Function ntpdexist of Product tcp
        #
```

```
        FUNCTION        ntpdexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcpntpd.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     ntpdexist
        #
        #       Define Function smailexist of Product tcp
        #
        FUNCTION        smailexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcpsmail.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     smailexist
        #
        #       Define Function snmpdexist of Product tcp
        #
        FUNCTION        snmpdexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcpsnmpd.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     snmpdexist
ENDPRODUCT      tcp
#
#       Define product urm
#
PRODUCT urm  ON
        MESSAGE PASSED  Test Passed
        MESSAGE FAILED  Test Failed
        #
        #       Define Function functional of Product urm
        #
        FUNCTION        function      ON
                RATE    25
                EXECUTE /usr/air/test/urm/urm.funct
```

```
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  PASSED  0       NONE
                        RETURN  FAILED  1       NONE
                ENDFUNCTION     function
                #
                #       Define Function response of Product urm
                #
                FUNCTION        response      ON
                        RATE    20
                        EXECUTE /usr/air/test/urm/urm.response
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     response
                #
                #       Define Function existence of Product urm
                #
                FUNCTION        existence      ON
                        RATE    10
                        EXECUTE /usr/air/test/urm/urm.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       restart
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     existence
                #
                #       Define Function restart of Product urm
                #
                FUNCTION        restart        OFF
                        RATE    NONE
                        EXECUTE /usr/air/test/urm/urm.restart
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     restart
        ENDPRODUCT      urm
ENDCONFIG       kernel_test_version
#
# End of Configuration File Generated by printcf on Thu Feb 21 15:31:40 1991
#
```

At the top of the configuration file, the global operational keywords are defined. The ASCII log and binary log files for the `aird` process are set to `/usr/spool/air/logs/coord.log` and `/usr/spool/air/logs/blog`, respectively; however, the ASCII and debug logging levels are both set to 0, which means that the `aird` ASCII log file should remain empty. The rate that the `aird` process logs its own heartbeat record is set to every 10 minutes.

Next, the return tags and associated types are defined. The required `PASSED` and `FAILED` tags are at the top of the specification block. Also, the `PROD_AVAILABLE` and `PROD_UNAVAILABLE` types associated with the various tags are specified.

The next component of the file is the product's specification, the first one being the disk-integ. Messages are assigned to the two required tags, `PASSED` and `FAILED`, and two functions, response and existence, are defined. The response function is configured to execute every 90 minutes. No log file or time-out limit is specified, and no subsequent action is defined for the two required return tags. The existence function is configured to execute every 15 minutes and has no output, time-out, or subsequent actions specified. The actual monitoring functions are found in `/usr/air/test/kern/kern.response` and `/usr/air/test/kern/kern.exist`, respectively.

3.3.5 Configuration File Tuning and Validation

You can change the contents of the configuration file by using the UNICOS Installation Menu System (installation tool) validation. You can validate your configuration through the `airckconf`(8) command prior to putting that file into production on your system. The `airckconf` command uses the same validation routines that the `aird` process uses on the contents of the configuration file.

The AIR configuration menu in the installation tool allows you to verify your configuration before activating it. Simply select the `Verify air configuration ...` action in the AIR configuration menu. This action will generate a test configuration file based on your selections and then run the `airckconf`(8) command on it.

An example of an area that you may need to change is the execution rates for the monitoring functions.

> **Note:** Before making changes to the execution rates of the functions, please read through Section 3.4, page 160, to determine the appropriate balance for your system.

Changing the rates is accomplished by editing the argument for the `RATE` keyword in a specific function (or the `Rate` field in the `Product functions ->` menu of the installation tool). For example, if you want the existence function for URM to run once every 10 minutes instead of the default of 5 minutes, you would change the `RATE` keyword argument from 5 to 10 in the existence function specification of the URM product specification.

Any associated keyword arguments can be changed in the same manner. To add functions or products, refer to Section 3.4.6, page 165.

3.4 Monitoring Functions

The *monitoring functions* are the actual product verification processes and are either shell scripts or executable binaries. The implementation of the functions for each monitored product follows a hierarchic philosophy. Several functions are specified for each monitored product, and they are differentiated by the cost of the resources they use and the aspects of the product that they test.

For example, a function that verifies that a process exists on the system would be low cost, and, thus, could be executed more frequently than a function that required a tape mount. However, the higher-cost function provides a better assurance of product verification. These issues must be balanced by the rate specification in the configuration file, and be configured for each site's specific needs.

This section discusses the product testing coverage provided by the monitoring functions. This section also describes functions that need to be configured on a site-by-site basis and the procedures for adding those functions and other products, to the AIR system.

Caution: Since the test scripts installed under `/usr/air/test` can be overwritten during a software update, local changes to the supplied monitoring functions should be made by copying the file to another directory (or renaming it) and then altering this copy. Be sure to update the configuration of AIR to point to the local copy of the monitor script.

Refer to the individual monitoring function man pages for more detailed information on the contents of the individual tests.

Monitoring functions are provided for TCP/IP, NQS, online tapes, URM, and disk-integ (general system checks). The functions for these products are divided into the following types determined by the aspect of the product they are testing:

- Existence
- Response
- Functional invocation

The *existence* functions are low-cost verification routines that check for processes on a system. The *response* functions are also lower-cost routines that cause a product to respond in some manner, but do not cause significant operational changes on a system. The *functional invocation* routines can be high-cost functions. These functions are responsible for causing the product to accomplish work in the same way a user would. The specified rates in the configuration file for each of these functions reflects the cost involved; the existence and response functions' rates are much higher than those for the functional invocation tests. Again, these rates are site-configurable.

The `airexist`(8) command verifies product component existence.

It is important to note that these functions are opaque objects in the AIR system. Although they are grouped as described previously, they are by no means restricted to those types. The basic structure of the `aird` process and monitoring functions allows maximum flexibility in monitoring the products.

The following sections describe the product testing coverage and the functions that can be configured for each product.

3.4.1 TCP/IP

Existence and functional monitoring functions are available for TCP/IP.

The existence functions use the `airexist` command for process verification. There are separate existence functions for each of the following processes:

- Internet services daemon (`inetd`)
- Gateway routing daemon (`gated`)
- Line printer daemon (`lpd`)
- Internet domain name server (`named`)
- Time synchronization daemon (`ntpd`)
- Mail daemon (`sendmail`)
- SNMP daemon (`snmpd`)

These daemons are checked separately so that a site can disable any of the functions that check processes not configured on their system.

The functional invocation test uses an enhanced version of `ping` to send `ECHO` packets to network hosts using the following protocols:

- Internet Control Message Protocol (ICMP)
- Internet Transmission Control Protocol (TCP/IP)
- Internet User Datagram Protocol (UDP)

Failure to receive the expected `REPLY` packet constitutes a failed status.

The existence functions for TCP/IP check for the following processes, which must be configured on the system:

Function	Process checked
`tcp.exist`	`inetd`
`tcpgated.exist`	`gated`
`tcplpd.exist`	`lpd`
`tcpnamed.exist`	`named`
`tcpntpd.exist`	`ntpd`
`tcpsmail.exist`	`sendmail`
`tcpsnmpd.exist`	`snmpd`

If you are not using one of the listed processes, you must disable the associated function in the configuration file. If you do not disable a function that checks for a process that does not exist, TCP/IP will always be reported as available 0%.

3.4.2 NQS

Existence, response, and functional monitoring functions are available for NQS.

The existence functions use the `airexist` command for process verification. There are separate existence functions for each of the following processes:

- Main NQS daemon (`nqsdaemon`)
- TCP/IP networking component (`netdaemon`)

These daemons are checked separately so that a site can disable any of the functions that check processes not configured on their system.

The response function uses the `nqsresp` command to verify that the `nqsdaemon` process is able to read its named pipe. The `nqsresp` command attempts to open the named pipe; if the open fails, this function returns a failed status (see `nqsresp`(8)).

The functional invocation test uses the NQS `qsub` command to submit a job. The test then verifies that the expected string is found in the job output file. This function must also be configured to use a valid enabled, started, and running batch queue.

The existence functions for NQS check for the following processes, which must be configured on the system:

Function	Process checked
`nqs.exist`	`nqsdaemon`
`nqsnet.exist`	`netdaemon`

If you are not using one of the listed processes, disable the associated function in the configuration file. If you do not disable a function that checks for a nonexistent process, NQS will always be reported as available 0%.

Because it invokes a generic `qsub`(1) command, the `nqs.funct` function monitoring test is dependent on an enabled, started, and running default batch queue. If you cannot ensure that a default batch queue is always available, edit `nqs.funct` so that the job is directed to an available queue by using the appropriate `qsub` options.

3.4.3 Online Tapes

Existence, response, and functional monitoring functions are available for online tapes.

The existence functions use the `airexist` command for process verification. There are separate existence functions for each of the following processes:

- Tape daemon (`tpdaemon`)
- Message daemon (`msgd`)
- AVR component (`avrproc`)

These daemons are checked separately so that a site can disable any of the functions that check processes not configured on their system.

The `tpstat`(1) command is a tape status command that forces a response from the `tpdaemon` process. The response function determines the ability of the `tpdaemon` to respond based on the return value from `tpstat`. The response function for the `msgd` process invokes the `msgd` command, and redetermines the ability for response based on the return value.

The function invocation test uses the `dd`(1) command to write a file to tape by using the tape daemon. This test serves as a template only. Due to the variance of device groups, label types, and volume serial numbers, each site needs to modify this script to reflect their configuration.

If you are not using AVR, disable the `tapeavr.exist` function in the AIR configuration file. If AVR is nonexistent and you do not disable `tapeavr.exist`, online tapes will always be reported as available 0%.

The `tape.funct` function monitoring test uses the `tpmnt`(1) command to mount a tape on a drive, and then uses the `dd` command to write data to that tape. Edit this test to reflect your local tape environment, paying particular attention to the `DEVGRP`, `LABEL`, and `VOLSER` variables and the time check at the top of the script. By default, this test executes between 10 A.M. and 4 P.M.; otherwise, it returns a passed exit status without execution.

3.4.4 Disk-integ

System monitoring functions that check existence and integrity are available for the kernel.

The existence script invokes various user commands, such as `cd`, `ls`, and `cat`, to verify that the kernel is working and to gain some idea of the interactive response time.

The integrity function checks the following:

- Unrecovered disk errors
- File system free space and inodes

This script is configured to send mail to the appropriate administrators with pertinent information.

3.4.5 URM

Existence, response, and functional monitoring functions are available for the Unified Resource Manager (URM).

The existence function uses the `airexist` command to determine if the `urmd` process exists in the system.

The `rmgr` command is a URM status command that forces a response from URM. The response function determines the ability of URM to respond based on the return value from `rmgr`.

The functional invocation test uses the URM `rmgr` command to send a `REGISTER` request to `urmd` to verify whether or not `urmd` is initialized and functioning. The URM product also has a special function called `restart`. The function is not a monitoring function, but rather, an action. The `existence` function defines it to be started in response to a `FAILED` exit status (the `urmd` process not running) in order to restart the `urmd` process. Since it is not a monitoring function, its `RATE` is set to `NONE` to indicate to `aird` that it should not be periodically started.

In the preceding example configuration, the `restart` function is turned off. In order for the `restart` function to be started by the `aird` process in response to the failure in `existence`, it must be turned on.

3.4.6 Adding Products and Functions

In the AIR system, you can easily extend product sets and enhance monitoring functions.

This section describes the addition of products and monitoring functions to the AIR system.

Follow these steps in order to add a product to the AIR system:

1. Create the appropriate monitoring functions for the product.
2. Create the appropriate directory in the test directory tree, and move the monitoring functions to that directory.
3. Add the product and monitoring function specifications to the configuration file.
4. Use the `airckconf` command on the newly edited configuration file to validate your changes.
5. Move the file into production; send a `SIGHUP` signal to the `aird` process.
6. Check to ensure that the added functions are working as expected.

3.4.6.1 Creating Functions

The first step in adding a product to the AIR system is creating the appropriate monitoring functions. This step is quite important because the accuracy and effectiveness of product monitoring depends on the quality of the monitoring functions. The creation method described in this section is recommended, but because the functions are opaque in the system, you may use any method that works for your site.

In this example, the data migration product and functions are added to the AIR system by using the same hierarchical implementation as the released functions. These functions are incomplete and meant only as examples; however, you can use them as the basis for monitoring the data migration facility.

The following example shows an existence function for the `dmdaemon` process:

```
#! /bin/sh
#
#       dmf.exist       Product DMF existence test.
#
#       Test Description:
#               This is an existence test for DMF.  It checks for the
#               existence of the following process:  dmdaemon.
#
#       Dependencies:
#               The environment variable PASSED must be set.
#               The environment variable FAILED must be set.
#               The airexist command is installed in /usr/air/bin.
#
PATH=$PATH:/usr/air/bin

airexist -u 0 dmdaemon
if [ $? -eq 1 ]
then
        EXITVAL=$PASSED
else
        EXITVAL=$FAILED
fi

exit ${EXITVAL}

#
#       End of product DMF existence test.
#
```

Functional monitoring of data migration could be accomplished in one of two ways:

- Attempting to migrate and restore a file to online tape media
- Attempting to migrate and restore a file to the MVS station

The following example shows the online tape function test; the MVS front-end function would be identical to this one except that the specified media would be the MVS front end:

```
#! /bin/sh
#
#       dmftape.funct   Product DMF online tape functional test.
#
#       Test Description:
#               This is a functional test for the online tape media
#               capability of the data migration facility.  Note that
#               this script, through the forced migration of a file,
#               calls for a tape mount and should be run at a rate
#               appropriate for the site operational personnel and tape
#               environment.
#
#       Dependencies:
#               The environment variable PASSED must be set.
#               The environment variable FAILED must be set.
#               The environment variable PUTFAIL must be set.
#               The environment variable NOMIGRATE must be set.
#               The environment variable GETFAIL must be set.
#               The MIGRATE_FILE must be defined and exist.
#
MIGRATE_FILE="site defined"
MEDIA_TYPE=1

#
#       Migrate the file.  Indicate that the tape media should be used.
#
dmput -p ${MEDIA_TYPE} MIGRATE_FILE
if [ $? -ne 0 ]
then
        exit ${PUTFAIL}
fi

#
```

```
#       Verify that the file has at least been premigrated.
#
if [ !(-M MIGRATE_FILE) ]
then
        exit ${NOMIGRATE}
fi

#
#       Restore the file.
#
dmget MIGRATE_FILE
if [ $? -ne 0 ]
then
        exit ${GETFAIL}
fi

#
#       Verify that the file has been returned.
#
if [ "‘ls -l MIGRATE_FILE | cut -c1‘" = "m" ]
then
        exit ${FAILED}
else
        exit ${PASSED}
fi

#
#       End of product DMF online tape functional test.
#
```

3.4.6.2 Integrating the Functions

After you have created the monitoring functions, you must create the appropriate directory in the test directory tree and place the new functions in that directory.

In this example, you would create the `/usr/air/test/dmf` directory. Then you would copy the `dmf.exist`, `dmftape.funct`, and `dmfmvs.funct` monitoring functions created in the previous section into that directory.

3.4.6.3 Configuring the Functions

After you create the monitoring functions and place them in the test directory, you must follow these steps to instruct the `aird` process what to do with these monitoring functions. (Refer to Section 3.3.2, page 141, for a discussion of the configuration file and information concerning the specific keywords and their associated arguments in the configuration language syntax. There is a section for users of the UNICOS Installation Menu System, as well as for those who manually update the configuration file.)

3.4.6.3.1 Manual Function Configuration

Use the following procedures to manually add new products/functions to the AIR configuration file.

1. This step is necessary only if you are not using the installation tool to configure AIR.

   Copy the configuration file, `/usr/air/config_file`, to a temporary file, as in the following example:

   ```
   $ cp /usr/air/config_file /tmp/tmp_config_file
   ```

2. Using a standard text editor, add the product and monitoring functions' specification to the file, in the following two-step process:

   a. Add the additional return types (`PUTFAIL`, `NOMIGRATE`, and `GETFAIL`) used in the functional tests to the `TYPES` definition area near the top of the configuration file, as in the following example:

   ```
   #
   #       Define TYPES
   #
   TYPE    PASSED      PROD_AVAILABLE
   TYPE    FAILED      PROD_UNAVAILABLE
   TYPE    CHANGED     PROD_AVAILABLE WARN_ADMIN
   TYPE    NODEVICE    PROD_AVAILABLE WARN_ADMIN WARN_OPS
   TYPE    NORESERVE   PROD_AVAILABLE WARN_ADMIN
   TYPE    NOMOUNT     PROD_UNAVAILABLE WARN_OPS WARN_ADMIN
   TYPE    BADWRITE    PROD_UNAVAILABLE
   TYPE    BADJOB      PROD_UNAVAILABLE
   TYPE    QSUBFAILED  PROD_UNAVAILABLE
   TYPE    AUDITERROR  PROD_AVAILABLE WARN_ADMIN
   TYPE    TCPFAILED   PROD_UNAVAILABLE
   TYPE    UDPFAILED   PROD_UNAVAILABLE
   ```

```
TYPE    ICMPFAILED  PROD_UNAVAILABLE
TYPE    PUTFAIL     PROD_UNAVAILABLE WARN_OPS WARN_ADM
TYPE    NOMIGRATE   PROD_UNAVAILABLE WARN_OPS WARN_ADM
TYPE    GETFAIL     PROD_UNAVAILABLE WARN_OPS WARN_ADM
```

Although the `WARN_OPS` and `WARN_ADMIN` return types are not implemented for use, they will be important pieces of information for the real-time evaluation of the monitored products.

b. Add the product and functions specification anywhere below the `TYPES` area, as in the following example:

```
ENDPRODUCT tcp
#
#       Define Product DMF
#
PRODUCT dmf     ON
   MESSAGE PASSED       Test Passed
   MESSAGE FAILED       Test Failed
   MESSAGE PUTFAIL      Migration of file failed
   MESSAGE NOMIGRATE    Expected migrated file not migrated
   MESSAGE GETFAIL      Restore of file failed
   #
   #    Define function existence of Product DMF
   #
   FUNCTION     existence       ON
        RATE    5
        EXECUTE /usr/air/test/dmf/dmf.exist
        LOGFILE NONE
        TIMEOUT NONE
        RETURN  PASSED 0        NONE
        RETURN  FAILED 1        NONE
   ENDFUNCTION  existence
   #
   #    Define function online functional of Product DMF
   #
   FUNCTION     tapefunct       ON
        RATE    60
        EXECUTE /usr/air/test/dmf/dmftape.funct
        LOGFILE NONE
        TIMEOUT 10
        RETURN  PASSED    0     NONE
        RETURN  FAILED    1     NONE
        RETURN  PUTFAIL   2     NONE
```

```
          RETURN  NOMIGRATE 3     NONE
          RETURN  GETFAIL   4     NONE
     ENDFUNCTION  tapefunct
     #
     #    Define function MVS functional of Product DMF
     #
     FUNCTION     mvsfunct        ON
          RATE    60
          EXECUTE /usr/air/test/dmf/dmfmvs.funct
          LOGFILE NONE
          TIMEOUT 10
          RETURN  PASSED    0     NONE
          RETURN  FAILED    1     NONE
          RETURN  PUTFAIL   2     NONE
          RETURN  NOMIGRATE 3     NONE
          RETURN  GETFAIL   4     NONE
     ENDFUNCTION  mvsfunct
  ENDPRODUCT dmf
```

3.4.6.4 Function Configuration through the Installation Tool

This section describes how to use the UNICOS Installation Menu System (installation tool) to add new products and functions to AIR.

1. Enter the `Return tags and types setup ->` submenu under the AIR configuration menu. Add the new return types (`PUTFAIL`, `NOMIGRATE`, and `GETFAIL`) used in the functional tests as follows:

```
                        AIR Return tags and types setup

     Tag Name     Tag Type          Type 1       Type 2       Type 3
     ----------   ----------------  ----------   ----------   ------
     ...
E->  PUTFAIL      PROD_UNAVAILABLE  WARN_OPS     WARN_ADM
     NOMIGRATE    PROD_UNAVAILABLE  WARN_OPS     WARN_ADM
     GETFAIL      PROD_UNAVAILABLE  WARN_OPS     WARN_ADM
     ...
```

2. Add the product to the product list in the `Product enable ->` menu:

```
                              AIR Product enable

                        Product Name     Product Enabled
                        ------------     ---------------
                        disk-integ       YES
                        nqs              YES
                        tapes            YES
                        msgdaemon        YES
                        tcp              YES
                        urm              YES
                  E->   dmf              YES
```

3. Create the three functions in the `Product functions ->` menu:

```
                            AIR Product functions

     Prod Name  Funct Name  Enabled  Rate  Command Log File                Timeout
     ---------  ----------  -------  ----  -----------------------------   -------
E->  dmf        mvsfunct    YES      60    /usr/air/test/dmf/dmfmvs.funct   10
     dmf        tapefunct   YES      60    /usr/air/test/dmf/dmftape.funct  10
     dmf        existence   YES      5     /usr/air/test/dmf/dmf.exist      NONE
```

4. Create the function return information through the `Function return configuration ->` menu:

```
                              AIR Function return configuration

    Product Name   Function Name   Return Name   Return Value   Action   Return Message
    ------------   -------------   -----------   ------------   ------   --------------
E-> dmf            tapefunct       FAILED        1                       Test Failed
    dmf            tapefunct       PASSE         0                       Test Passed
    dmf            tapefunct       PUTFAIL       2                       Migration of file failed
    dmf            tapefunct       NOMIGRATE     3                       Expected migrated file not migrated
    dmf            tapefunct       GETFAIL       4                       Restore of file failed
    dmf            mvsfunct        FAILED        1                       Test Failed
    dmf            mvsfunct        PASSED        0                       Test Passed
    dmf            mvsfunct        PUTFAIL       2                       Migration of file failed
    dmf            mvsfunct        NOMIGRATE     3                       Expected migrated file not migrated
    dmf            mvsfunct        GETFAIL       4                       Restore of file failed
    dmf            existence       FAILED        1                       Test Failed
    dmf            existence       PASSED        0                       Test Passed
```

3.4.6.5 Validating Configuration

All essential components are in place for monitoring a new product. Before placing the new configuration file into production, however, you should validate the changes and additions that have been made. The `airckconf`(8) command runs the configuration file through the same verification routines that the `aird` process uses during its internal processing initiation. The following sample command line verifies the temporary configuration file produced in the previous section:

```
$ airckconf tmp_config_file
```

If you are using the installation tool, perform the `Verify air configuration ...` action to verify that the changes made in the menu system are correct.

For more detailed information, see `airckconf`(8).

Note: Do not proceed to the next step until `airckconf` is executing without error on the new configuration file.

3.4.6.6 Production

Before copying the new configuration file over the current one, make a back-up copy of `/usr/air/config_file`. If you are using the installation tool to update the AIR configuration, the backup is not necessary, since backups are made automatically. After creating a back-up copy and copying in the new configuration file (activating the configuration if using the installation tool), either start the `aird` process, using the `/usr/air/bin/start_air` script, or, if `aird` is already running, send the `aird` process the `SIGHUP` signal. Use the `ps`(1) command to determine the process ID of `aird` and use the `kill`(1) command to send the `SIGHUP` signal to that *pid*, as in the following example:

```
$  ps -el | grep aird

0 S     0 12954     1  0  39  24   26236     98     561  -    0:07 aird

$  kill -1 pid
```

By sending the `SIGHUP` signal, you cause the `aird` process to break out of its processing loop and reread the configuration file, thus adding the changes to its internal work list.

3.4.6.7 Final Verification

Verify that the new monitoring functions are operating as expected. Wait until all added functions have executed several times; the time to wait depends on the configured execution rates for the functions. Use the `airdet`(8) command to examine the records that have been logged pertaining to those functions, as in the following example:

```
airdet -p dmf -dmt /usr/spool/air/logs/blog
```

If you receive messages, final verification of the process is complete.

3.5 Using the Report Generators

This section examines the use of the four report generator commands, `airprconf`(8), `airdet`(8), `airtsum`(8), and `airsum`(8). The section includes a discussion of the record types used as input for each generator, an explanation of the output from each generator, and an analysis of a binary log file that highlights how you can use these commands and how they interact.

The availability numbers produced by the AIR report generators indicate the length of time a monitored product is available to a monitoring function. Whether the reported availability is the true availability depends on the quality of the monitoring functions and the AIR configuration. For example, if you configure your functional tests to run once a day, the numbers reported and the actual availability may be quite different for a product. Through careful crafting of the monitoring functions, and thorough configuring of the systems, accurate and detailed availability statistics can be obtained.

It is recommended that you read the man pages for the four report generators before reading this section, and that you keep the text of the man pages available for reference.

3.5.1 Record Types

The four report generator commands provide extensive and tunable selection criteria for the presentation of the availability statistics from the data gathered by the automated incident reporting system (AIR). This data consists of the binary records logged by the `aird`(8) process into its binary log file.

The following record types make up the contents of the `aird` binary log file:

Record type	Description
Configuration header	When initiated, the `aird` process reads the configuration file and translates the contents of that file into its own internal processing worklist. After completing the file translation, the `aird` process logs a configuration header record that delineates the contents of the configuration file just processed. If the `aird` process receives a `SIGHUP` signal, it breaks out of its internal processing loop and rereads the configuration file. After processing the configuration file, the `aird` process then logs another configuration header record into its binary log file.
Event	Upon the return of each configured monitoring functions, the `aird` process logs an event structure denoting the product and function names, the start and end times of the function, and the return type structure as specified in the configuration file.
Heartbeat	At a configured rate, the `aird` process logs a heartbeat record from which subsequent AIR system availability can be determined.
Time-out	If a monitoring function exceeds the specified `TIMEOUT` value, the `aird` process kills the test and logs a message indicating the abnormal termination.

The following list contains the AIR commands and descriptions of the reports they provide:

Command	Description
`airprconf`	Reads the configuration header records logged by the `aird` process and prints the contents of the configuration files that the records represent
`airdet`	A detailed reporting mechanism that reads and prints event, heartbeat, and time-out records
`airtsum`	Collates event records and prints a summary of statistics for the configured monitoring functions

`airsum`	Prints statistics on the availability of each monitored product by using event and heartbeat records

3.5.2 Output

This section discusses the output information printed by each generator command.

> **Note:** Because the `airprconf` and `airdet` commands merely log available statistics and do no collation or summarization, no interpretation of the output they produce is needed. However, the reports produced by the `airtsum` and `airsum` commands do require interpretation and this section contains the explicit derivation for the numbers found in the reports.

3.5.2.1 Using the `airprconf` Command

The default report generated by the `airprconf`(8) command contains the following information:

- Time the configuration header record was written to the binary log file by the `aird` process
- Number of types, messages, products, and functions defined
- Mapping of values and tags to the types
- Messages and functions defined for each product

For an example of the default report, see `airprconf`(8). Refer to Section 3.3.2, page 141, for a discussion of the configuration file and its contents, and for further information regarding the return types and their mappings.

If the `-P` option is specified, `airprconf` prints the configuration as it appears in the configuration file as it was in that period. All information found in the original configuration file is printed. Refer to Section 3.3.2, page 141, when interpreting this output.

3.5.2.2 Using the `airdet` Command

The default report generated by the `airdet` command consists of all event, heartbeat, and time-out records found in the `aird` binary log file. For each record, the product and function names and the message type are shown. For an example of a default report, see `airdet`(8).

The options for `airdet` allow you to change the selection of records and the information displayed for each selected record.

The selection criteria contain the following capabilities:

- The `-b` and `-e` options let you specify the range of time within which the records must fall in order to be selected. This option is helpful if you want to analyze only certain time periods.
- The `-p` and `-f` options let you limit the printed records to those from the given product or function, respectively.
- The `-n` option lets you specify a time so that the only records printed are those whose elapsed time exceeds the specified time. This option provides a back-end implementation of *noticing* for the AIR data. *Noticing* is the capability to indicate, or provide notice, when a particular record's elapsed time has exceeded a specified time. This capability can be useful if you want to highlight periods of time when a system may have been experiencing performance problems.
- The `-T` and `-O` options let you isolate the specified return type tags. These options are helpful if you are searching for specific as well as general product status. Refer to Section 3.3.2, page 141, for an explanation of the return type tags and what they indicate about their associated products.

The information criteria contain the following capabilities:

- The `-l` option outputs the elapsed time of the given record. If `-l` is specified with the `-n` option, `airdet` verifies the specified status.
- The `-t` option outputs the time stamps (beginning and end) for the selected records.
- The `-m` option outputs the text associated with the return type found in the record. This further denotes the status of the selected record.
- The `-h` option provides headers for the record delineation.

You can use the options previously described in a myriad of ways to assist you in analyzing the data collected by the AIR system. The `airdet`(8) man page describes basic concepts and provides examples to help you easily analyze a binary log file.

The following usage tips may be useful:

- When attempting to observe a particular range of time, use the `-b` and `-e` options.

- When attempting to observe particular products or functions, use the `-p` or `-f` options, respectively.
- When attempting to isolate product status, use the `-T` and `-O` options, keying off the desired status type.
- When attempting to identify abnormal elapsed times, use the `-n` option. You can use either the `airtsum` or `airsum` command with the `-E` option if you need to isolate a certain period of time. This option prints summary reports for each configuration header encountered. The default action is to display only the summary information over the range of files specified on the command line and not denote the smaller samples contained within.

Note: If the `-E` option is specified, both commands generate much more information, depending on the number of `SIGHUP` signals, machine boots, and AIR system startups contained within the specified binary log file.

3.5.2.3 Using the `airtsum` Command

The `airtsum` command prints summary statistics for the monitoring functions. This section discusses each column of information available within this report and the options associated with those columns. Note that the first four columns contain the default information printed in these reports.

Column	Description
`Product Name`	Name of the monitored product. Each of the configured products is displayed.
`Function Name`	Name of the monitoring tests. The monitoring functions specified for each of the monitored products are displayed.
`Total Executed`	Number of monitoring tests that returned. This number is really a count of the records logged by the `aird` process on return of each specific monitoring function. This number reflects the number of times that a specific monitoring function was executed during the given sample time of AIR data.

`Total Time Tested`

Time over which a specific function was monitoring its product. This time begins at the start time of the first record logged for the given function, and ends at the end time of the last record logged for the given function.

`From Time/Until Time`

(Displayed using the `-S` option) Beginning and end times that a specific function was monitoring its product. These times are used in the calculation of the total time tested.

`Percent Time`

(Displayed using the `-p` option) Percentage of time that the product was monitored by the specific function. This percentage is calculated by dividing the total time the specific function tested the product by the total time that UNICOS was running. The total time that UNICOS was running is calculated by subtracting the end time of the last record read from the boot time of the system. The equations for these calculations are as follows:

%_tested = *total_time_tested* / *total_system_time*
total_system_time = *system_boot_time* - *last_end_time*

`Return Type/Number Returned`

(Displayed using the `-r` option) Breakdown of each return type for each monitoring function. Each return type and the number of times that each type was returned are displayed for each configured monitoring function. The return numbers indicate the number of records logged by the `aird` process that contained the specific return type.

In the following columns, intervals are determined by subtracting the end time of the previous record of the same function type from the start time of the current record:

Column	Description
`Long Interval`	(Displayed using the `-l` option) Longest period of time between executions of the specific monitoring function

`Short Interval`

(Displayed using the `-s` option) Shortest period of time between executions of the specific monitoring function

`Average Interval`

(Displayed using the `-a` option) Average period of time between executions of the specific monitoring function

`Configured Interval`

(Displayed using the `-c` option) Period of time that should transpire between the execution of the specific monitoring function, as specified in the configuration file

The `airtsum` and `airsum` commands also accept `-b` and `-e` options, which specify sample times. They are used in the same manner as described previously in the `airdet` discussion.

3.5.2.4 Using the `airsum` Command

The `airsum` command reports summary statistics on the availability of each of the monitored products. The formats of report information are described in the following sections.

3.5.2.4.1 Default Summary Information Section

The `Default Summary Information` section contains default output from the `airsum` command. Specifically, this section contains the following basic availability information for each of the monitored products:

Column	Description
`Product Name`	Name of the monitored product. Each of the configured products are displayed.
`Total Time Available`	Entire time that the monitored product was available to its monitoring functions.

`Total Time Unavailable`

Entire time that the monitored product was unavailable to its monitoring functions.

`Relative Percentage Available`

Percentage of time that the monitored product was available with respect to the total time that the `aird` process was available.

`Real Percentage Available`

Percentage of time that the monitored product was available with respect to the total time that the system was available.

3.5.2.4.2 Product Availability Breakdown Section

When the records in the `aird` binary log file are processed, the final step is to determine the availability state of each product. A product's *state* indicates whether it is available or unavailable, and also the time during which this status is in effect. The state is determined not only by a change in availability, but also by configuration restarts either through `SIGHUP` signals or system boots. The *availability breakdown* for a product is the complete record of any availability changes; this section of the report contains availability breakdowns for each product and can be printed using the `-B` option. The report contains the following columns of information:

Column	Description
`Product Name`	Name of the monitored product. Each of the configured products is displayed.
`Product Status`	Status of each of the product's states. The status can be either `available` or `unavailable`.

`From Time/Until Time`

Range of time that the specific product was in the current state.

3.5.2.4.3 Summary Information for Periods Section

You can print each column of information in the `Summary Information for Periods` section by using the appropriate option, or you can print all columns by using the `-A` option. The `airsum` command collects the statistics from the product availability state breakdown. The report contains the following columns of information:

Column	Description
`Product Name`	Name of the monitored product. Each configured product is displayed.
`Total Time Available`	(Displayed by using the `-t` option) Entire time that the monitored product was available to its monitoring functions.
`Total Time Unavailable`	(Displayed by using the `-T` option) Entire time that the monitored product was unavailable to its monitoring functions.
`Longest Period Available`	(Displayed by using the `-l` option) Longest period of time that the specific product was in the available state.
`Longest Period Unavailable`	(Displayed by using the `-L` option) Longest period of time that the specific product was in the unavailable state.
`Shortest Period Available`	(Displayed by using the `-s` option) Shortest period of time that the specific product was in the available state.
`Shortest Period Unavailable`	(Displayed by using the `-S` option) Shortest period of time that the specific product was in the unavailable state.

`Average Period Available`

(Displayed by using the `-m` option) Average period of time that the specific product was in the available state.

`Average Period Unavailable`

(Displayed by using the `-M` option) Average period of time that the specific product was in the unavailable state.

The `-a` and `-u` options let you specify the return type tags used in determining the availability of the monitored products. By default, the `airsum` command uses the `PROD_AVAILABLE` and `PROD_UNAVAILABLE` return type tags for the availability determination. For examples of all options, see `airsum`(8).

3.5.2.5 Log File Analysis

This section analyzes an `aird` binary log file to provide you with the basic concepts involved in determining product availability from the data collected by the AIR system.

The following steps lead you through the analysis of an `aird` binary log file.

The most direct way to determine the availability of each monitored product on your system is to create the default report generated from the `airsum` command, as in the following example command line:

```
airsum -h /usr/spool/air/logs/blog
```

The `-h` option provides headers for the printed information and the default report contains the availability numbers for each monitored product on the system, as well as for the `aird` process itself. The following output is produced by the previous command line:

```
*** Total Availability Summary ***

Summary Information

Product          Total Time     Total Time     Rel. Perc. Real Perc.
Name             Available      Unavailable    Available  Available
---------------- -------------- -------------- ---------- ----------
aird                   04:22:43       00:00:00        100         99
tcp e                  00:00:00       04:20:33          0          0
nqs                    00:00:00       04:20:33          0          0
tapes                  04:20:33       00:00:00         99         99
```

```
msgdaemon             04:15:33      00:00:00         97         97
urm                   04:15:33      00:00:00         97         97
disk-integ            04:08:03      00:00:00         94         94
--------------------------------------------------------------------
```

By interpreting this information, you can tell that the example binary log file spans approximately 4 hours (most log files will be much longer than this).

This summary report indicates that the `aird` process was available the entire time the system was up; thus the relative and real percentages are identical for all products. If, for any reason, the `aird` process had not been running during the entire system time contained in the sample, the relative and real percentages would differ.

The availability times vary greatly among the products shown as never unavailable. Products that are always available do not necessarily have the same availability time values for a given sample for the following reasons:

1. Each monitoring function is executed at its individual, and commonly different, rate, as specified within the configuration file.

2. The availability of the product is keyed off the time marks found in the records logged by the `aird` process when each respective function returns from execution.

Thus, a product showing a shorter availability time indicates that the product's monitoring functions are configured to execute at a slower rate than a product showing a longer availability time. The disparity depends on the sample size and the configured execution rate for a function; the disparity increases as sample sets decrease in size and execution rates increase.

The following `airtsum` command line illustrates these concepts:

```
% airtsum -h /usr/spool/air/logs/blog

*** Total Test Summary ***

Function Summary Information

Product          Function         Total    Total Time
Name             Name             Executed Tested
---------------- ---------------- -------- --------------
tcp              gatedexist             14       01:05:00
                 namedexist             14       01:05:00
                 ntpdexist              14       01:05:00
                 existence              14       01:05:00
                 smailexist           14       01:05:00
                 snmpdexist           14       01:05:00
                 lpdexist             14       01:05:00
                 function              7       01:00:00
nqs              existence            14       01:05:00
                 netexist             14       01:05:00
                 response              7       01:00:00
                 function              5       01:00:16
tapes            existence            14       01:05:00
                 avrexist             14       01:05:00
                 response              7       01:00:00
msgdaemon        response              7       01:00:00
                 existence            14       01:05:00
urm              response              7       01:00:00
                 existence            14       01:05:00
                 function              5       01:02:00
disk-integ       existence            14       01:05:00
                 function              5       01:02:00
-----------------------------------------------------------------------
```

The output from this command shows how varied testing can be for each product. The frequency of executions indicates how often the function has monitored the product. The kernel response test has been configured to execute at a very high frequency, as indicated by the high number of total executions. Also, because it has the highest frequency of execution, it also has the longest testing time accumulated. Remember that these times converge as the sample length is extended.

The testing times shown in the `airtsum` report and the availability times shown in the `airsum` output differ greatly (approximately 1 hour versus approximately 4 hours). The `airsum` report generator cause this variance by assigning a product's first state to the status returned by the first record pertaining to the product. This first state extends from the system boot time to the end time of the first record. In normal operations, the time between booting the system and logging the first record is negligible. In this example, approximately 1 hour previous to the time this snapshot of the binary log file was taken, the `mv_files` script was executed. Therefore, the binary log file contains information from only the last hour or so. The availabilities calculated by the `airsum` command, however, reflect the fact that the system was booted 4 hours ago. In cases such as this, where the time between the system boot and first record of the sample is no longer insignificant, you should gather all information by specifying the last binary log file, in addition to the current binary log file, on the report generator command lines.

The numbers in the airtsum report indicate the execution rate for each function and you verify the information by using the airprconf command, which prints the current configuration file (see Section 3.3.4, page 150 for a sample configuration file).

By observing the airtsum output and this file, you can see the results of the different execution rates specified in the file. For example, the kernel response function is set to a 1-minute execution rate while most of the other functions are 5 and 10 minutes. Refer to Section 3.3.2, page 141, for more information on using this file.

If you return to the summary report generated by the airsum command, you can see that, although NQS and TCP/IP are available, AIR marks them as unavailable. To understand this discrepancy, you could run airsum to create the product availability breakdown section, as follows:

```
% airsum -hB /usr/spool/air/logs/blog

*** Total Availability Summary ***

Product Availability Breakdown

Product    Product          From                 Until
Name       Status           Time                 Time
---------- ---------------- -------------------- ---------------------
aird  PROD_AVAILABLE   May  1 08:37:44 1991 May  1 13:00:27 1991
tcp        PROD_UNAVAILABLE May  1 08:37:44 1991 May  1 12:58:17 1991
nqs        PROD_UNAVAILABLE May  1 08:37:44 1991 May  1 12:58:17 1991
tapes      PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:58:17 1991
msgdaemon  PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:53:17 1991
urm        PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:53:17 1991
disk-integ PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:45:47 1991
```

In this example, the breakdown does not provide any additional information because the sample is too short for much change in states. However, for larger samples, this report allows you to view the availability breakdown for any products you may be examining. For example, if the airsum summary report indicates that the aird process is unavailable for some period of time, the product availability breakdown report would show the time that the product was available and unavailable.

Because this report has not revealed any additional clues, the next report to examine is that created by the airtsum command. In particular, it shows the functional breakdown in terms of return types or the status each function marked the product, as follows:

```
% airtsum -hr /usr/spool/air/logs/blog

*** Total Test Summary ***

Function Summary Information

Product    Function   Total    Return           Number   Total Time
Name       Name       Executed Type             Returned Tested
---------- ---------- -------- ---------------- -------- --------------
tcp        gatedexist       14 PROD_UNAVAILABLE       14       01:05:00
           namedexist       14 PROD_AVAILABLE         14       01:05:00
           ntpdexist        14 PROD_AVAILABLE         14       01:05:00
           existence        14 PROD_AVAILABLE         14       01:05:00
           smailexist       14 PROD_AVAILABLE         14       01:05:00
           snmpdexist       14 PROD_AVAILABLE         14       01:05:00
           lpdexist         14 PROD_AVAILABLE         14       01:05:00
           function          7 PROD_UNAVAILABLE        7       01:00:00
nqs        existence        14 PROD_AVAILABLE         14       01:05:00
           netexist         14 PROD_AVAILABLE         14       01:05:00
           response          7 PROD_AVAILABLE          7       01:00:00
           function          5 PROD_UNAVAILABLE        5       01:00:16
tapes      existence        14 PROD_AVAILABLE         14       01:05:00
           avrexist         14 PROD_AVAILABLE         14       01:05:00
           response          7 PROD_AVAILABLE          7       01:00:00
msgdaemon  response          7 PROD_AVAILABLE          7       01:00:00
           existence        14 PROD_AVAILABLE         14       01:05:00
urm        response          7 PROD_AVAILABLE          7       01:00:00
           existence        14 PROD_AVAILABLE         14       01:05:00
           function          5 PROD_AVAILABLE          5       01:02:00
disk-integ existence        14 PROD_AVAILABLE         14       01:05:00
           function          5 PROD_AVAILABLE          5       01:02:00
------------------------------------------------------------------------
```

This output shows that only the `gatedexist` function (not all of TCP/IP) was returning a status of unavailable. For NQS, the function test was failing.

If an existence test continually returns an unavailable status, ensure that the particular process the function is verifying is actually configured to exist on the system. In the example, the running system does not have the routing daemon configured. Thus, the `gatedexist` function should be disabled in the configuration file.

If you do not properly configure monitoring functions for your system, the tests might report incorrect information about the monitored products. Ensure that all configured functions are appropriate for your system so that you may quickly diagnose and correct errors.

Although the cause of the existence functions failures in the example has been identified, the cause of the NQS functional test failure is still unknown. Use the `airdet` command to produce more details on system activities. Use the following command line to print all records for the NQS function test:

```
% airdet -hm -p nqs -f function /usr/spool/air/logs/blog

Product Function Type of Message  Message
Name    Name                      Text
------- -------- ---------------- ---------------------------------------------
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
```

Obviously, the batch job submitted to NQS was returned and does not contain the expected output. At this point, you must log into the system and try to determine why the output was not appearing correctly.

Note: If you do not properly configure monitoring functions for your system, the tests might report incorrect information about the monitored products. Ensure that all configured functions are appropriate for your system so that you may quickly diagnose and correct errors.

3.5.2.6 Summary

In the example in the previous section, a short binary log file is analyzed. The `airsum` command is used to first look at the global availability statistics for each of the monitored products. This section discussed the concepts of real versus relative percentages as they were displayed in the report, and examined the derivation of the net system statistics. We also touched upon the reasons that the availability numbers did not always match other products of similar states, and used the `airtsum` and `airprconf` commands to convey the results of the variable execution rates. As an aside, we talked about the assumption made by the `airsum` command in calculating the total test time starting from the system boot time. We then went on to troubleshoot why various products were marked unavailable, and used the breakdown report generated by `airsum`, as well as the return type breakdown report generated by `airtsum`, in our examination of the failing products. We pointed out the importance of properly configuring the AIR system, and the results of an incorrect configuration. And finally we went on to use the `airdet` command to isolate the truly failing function and to determine the reason for that failure.