# EMDIS Communication System - ECS

# Revision History

| | | |
|---|---|---|
| Version 0 | 20/09/93 | Internal Draft. |
| Version 1.01 | 27/09/93 | "ECS 1.0 Beta 1": First INFORMIX based version. |
| Version 2.00 | 20/01/94 | "ECS 2.0 Beta 0": Major rewrite for NON-INFORMIX version. |
| Version 2.01 | 12/02/94 | "ECS 2.0 Beta 1": Intermediate release: Small rewrite with new chapters mainly on backup and archiving. |
| Version 2.03 | 29/03/94 | "ECS 2.0 Beta 3": Minor changes after beta test phase. |
| Version 2.04 | 05/07/94 | Minor modifications |
| Version 2.05 | 2007-12-20 | Reformatting, start using ISO date format, removed Unix dependencies, Mime, encryption. |
| Version 2.06 | 2014-12-14 | Clarification of the ECS node value domain. Enforce independence of application using ECS. Added Chapter 3.1 |
| Version 2.07 | 2015-07-10 | Remove requirement of PGP 2.6.1 compatibility of encryption keys and remove the required key length. |

1. Introduction ..... 1
   1.1. Purpose and Basic Design ..... 1
   1.2. Some Design Details ..... 2
   1.3. Basic Requirements ..... 2
   1.4. Major Changes with respect to Version 1 ..... 2
   1.5. References ..... 3
2. Design Concepts ..... 3
   2.1. Terminology ..... 3
   2.2. Message Sequencing ..... 4
   2.3. Connection Check ..... 5
   2.4. Backup, Recovery and Archiving ..... 6
3. Application Interfacing ..... 7
   3.1 General considerations ..... 7
   3.2. Processing Messages: Input Interface ..... 7
   3.3. Sending Messages: Output Interface ..... 8
4. Administration Reference ..... 9
   4.1. Administration ..... 9
   4.2. Configuration File ..... 14
5. Software Design ..... 16
   5.1. Message Format ..... 16
   5.2. Meta-Messages Format ..... 16
   5.3. Mail Subject String Format ..... 19
   5.4. Message States ..... 19
   5.5. Overview ECS Directories ..... 20
   5.6. Scripts, Modules and Libraries ..... 21
   5.7. ECS Database Files ..... 23
   5.8. ECS Mailboxes ..... 24
   5.9. ECS Semaphores ..... 25
   5.10. ECS Error And Log File Format ..... 25
6. Problems and Plans ..... 25
   6.1. Known Problems ..... 25
   6.2. Not Yet Implemented Items ..... 26
   6.3. Other Plans and Considerations ..... 26
7. Short Reference Page ..... 26

# Preface

This document is primarily intended as a reference for an application programmer who wants to use ECS. For those it is recommended to read chapters 1 to 4.

But since the document also explains the overall concepts in the initial chapters and implementation details in chapter 5, it can also be served as a Software Design Document.

This document in many places describes the orginal ECS implementation which was a package of C programs and shell scripts heavily depending on SCO Unix and its mail command. As of version 2.05 this original package was replaced with two freely available ECS software packages: ESTER and perlECS. However, the original implementation may still serve as an overall example for its successors.

# 1. Introduction

## 1.1. Purpose and Basic Design

The purpose of ECS is to provide a permanent, reliable, connection-oriented communication service for the automatic exchange of messages between computer applications. ECS only requires that accounts under which these applications are running can exchange electronic mail (e-mail) messages. ECS is also designed to be independent of the transportation technique actually used and similarly it is designed to be independent from the applications using it (e.g. independent from the syntax and semantics of the messages exchanged).

The main features added by ECS to the e-mail functionality are:

(1) It checks the connections regularly making sure that all nodes involved are 'on line'.

(2) It makes sure that all messages sent from a node A to another node B are made available for processing on node B in exactly the same order they were sent from node A.

(3) It will automatically request re-sends from another node if missing messages are detected.

(4) It seamlessly encrypts and decrypts messages.

ECS can be seen as a "stateful" layer between the application layer and the low level communication layer. The term "stateful" shall express that - unlike the stand-alone e-mail - this layer knows about the overall state of the communication channel. One could also say that ECS offers the "connection-oriented" service by adding some functions to the "connection-less" services offered by stand-alone e-mail[1].

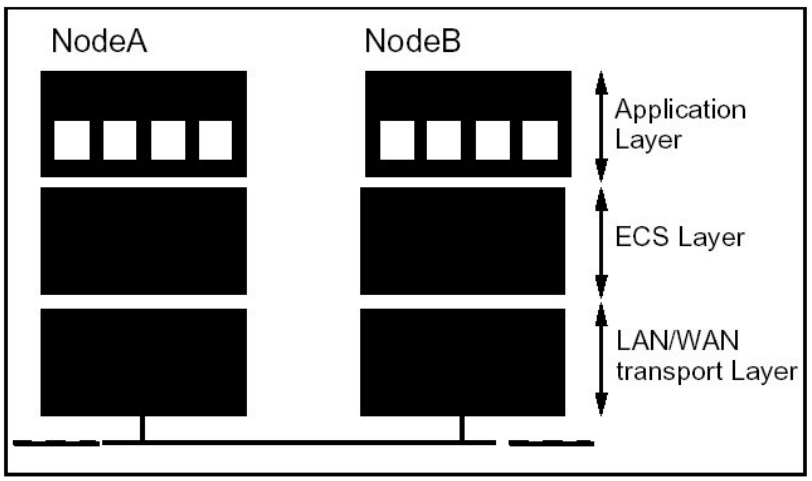



Figure 1: Layers

[1] As defined in Reference /4/

## 1.2. Some Design Details

ECS achieves correct message sequencing and failure recovery by assigning each message a unique number before sending it, by keeping track of these numbers and by keeping backups of the messages for some time. When receiving messages ECS checks this sequence number to make sure that the message is the next expected one. If not, it takes the appropriate actions as explained later. All these actions are done inside ECS, and are invisible to the application layer. Intervention from applications or the system administrator is not required. An application just "throws" a message to be sent at ECS, and ECS on the receiving side gives to the application level there only the message which is the next one to be processed.

Message sequencing is only applied to messages arriving from the same remote node. Messages arriving from different remote nodes are treated according to "first come, first served".

Connection check is done by two actions: First ECS makes sure that at least one message is sent to each remote node within a well-defined check interval. If the applications using ECS did not sent messages for a period longer than this interval, ECS automatically sends a "I am alive" message (which is some kind of an empty message that does not cause execution of applications on a remote node, but updates the ECS database there). Second it checks regularly that some message has arrived from each remote node in the ECS network within this check interval. The size of this check interval depends on the transport facility used, but it can be configured to be in the range from seconds to days (or even higher).

From the point of view of an application programmer ECS can be seen as an application programming interface (API). The calls and commands offered by ECS to the application layer, i.e. its "service access point", can be described as follows:

Output
: An application which wants to send a message to another node must first write this message onto a (temporary) file and must then execute an ECS function specifying the name of this file and the ECS node name of the receiving machine. This ECS function can be called directly, can be executed through system() call or from a shell script.

Input
: At ECS installation time it must be defined which command is to be executed to start an application module or script that processes an incoming message. An ECS daemon which regularly scans the mail queue will use this information.

## 1.3. Basic Requirements

ECS requires that the e-mail system is up and running properly on or between the machine(s) on which ECS is to be installed. ECS is currently distributed as part of the ESTER package (Windows) and as perlECS (Unix and Windows).

## 1.4. Major Changes with respect to Version 1

Unlike version 1.0, ECS 2.0 no longer contains INFORMIX 4GL modules and/or 4GL interfaces. 4GL applications are supposed to call the ECS functions by "RUN" commands.

### 1.5. References

/1/ S. Garfinkel and G. Spafford, "Practical Unix Security" (Unix Nutshell Series); O'Reilly & Associates, 1991

/2/ Tim O'Reilly and Grace Todino, "Managing UUCP and Usenet" (Unix Nutshell Series); O'Reilly & Associates, 1990

/3/ Thomas W. Madron, "LANs - Applications of IEEE/ANSI 802 Standards", John Wiley and Sons, 1989

/4/ Andrew S. Tanenbaum, "Computer-Netzwerke", Wolfram's Fachverlag, 1990

## 2. Design Concepts

### 2.1. Terminology

ECS Network: A set of participants running applications which exchange messages regularly and automatically.

ECS Node: One of the participants in an ECS network.

Message: A self contained set of information to be sent or received from a remote node for further processing. Seen from ECS a message is always a single file, therefore also sometimes also called "message file" in the following. (Seen from the application and in its terminology, such a message file may contain various "messages".)

Meta-message: Special type of messages generated internally by ECS to check the correct flow of information and to recover from problems. Meta-messages never reach the application layer. Meta-messages are sent in separate files which have an envelope that makes it possible to distinguish them from the other messages. Currently a meta-message can only contain a single piece of information. Meta-messages could also be seen as "out-of-band" messages.

Note that the term "meta" herein generally identifies the *additional* information *about* the information to be exchanged between the application programs. To clarify this difference the information or a message generated and processed on the application level is sometimes also called "real" or "regular" information or message.

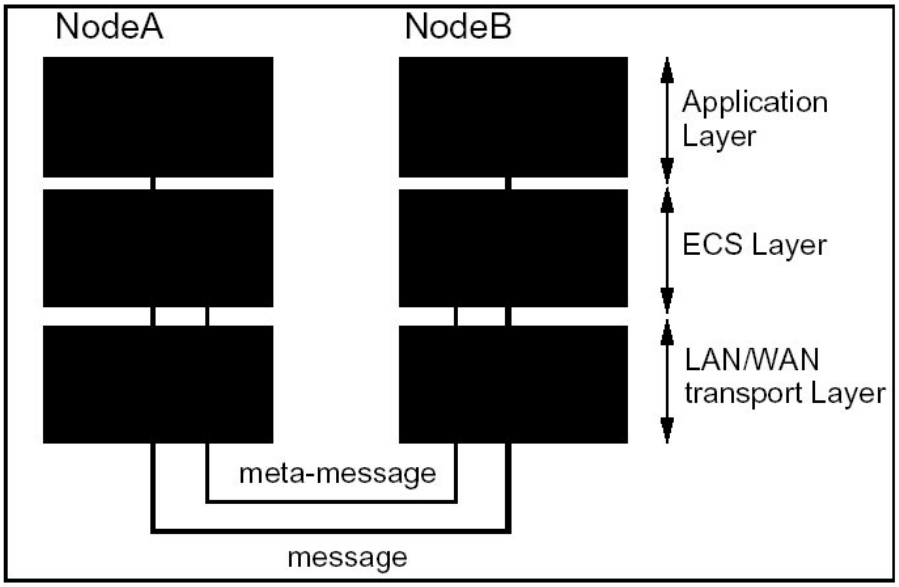



Figure 2: Messages and Meta-Messages

## 2.2. Message Sequencing

All messages sent from node A to node B are numbered sequentially. Messages to different nodes may have identical sequence numbers, but the information-triple

```
{sender, receiver, sequence number}
```

identifies uniquely every message in the ECS network at any time. And since within an ECS node the receiver is inherently known, the pair of sender and sequence number is sufficient to identify a message uniquely within a node. Every node expects that messages from a specific remote node are numbered sequentially and shall be processed in this order. However, they do not need to arrive in this order. So this sequence number allows for protection against missing or duplicate messages.

On arrival of a message, the receiving system performs several checks, some of them based on sequence number. Duplicate messages are ignored. If a message arrives ahead of time (a "fast" message), the gap in sequence numbers is interpreted as an indication of missing message(s). ECS will not allow application programs to process messages before the gap is filled. The fast message is stored and the process waits for the arrival of the message with the correct sequence number. If the missing message ("slow" message) does not arrive in a given time, the receiving node becomes active and sends the appropriate "re-send" meta-message(s) to the sending mode. When the slow message finally arrives processing continues as described before. Note that this protocol does not require explicit 'acknowledge' messages for each message arrived.

Note also that the sequence number assigned by ECS to each message is regarded as "meta-information". The application programs will (usually) never notice the sequence numbers and cannot control the assignment. If applications need information to identify specific messages, e.g. an "invoice number", this information must be part of the contents of the "real" message.

Meta-messages do not have a sequence number and this difference is used by ECS to mark and recognize message files as meta-message. This implies that meta-messages are not protected against loss. The functions based on exchange of meta-messages take this into account.

Backup copies of all message files are stored for some time by the sending node allowing resending of messages. Backup copies of all received message files must also be stored for some time to be able to handle system or disk crashes.

## 2.3. Connection Check

All nodes in the ECS network must send at least one message within a well-defined interval regularly to each other node. In case one did not have to send a "real" message for some time it must send an "I am alive" meta-message to the remote node. Received messages (or meta-messages) of any kind cause the update of the "last_message_received_time" for the corresponding remote node. All nodes must check regularly (in the same check interval) whether any of the last_message_received_time is "older" than the interval mentioned. If this is true, it must be assumed that the remote "is silent", i.e. that a system and/or communication breakdown has occurred, and the local administrator is notified.

There is no need to synchronize the nodes. The only requirement is that they all use the same "ECS check interval" $T_{chk}$. The longest time until a "silent" node is detected, would be approximately $2*T_{chk}$, the shortest $1*T_{chk}$. See figure.

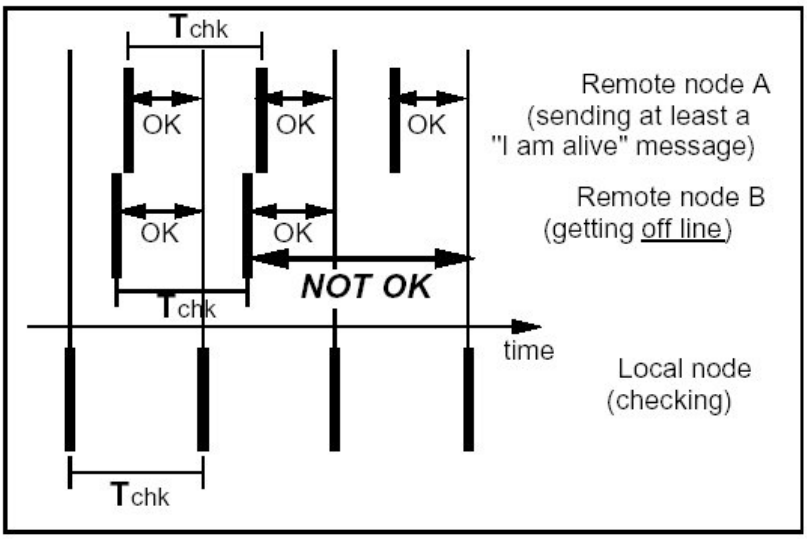



Figure 3: Check Interval

To avoid false alarms, e.g. due to a message which is just being processed while the connection check is executed, the check for a "silent node" will be based on 2 check intervals rather than one. This implies that the longest time until a "silent" node is detected will be $3*T_{chk}$, the

shortest $2*T_{chk}$. Checks for missing messages (see below) causing a "RE_SEND" request will be based on $3*T_{chk}$. However, the minimal period after which a node must issue at least a "I am alive" message, is $1*T_{chk}$. All the criteria mentioned will be applied with a tolerance of 10% "towards the safe side", i.e. the effective factors are 2.2, 3.3 and 0.9.

## 2.4. Backup, Recovery and Archiving

ECS provides basic facilities for archiving, backup and recovery. The first line of defence is that all outgoing ECS messages are stored in special mailboxes ("out-mboxes") and that all incoming messages are stored on the default disk in another special mailbox ("in-mbox") before they are actually processed.

A special problem arises when one of the nodes crashes and needs to be recovered. Assuming that backups with the state just before the crash are not available (otherwise Murphy's law would be violated) one should take measures that one can reconstruct the state before the crash from an older backup plus younger data which are (hopefully) not affected by the crash.

The most convenient recovery mechanism would be to rely on the remote partners - assuming that they are not likely to crash at the same time - and to ask the other nodes to send these messages again, which have been sent between the last backup and the crash. In fact an ECS system which was restored from the backup would request RE_SENDs automatically as soon as it sees the first message after the restore, because this would look like a "fast" message. But this type of recovery will only work if one of the two conditions is fulfilled:

(1) There are only two nodes in the network.

(2) The messages exchanged are of such a nature that the order in which messages from different hosts are processed is NOT significant, i.e. that any pair of messages

```
<Node_A:N>
<Node_B:M>
```

originating from different nodes can be processed in any order (Messages coming from the same host are still processed in the correct order). One could call such a system "time-invariant".

The second condition is only fulfilled if the content of the messages is "report-like": It does for example not matter if partner A tells me his phone number before partner B does. But if the messages include some kind of transaction, the order is significant. If A ordered the last hotel room before B did, the restored system must not send a denial to A simply because B happened to answer the RE_SEND request earlier.

A theoretical solution for a non-time-invariant system would be require

(1) that every message is acknowledged,

(2) that the acknowledge is sent with a processing order number,

(3) that each RE_SEND answer contains the processing order number,

(4) that the processing order number is encrypted so that partners answering RE_SENDs cannot cheat.

And still then these ACK's could get lost making it impossible to reconstruct who came first before the crash.

A practical solution for a non-time-invariant system: Copies of incoming messages must be stored on a separate device (another) or on another system. Shortly: on something that is not affected by a system crash or disk failure. (However, this does not protect against fire, EMP, burglary or "rm -r *" from root by root.) For this purpose ECS allows to configure a backup directory, which should be mounted on a separate disk. If configured copies of incoming messages will be stored there.

Additionally ECS provides the following facilities:

(1) "MSG_ACK" meta-message: By this meta message a system can tell other systems up to which number its messages have been received, processed and (!) backed up. Note that this is an ACK for a range of messages, and that it does not imply the acceptance of the contents of any message by the higher layer.

(2) A tool that removes all messages from the out-mboxes with sequence numbers less than or equal to the last acknowledged sequence number. This can be run regularly from a scheduler like e.g. cron to keep the mboxes small.

(3) A tool that archives ECS log and error files as well as the in-mbox on an external archive such as a tape and subsequently truncates these files and mailboxes. This also can be run regularly from a scheduler to keep these files small.

# 3. Application Interfacing

## 3.1 General considerations

ECS' Service Access Point can be divided into an input interface and an output interface. Both interfaces must not rely on any information from the application using it nor must the application rely on any information to be provided from ECS (see also Purpose and Basic design (1.1)). Namely, in the case of EMDIS using ECS as communication layer, it cannot be guaranteed that the ECS node designation is identical with the EMDIS registry code (formerly known as EMDIS hub).

## 3.2. Processing Messages: Input Interface

The input interface is defined by a command by which the ECS daemon can call a script or program that processes a message. This script or program must obviously be provided by the application programmer, and it must follow the following syntax:

```
<application> <file> <sender> <seq_num>
```

| | |
|---|---|
| application | The script or executable provided by the application programmer. This is defined by the full path name in the ECS configuration file and will be executed by the ECS daemon whenever a message is to be processed. |
| file | The name of a file containing the message to be processed. This will usually be a file in the tmp sub-directory within the ECS directory. This file will be deleted by the ECS daemon after the application has exited. Therefore if the application does not immediately process the file or if it needs a backup copy for what reason ever, it is the application's task to create such a copy. However, ECS itself maintains copies of all ECS messages for internal purposes and these copies will also be accessible in emergency cases. |
| sender | The name of the NODE sending the message. |
| seq_num | The sequence number of this message. This is only supplied as additional information and allows the application programmer to trace messages back in case of problems or to control ECS. But note that an application sending a message via ECS does not have control over the sequence number, and therefore the receiving applications should regard this information really only as meta-information. |

## 3.3. Sending Messages: Output Interface

The output interface provides facilities to the application level to send a file to a remote ECS node. These facilities consist of a function call (exampes in given in C) and a command that can be executed from the command line.

C:

```
int send_msg( char *file_name, char *node_name, char
*flags )
```

Command line:

```
ecstool -s <file_name> <node_name> {flags}
```

or

```
ecs_send_msg <file_name> <node_name> {flags}
```

| | |
|---|---|
| file_name | A string containing the name of the file with the message to be sent. |
| node_name | A string containing the ECS node name of the receiving node. |
| flags | A string containing a sequence of single character flags such as "U" for 'urgent' or "R" for 'Reverse Charge'[2]. |

Both commands are equivalent.

[2] In the current version these flags are just ignored.

# 4. Administration Reference

## 4.1. Administration

ECS will generate mail to the local administrator whenever it detects a problem. Independent of this, the administrator should from time to time check the two files generated continuously by ECS, especially the error file usually called "ecs.err". In the log file "ecs.log" s/he will find additional information. Note that ECS error messages are **not** going to standard output or error output, but to the err file!S/He should also check whether the ECS daemons are still running. This can be done by the

"ecs s" command.

Configuration changes can be made by editing the ECS configuration file "ecs.cfg" in the ECS directory. It is recommended to stop the daemons and also all message-generating applications whenever the configuration file is changed.

The usual administrative actions will be described in the following sections.

### 4.1.1. Look for Problems

First: the administrator should regularly look into the mailbox addressed to the ECS administrator - if this mailbox was not defined to be his/her usual mailbox.

Second: s/he should look into the ECS err file and – if required to get more information – also into the log file. This can be done by the following commands:

```
tail ecs.err
tail ecs.log
```

The location of these files depends on the actual configuration. The two files are written continuously by ECS. The "err" file contains only lines describing problems. The "log" files contains a history of all ECS actions.

Additionally the directory "mboxes" under the ECS_DAT_DIR should be visited and a few mailboxes should be inspected. First the mailbox "trash", which is supposed to keep obsolete copies of messages which arrived more than once. Use mail(C) to look into a mailbox:

```
mail -f trash
```

If you see a few messages there, and if you are sure that these messages have already been processed (look at the sequence number and compare with the "ecs state" output), just delete them. (See man pages of mail(C) for more.) If you see many messages there which have been received already, please tell your partner(s) and/or your system administrator to check the set-up, because it looks as if you receive messages twice or more.[3]

[3] If you see other messages, you have detected an ECS bug.

The second mailbox to be checked is "store", which should only contain "fast" messages or messages from unknown hosts.

```
mail -f store
```

"Fast" messages which are not listed in the "ecs state" output, can be deleted (although this should not happen). Messages from unknown hosts should be kept, because they will be processed as soon as the node is registered. But you may have to contact your partner(s) before you register such a new node.

### 4.1.2. ECS Start & Stop, State Display and Monitor

The ECS daemons are started and stopped by the command

```
ecs start        # Short form: ecs b
ecs stop         # Short form: ecs e
```

If one of these commands should unexpectedly fail, one can stop the daemons "by hand":

(1) Remove all files with the extension ".lck" from the directory $ECS_DAT_DIR. After at most T_SCAN and T_CHK seconds, the daemons should die.

(2) If you do not want to wait so long, run "ps -ef" to find out the PID of the daemons "ecs_scan_mail" and "ecs_chk_com". Then use these PIDs in the "kill -9" command. Be sure that you only kill the daemons belonging to you! Use the "ps -ef" to check the process and - sooner or later - all child processes are gone. In the real worst case, reboot the system. In any case, do not forget to remove the semaphore files ("ecs_scan_mail.lck" and "ecs_chk_com.lck" in ECS_DAT_DIR).

Note that the start and stop commands can also be issued by the applications based on ECS, and most likely in this applications' start/stop facilities, if such exist.

The current ECS state, i.e. the state of remote nodes and messages and whether the local daemons are running, is displayed by the command

```
ecs state        # Short form: ecs s
```

A continuous state display (monitor) can be started by

```
ecs mon          # Short form: ecs m
```

### 4.1.3. Add or Modify ECS nodes

The ECS tool commands

```
ecstool -a <node> <addr> <addr_r>
ecstool -m <node> <addr> <addr_r>
ecstool -d <node>
```

allow you to add a node to your ECS network, to modify the address(es) of an existing node or to delete a node from your network. Note the 'modify' option only affects the addresses, not the sequence numbers.

WARNING: Do not modify a node by deleting it first and adding it again, if you are not sure that your partner(s) are also re-initializing the sequence numbers simultaneously.

### 4.1.4. How to Keep ECS Running All The Time

In order to make sure that ECS is running all the time, e.g. after a system reboot or if some daemons crashed, one can add an entry to the crontab file of the account running ECS, which would call a script that does the following:

(1) Check the existence of the ecs_off lock. If this exists, just exit, because ECS is switched off.

(2) Check the ECS state. If ECS seems to be off or sick, restart it.

A script fragment like the following will do the job (Note that this script probably needs to source the .profile first to have all environment variables set properly.)

```
$ECS_BIN_DIR/ecs -s state; stat=$?
if [ $stat -eq 0 ]
then
      exit 0
fi
$ECS_BIN_DIR/ecs stop
$ECS_BIN_DIR/ecs start
$ECS_BIN_DIR/ecs -s state; stat=$?
if [ $stat -ne 0 ]
then
      # Still not running! Create and mail an error
      message exit 1
fi
exit 0
```

### 4.1.5. Emergency Actions

In special cases the administrator may have to request the re_send of a message by hand. This can be done by the command

```
ecs_resend_req <node> <seq>
```

Note that this re_send request is not registered (counted) in the ECS message table.

In rare cases when you run out of disk space, you may want to get rid of much junk in some mailboxes. First you should shut down ECS and then run "ecs_archive". Then you can delete everything in the mailboxes "active" and "store", because the system will request RE_SENDs from other nodes if required. You should NOT remove or delete messages in the out_XX mailboxes. You can also delete messages in the "out" mailbox, after you have read them.

In very rare cases you may want to change the contents of the ECS tables. You can do this by the export and import feature of "ecstool". But you should exactly know what you are doing, i.e. you should have a good understanding of ECS when you plan to do so. However, here are the steps you would have to do:

```
ecs stop
cd $ECS_DAT_DIR
ecstool –E
vi *.asc
ecstool –I
ecs start
```

The content of the asc file is a 1:1 dump of the contents of the .dat files.

### 4.1.6. Archive ECS Log and Error Files

The tool

```
ecs_archive
```

archives the ECS log file, the ECS error file and the "in" mailbox in a special sub-directory under the ECS_DAT_DIR and subsequently truncates the actual files. It will also store ASCII dumps of the database contents. The files will be stored in the archive directory under the names

```
ecs.log_YYYYMMDDhhmmss.Z
ecs.err_YYYYMMDDhhmmss.Z
in_YYYYMMDDhhmmss
ecs_node_tbl.asc_YYYYMMDDhhmmss.Z
ecs_msg_tbl.asc_YYYYMMDDhhmmss.Z
```

where "YYYYMMDDhhmmss" is a timestamp allowing it to store any number of archived pieces of a file. Note also that the log and error files are ASCII files and are compressed. The mailbox files are not compressed to allow easy access by mail(C) for recovery purposes.

It is proposed to run this tool regularly, e.g. every night, from cron(C). But note that this script will immediately exit if the ECS daemons are running. Therefore the appropriate sequence in a script doing this job and called by cron would look like

```
$ECS_BIN_DIR/ecs -s stop
$ECS_BIN_DIR/ecs_archive
$ECS_BIN_DIR/ecs -s start
```

### 4.1.7. External Archives

From time to time (e.g. once a week) all archived files should be written to an external archive (on another disk or onto tape). This can also be done by ecs_archive or by some other tools. If the backup is successful – but only if it is successful – some more actions are to be performed:

(1) The sending of MSG_ACK meta-messages should be triggered which tell each remote host, up to which sequence number its messages have been received, processed and backed up(!). Note that the term "acknowledgement of processing" does NOT imply that the message was processed successfully or that the contents of the message were accepted. This is a problem of the upper layers.

(2) The local out-mailboxes should be cleaned.

(3) The copy of the in-mailbox in the backup directory should be removed now to gain disk space for the next messages to come in. The backup copy can be removed now, because there is another copy on tape.

ECS provides facilities for these purposes.

Here the command sequence that should be put into a script triggered by cron, if the external archive facility of ecs_archive is used. Note that ecs_archive uses tar(C).

```
$ECS_BIN_DIR/ecs -s stop
$ECS_BIN_DIR/ecs_archive -X /dev/rStp0
if [ $? -ne 0]
then
      # error message
else
      $ECS_BIN_DIR/ecs_ack_all
      $ECS_BIN_DIR/ecs_clean_outboxes
fi
$ECS_BIN_DIR/ecs -s start
```

The -X option can also be used to write to a regular file which is then backed up by other means. Example:

```
$ECS_BIN_DIR/ecs -s stop
$ECS_BIN_DIR/ecs_archive -X /second_disk/ecs_arc.tar
if [ $? -ne 0]
then
      # error message
else
      $ECS_BIN_DIR/ecs_ack_all
      $ECS_BIN_DIR/ecs_clean_outboxes
fi
$ECS_BIN_DIR/ecs -s start
```

Additionally, the archive directory can be backed up by a special procedure and/or a part of a bigger backup. In this case one can use the -Y option of ecs_archive which tells this script that an external backup was done by some other means.

```
$ECS_BIN_DIR/ecs -s stop
#  Do the big backup now here
if [ $? -ne 0]
then
       # error message
else
       $ECS_BIN_DIR/ecs_archive –Y
       $ECS_BIN_DIR/ecs_ack_all
       $ECS_BIN_DIR/ecs_clean_outboxes
fi
$ECS_BIN_DIR/ecs -s start
```

#### 4.1.8. Other Administrative Tasks

The administrator should also do the following from time to time:

(1) Check the ECS error file

(2) Check the size/contents of the mailboxes in the "mbox" sub-directory and in the backup directory, if configured.

Whenever the administrator wants to change the configuration s/he can do so by using the script "ecs_config" again or edit the configuration file directly with a normal text editor. But in any case the ECS daemons must be stopped first!

## 4.2. Configuration File

The configuration file is a plain text file stored in the ECS_DAT_DIR directory under the name "ecs.cfg". Note that neither location nor name of this file must be changed. It contains a line for each configuration entry in the following format:

```
<cfg_item> | <cfg_value> { | comments }
```

| | |
|---|---|
| cfg_item | Name of the configuration item. This name must not be changed! All names must be given in uppercase letters and must match exactly one of the names given below. |
| cfg_value | Value of the configuration item. This value might be changed by the administrator due to the needs of a particular installation. |
| comments | Space for some remarks by the administrator. Optional. |

The following list explains all configuration items:

MSG_PROC — The command that shall be executed by the ECS daemon to call an application processing a message. This is usually the full pathname of a script or a executable outside ECS, i.e. within your application. Specify only the full pathname, including possibly some options. Note that ECS will automatically append three arguments (not options) when executing this command: file, sender and sequence number.

MAIL_MRK — The string to be used as ECS mark in subject string. You must agree with all your ECS partners to use exactly the same string. The ECS daemon scanning incoming mail will only regard these messages as messages to be recognized and processed, if this mail mark is used.

THIS_NODE — The ECS node name of this node. All the participants in an ECS network must have different node names. This name is used to identify each other in the network. It is NOT the E-mail address. An ECS node name must be a non-empty string of up to eight characters.

T_CHK — The time interval in seconds between two consecutive connection checks. It depends on your applications. As a rule of thumb choose a value twice or three times as long as the longest interval which is used on average to exchange mail. If you have for example partners in your network which exchange E-mail via uucp, and uucp is set up such that they try to contact each other on average every hour, then use a value of two or three hours for T_CHK.

T_SCN — The time interval in seconds between two consecutive scans of the mail queue for incoming mail. As a rule of thumb choose a value which is 50% longer than the shortest interval which is used to exchange mail. If you have for example partners in your network which exchange E-mail via SMTP, and the delivery channel for uucp is activated every minute, a value of 90 seconds seems appropriate.

ERR_FILE — The (full) pathname of the ECS error file. By default a file within the ECS directory is used, but if you have defined a directory where you collect all the log and/or error files of your application, it is recommended to store the ECS errors there, too, as "ecs.err".

LOG_FILE — The (full) pathname of the ECS log file. By default a file within the ECS directory is used, but if you have defined a directory where you collect all the log and/or error files of your application, it is recommended to store the ECS errors there, too, as "ecs.log".

ADM_ADDR — The E-mail address of the administrator. This address will be used by ECS to report problems. It is recommended to choose an address different from the address used for the ECS messages, but it is not required. However, if these warning messages are not read (and deleted), the ECS daemon might at the end be absorbed by finding real ECS messages among the thousands of warnings it has generated.

M_MSG_PROC — The command that shall be executed by the ECS daemon to call an application processing a meta-message. Note that this application is usually provided by ECS and therefore the default value given here should usually not be changed.

BCK_DIR — The directory under which ECS can store additional copies of incoming messages for backup purposes. If you do not need such a backup copy, specify "NONE" (in uppercase without quotes). Remember that a backup directory only makes sense if you install it on another disk different from the one ECS is installed on, or on a special floppy.

ACK_THRES — This value defines a sequence number threshold that shall be applied when copies of messages are to be deleted from the out-mailboxes. If this value is zero, the process cleaning up the out-mailboxes will delete all messages up the number acknowledged by the remote system. If it is non zero, only the messages up to the number acknowledged minus this value are deleted. So messages are still stored although they could theoretically be removed. It is recommended to put a non-zero value in to gain additional security. The value should be in the order of the number of messages you send to a single node during a day or a few days.

# 5. Software Design

## 5.1. Message Format

As already mentioned, ECS does not make any assumption about the format of the messages exchanged via ECS. However, since this version of ECS is based on standard e-mail, all messages must respect the following restrictions:

(1) Messages must consist of printable ASCII characters only (plain text, see RFC2822), i.e. content type encodings such as Mime are not supported.

(2) Line width must not exceed 80 characters.

(3) OpenPGP compatible public/private key pairs are used for encryption. If encryption is configured for a node, all messages have to be encrypted and signed, meta-messages only have to be signed (for an example see READY meta-message below).

## 5.2. Meta-Messages Format

In this version the format is intentionally simple and unsophisticated. It is line oriented and each message starts with a line

```
msg_type=<message type identifier>
```

Additional lines may contain additional information in the form

```
item = value
```

The remainder of this chapter describes the meta-messages currently implemented:

### 5.2.1. Meta-message READY

Format:

```
msg_type=READY
last_recv_num=XXX
last_sent_num=XXX
```

Purpose:

Sent by a node A to a node B if A did not have "regular" mail to send for a long time. Tells the node B that A is still alive, but simply has no work for him. Inform about the last sent and received message in order to detect missing messages between nodes with low message frequency.

Example:

```
-----BEGIN PGP SIGNED MESSAGE-----

msg_type=READY
last_recv_num=69425
last_sent_num=190934
# Hello Partner, I am alive.

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.6 (MingW32)

iQCVAwUBR11xEOj1jpXXV7mdAQF32gP9Frl9qTHh6ni/iWk9FwhM1lDQISSvTDr2
w21oHjqmX0+UWuluHzpnbMD4WsgrJoT+y+RVBqjCsd1Z++B72z9ODzxwx8D2HSIz
UaYogKyyShHswzEQbwEA8GjHP35gdtVOvvwNiM01IMp6yMPFoDhpTCvhigXMPSgB
wBNerpsbM/A=
=6kwI
-----END PGP SIGNATURE-----
```

### 5.2.2. Meta-message RE_SEND

Format:

```
msg_type=RE_SEND
seq_num=<seq_number>
```

Purpose:

Sent by a node A to a node B if A detected that the message from B with the specified number did not arrive, but should have arrived already, because messages from B with higher numbers were received.

Notes:

If the node receiving the RE_SEND message cannot satisfy the request, e.g. because the corresponding out_XX mailbox does not contain this message, a mail to the local administrator is generated.

### 5.2.3. Meta-message MSG_ACK

Format:

```
msg_type=MSG_ACK
seq_num=<seq_number>
```

Purpose:

Sent by a node A to a node B to tell node B that messages from node B up to sequence number <seq_number> have been received on A, processed and backed up.

## 5.3. Mail Subject String Format

Since ECS is using the ordinary mail facility, i.e. a message file is given to the mail system for delivery. The mail facility allows the transport of meta-information on the "envelope" of the message which can be used to simplify processing. The "envelope" of the message is just the subject line which can be specified with any mail.

For the current version of ECS the following format of the subject string was defined: It starts with a fixed string making it possible to distinguish an ECS message from any other mail just by looking at the "mail directory" (which shows the subject string)[4]. This is followed by the name of the sending node. Additional information in the subject line makes it possible to distinguish between messages and meta-messages. Meta-messages, by definition, do not have sequence numbers. All these subject string fields are separated by colons, but the string is defined to be blank-free.

Messages:

```
<ECS_MARK>:<sending_node>:<seq_num>
```

Meta-messages:

```
<ECS_MARK>:<sending_node>
```

## 5.4. Message States

Messages arrived on an ECS node can have three different states: arrived, accepted and stored. Every regular message takes these states in the following order:

Arrived: Whenever a message can been seen by ECS modules (not by an application program), it is called an "arrived" message. In the actual implementation, this implies that the mail was moved from the user's mailbox into the ECS mailbox "active" and is now waiting to be processed.

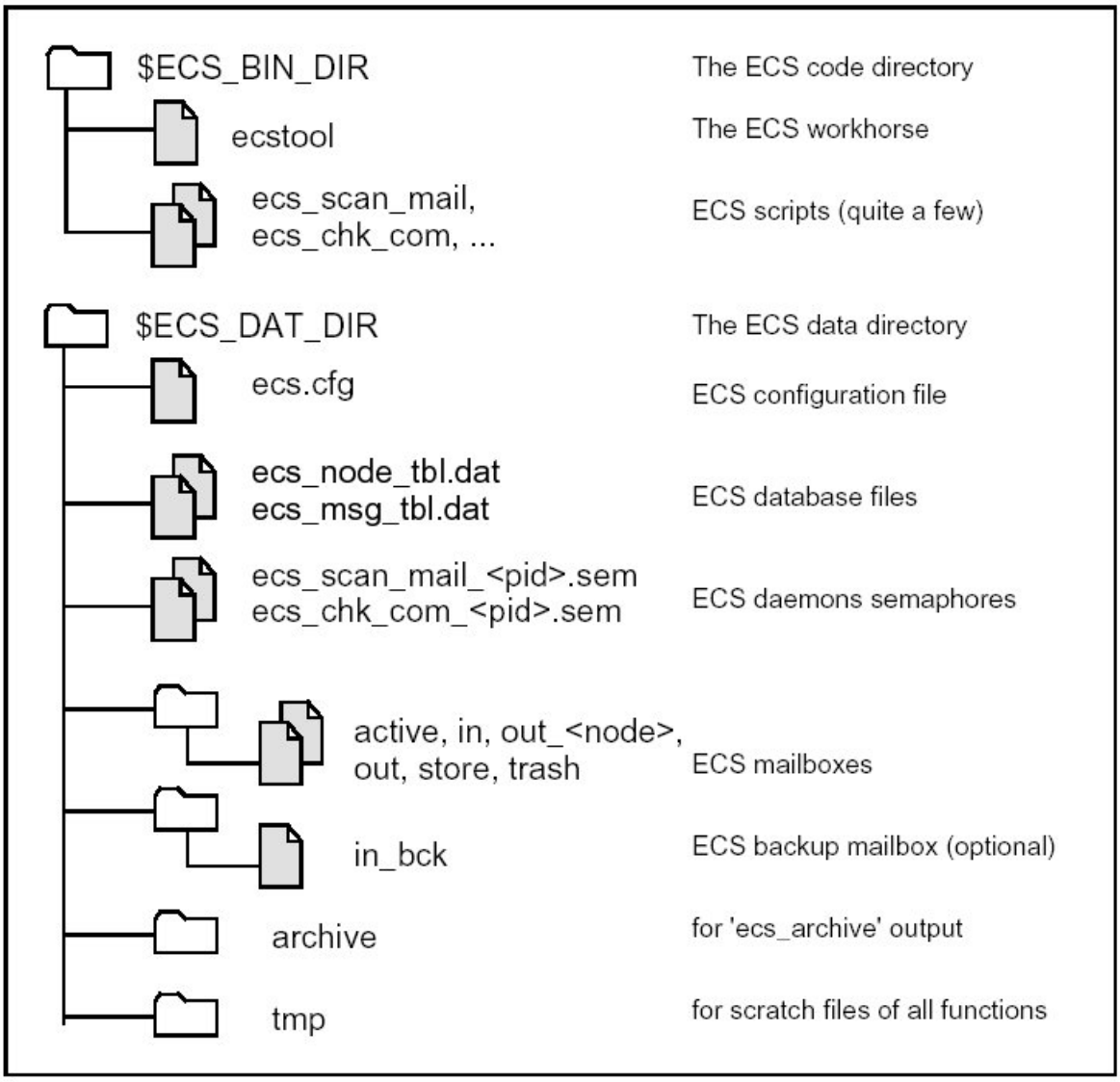
Accepted: A message is accepted, when its message number is one higher than the last message accepted. (To make this recursive definition complete: The first message which can be accepted must have the sequence number 1). A message which did not arrive in order is not accepted. If a message is accepted by ECS, the next action taken by ECS is to call the application program as defined by the configuration item "MSG_PROC" in order to process the message (after storing a copy of the message in the "in" mailbox and - if configured - also in the "in_bck" mailbox.)

Stored: A "fast" message, i.e. one that arrived too early, is stored in a special mailbox.

[4] This allows that the same system mailbox is used for different ECS networks.

## 5.5. Overview ECS Directories

The following figure gives an overview of the original ECS directories:




ECS uses two directories, one for data and one for binaries and scripts (and source code). There are two environment variables associated with these directories – ECS_BIN_DIR and ECS_DAT_DIR – which must contain the actual paths of these directories. The contents and the structure of these directories can be described as follows:

| | |
|---|---|
| ECS_BIN_DIR: | All executables, source code, scripts, Makefile etc. |
| ECS_DAT_DIR: | ECS configuration file and various sub-directories. |
| ECS_DAT_DIR/tmp: | A directory where all ECS modules can create scratch files. |
| ECS_DAT_DIR/mboxes: | A directory where ECS will maintain several mailboxes. |

ECS_DAT_DIR/archive: A directory where the tool "ecs-archive" can store pieces of archived files (log file, error file, in-mailbox)

## 5.6. Scripts, Modules and Libraries

The current version of ECS is a mixture of Bourne shell scripts and C code. The following tries to mention the most important items which can be found in the ECS development directory. More information can then be found in these files.

### 5.6.1. Scripts and Modules for Input

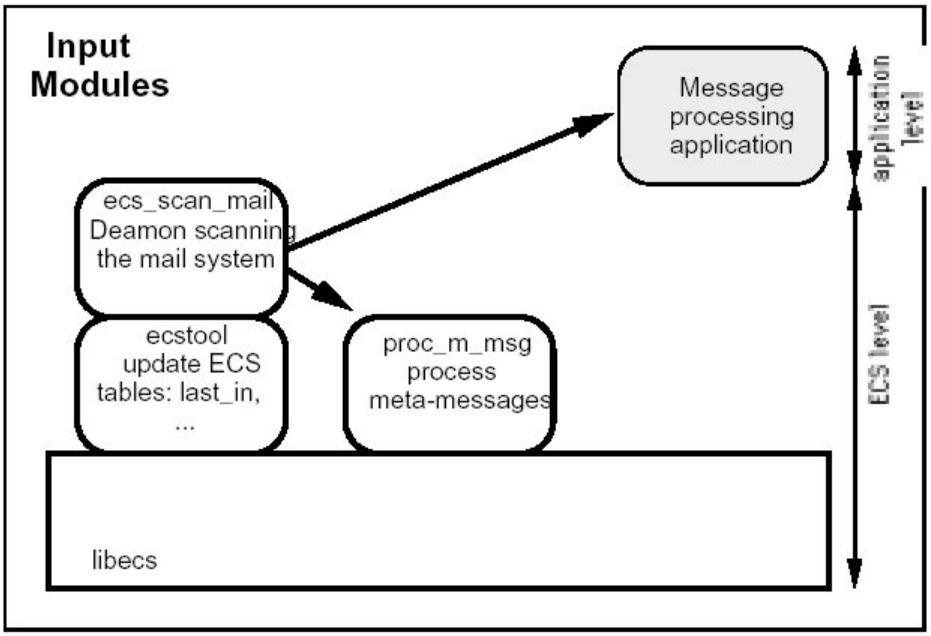



Figure 4: Input Modules

### 5.6.2. Scripts and Modules for Output

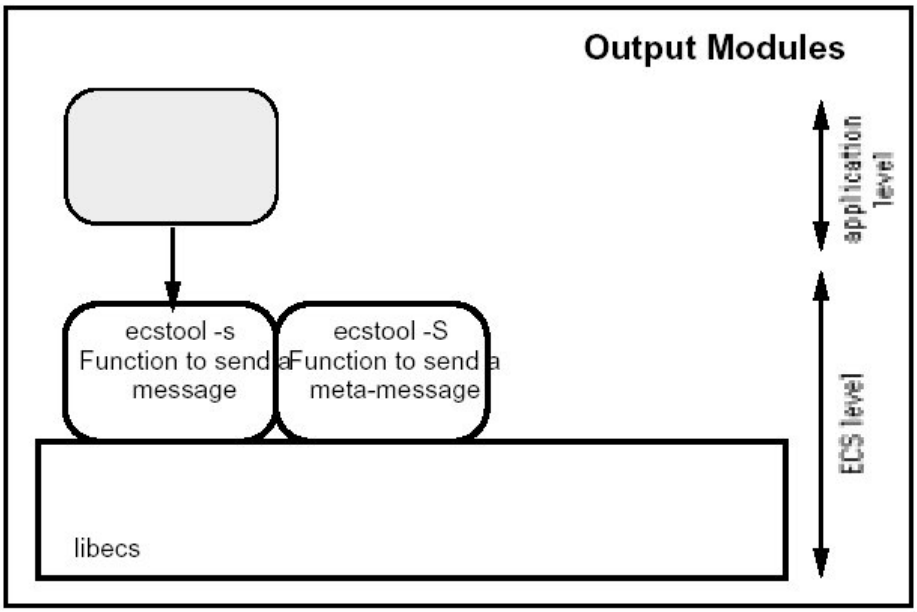



Figure 5: Output Modules

### 5.6.3. Other Scripts

There are a lot of other scripts which might be useful to the ECS consumer, but they are not all documented here. However, the scripts are well documented themselves and therefore the reader is invited to take a look at all the things s/he finds in ECS_BIN_DIR.

### 5.6.4. ECS Daemons

ECS is based on two daemons, i.e. processes which are continuously running in the background:

```
ecs_scan_mail
ecs_chk_com
```

The first daemon "ecs_scan_mail" scans the mail system for incoming ECS messages and calls the message processing script or program, when appropriate. If there is nothing to do it sleeps for T_SCN seconds as defined in the configuration file.

The second daemon "ecs_chk_com" performs the regular connection check. Between two checks it sleeps T_CHK seconds, as defined in the configuration file.

Starting and stopping of these daemons is done by the "ecs" command as shown in the chapter on maintenance.

Both daemons use semaphores (lock files in ECS_DAT_DIR) to prevent several daemons from running on the same database.

## 5.7. ECS Database Files

The ECS system stores all internal information in two files in the ECS_DAT_DIR directory. The file "ecs_node_tbl.dat" contains one line or record for each remote node in the ECS network and the associated data and the file "ecs_msg_tbl.dat" contains one line or record for each "strange" message, i.e. fast or slow messages.

Both tables can be exported and imported (i.e. converted from and to text format) by the -E and -I option of "ecstool". The corresponding text files are "ecs_node_tbl.asc" and "ecs_msg_tbl.asc".

### 5.7.1. File “node_tbl.dat”

Record elements:

node: The ECS node name. (Primary key)

addr: The e-mail address of this ECS node.

addr_r: The secondary e-mail address of this ECS node[5].

in_seq: The sequence number of the last message accepted from this node.

last_in: Date and time when the last message from this node was received.

out_seq: The sequence number of the last message sent to this node.

last_out: Date and time when the last message was sent to this node.

ack_seq: The highest sequence number acknowledged by this node.

Notes:

(1) Each remote node in an ECS network is represented by a record in this file.

(2) Date and time is represented by a long INTEGER containing the number of seconds since 1.1.1970, i.e. it is equivalent to the UNIX internal time format. This data type for date and time was chosen to allow for simple calculation of time differences in a broad range from days down to seconds. (The absolute time is not interesting.)

### 5.7.2. File "msg_tbl.dat"

Record elements:

node: ECS node name representing the sender of a "strange" message.

[5] Not really used currently. Reserved for future use.

seq_num: Sequence number of a missing message - represented by a negative number - or sequence number of a message which arrived too early - represented by a positive number.

d_detected: Date and time when it was detected that this message is missing.

re-send: Number of re-send meta-messages sent so far.

cycle_number: Number of check-cycles done since the first RE_SEND message was issued.

Notes:

(1) Each record in this file represents either a missing message or a message which arrived too early. The pair of node and sequence number is unique within this file.

## 5.8. ECS Mailboxes

Under the sub-directory "mboxes" under ECS_DAT_DIR the following mailboxes are maintained which can be accessed via the mail(C) command line option "-f".

active A mailbox which contains all mail (not only ECS mail) found in the user's mailbox defined by the environment variable $MAIL. The ECS daemon re-fills this mailbox from the user mailbox whenever it is empty. Non ECS messages are immediately put back to the user's mailbox. A separate mailbox is used to make sure that the contents of the mailbox is not changed while the ECS daemon looks for ECS messages, which cannot be guaranteed for the normal user mailbox.

in All messages and meta-messages which were recognised as ECS messages (due to the fact that their subject line started with the defined ECS mark) and that have been unloaded from the mail system for processing.

out_<node> All messages, but not meta-messages, sent via the ECS tools to the specified node.

out Messages sent to the local administrator. (non-ECS messages).

trash All ECS messages that arrived more than once and which were therefore discarded and never processed. Meta-messages from an unknown node are also stored here.

store All ECS messages that arrived ahead of time and which were not processed yet. 'Real' messages from an unknown node are also stored here. However, these messages do not show up in the msg table, but they will be processed as soon as the node is registered.

If configured, another directory on another disk can contain this mailbox:

in_bck A 1:1 copy of the mailbox "in" mentioned above.

## 5.9. ECS Semaphores

A semaphore technique is used to make sure that there are not multiple daemons running using the same database. The semaphores are implemented by lock files[6] which are created in the ECS_DAT_DIR for each of the two processes. The lock files have the names

```
ecs_scan_mail_<pid>.lck
ecs_chk_com_<pid>.lck
```

where <pid> is the process id of the daemon which created the lock. When a daemon script is started, it will check for the existence of another lock - before entering the "forever" loop - and terminate, if the lock exists. Inside the forever loop the script will always check the existence of its own lockfile, and terminate if this does not exist anymore.

While the two lock files mentioned above are used to prevent running of several daemons on the same database, a third lock file called

```
ecs_off.lck
```

is used to define whether ECS is switched on or off. If this lock file exists, ECS daemons will not start or will terminate as soon as they notice it.

## 5.10. ECS Error And Log File Format

ECS writes continuously log output and error output to two files which are called by default "ecs.log" and "ecs.err". By default these two files are located in the ECS_DIR directory, but the configuration file allows both filename and location to be changed.

The format of both files is identical. Each entry consists of a single line with at least three fields separated by the vertical bar "|":

```
timestamp | origin | code { | text }
```

| | |
|---|---|
| timestamp: | Date and time in the form "YYYY/DD/MM,hh:mm:ss" |
| origin: | The module, function or script writing this entry |
| code: | A short mnemonic string indicating the type of message or error |
| text: | Some additional information dependent on the message. This is optional. |

# 6. Problems and Plans

## 6.1. Known Problems

The current format of the RE_SEND meta-message allows only to specify a single message number to be re-sent. This can result in a burst of meta-messages, if a very fast message occurs (e.g. a message with a sequence number that is 20 higher than the expected one will cause 20 meta-messages. In a test setup with various nodes running on the same machine with

[6] This may not look too reliable to the expert, but it seems that it does its job, although definitely not in the most efficient manner. Future versions may use other techniques.

very short scan and check intervals this can cause deadlock. Real life applications must show, whether a single RE_SEND with multiple numbers is necessary.

A missing message, which arrives finally on a node which requested the RE_SEND, is treated like any other incoming message, i.e. it goes to the end of the input queue. If in this queue many messages are already waiting (e.g. because ECS was stopped), the processing of message of this particular node is blocked, until the formerly missing message becomes the first in the queue. Maybe the scan daemon should be changed such that after normal processing it scans the incoming queue quickly for the missing message with the lowest sequence number from each node.

In case a node loses (among other things) its "store" mailbox and needs to recover by issuing RE_SEND request, these re-sent messages may be trashed on arrival if they do not have exactly the sequence number of the next message to recover. This happens because the database is used to decide whether the message is stored or not, but not the mailbox. Therefore the recovery is very slow in this case (but it works). But since this case is so weird, the optimization of the recovery is postponed to later versions.

## 6.2. Not Yet Implemented Items

Implementation of the handling of the "flags" like reverse charging.

## 6.3. Other Plans and Considerations

- Sequence number overflow / re-sync or re-init of the messages between two nodes.
- Allow for re-sync: A node starts again at 1.

# 7. Short Reference Page

Start & Stop ECS daemons, report current state (use -s option when used from scripts), display monitor:

```
ecs < start | stop | state | mon >
ecs < b     | e    | s     | m   >
```

Send a message from a C application

```
int err_code;
err_code = ecs_put_msg
( char *file_name, char *node_name, char *flags )
```

Send a message from the Unix command line:

```
ecstool -s <file_name> <node_name> {flags}
ecs_send_msg <file_name> <node_name> {flags}
```

Request RE_SEND of a message

```
ecs_resend_req <node_name> <seq_number>
```

Summary of the most important "ecstool" functions:

```
ecstool –v  List tables (default)
ecstool -a <node> <addr> <addr_r> Add ECS node
ecstool -m <node> <addr> <addr_r> Modify existing ECS
node
ecstool -d <node> Delete ECS node
ecstool -r <node> {seq}Register incoming message
ecstool -c  Perform communication check
```

For a complete list of ecstool functions use the help function "ecstool -h". All of these functions are also available as C function calls.