EPICS
Channel Access Security
Design

Marty Kraimer and Jeff Hill

May1994

This document presents the design for implementing the requirements specified in:

EPICS: Channel Access Security - Functional Requirements, Ned D. Arnold, 03/09/92

Chapter 1: User's Guide

This chapter describes how to use the access security system.

1.1 Features

EPICS security protects IOC databases from unauthorized Channel Access Clients. Channel Access security is based on the following:

Who Userid of the channel access client.

Where Hostid where the user is located. It should be noted that if the user is logged on via a telnet or rlogin session his host is considered "REMOTE".

What Individual fields of records are protected. Each record has a field containing the Access Security Group (ASG) to which the record belongs. Each field has an access security level, which must be 0 or 1. Note that the security level is defined in the ascii record definition file. Thus the access security level for a field is the same for all instances of records of a given type.

When Access rules can contain input links and calculations similar to the calculation record, thereby including current values of process variables as part of access privilege.

1.2 Limitations

An IOC database can be accessed only via channel access or via the vxWorks shell. It is assumed that access to the local ioc console is protected via physical security and telnet/rlogin access protected via normal Unix and physical security.

No attempt has been made to protect against the sophisticated saboteur. Unix security must be used to limit access to the subnet on which the iocs reside.

1.3 Definitions

This document uses the following terms:

ASL	Access Security Level (Called access level in Req Doc)
ASG	Access Security Group (Called PV Group in Req Doc)
UAG	User Access Group
HAG	Host Access Group

1.4 Access Security Configuration File.

This section describes the format of a file containing definitions of the user access groups, host access groups, and access security groups. An ioc creates an access configuration database by reading an access configuration file(the extension .acf is recommended). Lets first give a simple example and then a complete description of the syntax.

1.4.1 Simple Example

```
UAG(uag) {user1,user2}
HAG(hag) {host1,host2}
ASG(DEFAULT) {
   RULE(1,READ)
   RULE(1,WRITE) {
```

```
        UAG(uag)
        LAG(lag)
    }
}
```

These rules provide read access to anyone located anywhere and write access to user1 and user2 if they are located at host1 or host2.

1.4.2 Syntax Definition

In the following description:

[ ]	Lists optional elements
|	Separator for alternatives
...	Means that an arbitrary number of definitions may be given.
	Any line beginning with # is a comment

```
UAG(<name>) [{ <user> [, <user> ...] }]
...
HAG(<name>) [{ <host> [, <host> ...] }]
...
ASG(<name>) [{
    [INP<index>(<pvname>)
    ...]
    RULE(<level>,NONE | READ | WRITE) {
        [UAG(<name> [,<name> ...])]
        [HAG(<name> [,<name> ...])]
        CALC("<calculation>")
    }
    ...
}]
...
```

Rules:

UAG User Access Group. This is a list of userids. The list may be empty. The same userid can appear in multiple UAGs. For iocs the userid is taken from the user field of the boot parameters.

HAG Host Access Group. This is a list of host names. It may be empty. The same host name can appear in multiple HAGs. The name "REMOTE" is handled separately. If a user has performed a remote login to a host then the users host is"REMOTE" rather than the host itself.???What is exact definition of host??? For iocs the host name is taken from the target name of the boot parameters.

ASG An access security group. The group"DEFAULT" is a special case. If a member is null or does not have an ASG definition then the member is assigned to the group "DEFAULT".

INP<index> Index must have one of the values "A" to "L". These are just like the INP fields of a calculation record. It is necessary to define INP fields if a CALC field is defined in any RULE for the ASG.

RULE This defines access permissions. <level> must be 0 or 1. Permission for a level 1 field implies permission for level 0 fields. The permissions are NONE, READ, and WRITE. WRITE permission impels

READ permission. The standard EPICS record types have all fields set to level 1 except for VAL, CMD (command), and RES (reset).

UAG specifies a list of user access groups that can have the access privilege. If UAG is not defined then all users are allowed.

HAG specifies a list of host access groups that have the access privilege. If HAG is not defined then all hosts are allowed.

CALC is just like the CALC field of a calculation record except that the result must evaluate to true or false. If the calculation results in (0,1) meaning (FALSE,TRUE) then the rule (doesn't apply, does apply) . The actual test is .99 < result < 1.01.

Each IOC record contains a field ASG, which specifies the name of the ASG to which the record belongs. For each channel access client attaching to a record the access privilege is determined as follows:

1 The ASG associated with the record is searched.
2 Each RULE is checked for the following:
    a Check that this field's level is <= the level for this RULE.
    b If UAG is defined, is this user a member of one of the specified UAGs. If UAG is not defined all users are accepted.
    c If HAG is defined, is the user located at one of the HAGs. If HAG is not defined all hosts are accepted.
    d If CALC is specified, does the calculation yield the value 1, i.e. TRUE. If any of the INP fields associated with this calculation are in INVALID alarm severity the calculation is considered false. The actual test for TRUE is .99 < result < 1.01.
3 The maximum access allowed by step 2 is the access chosen.

Note that these rules allow multiple RULEs to be defined for a given ASG, even RULEs with identical levels and access permission.

1.5 ascheck - Check Syntax of Access Configuration File

After creating or modifying an access configuration file it can be checked for syntax errors by issuing the command:

```
ascheck < "filename"
```

This is a unix command. It displays errors on stdout. If no errors are detected it prints nothing. Please note that this only detects syntax errors not logic errors. Thus it is still possible to get your self in trouble.

1.6 IOC Access Security Initialization

In order to have access security turned on during ioc initialization the following command must appear in the startup file before iocInit is called:

```
asSetFilename("<access security file>")
```

If this command does not appear then access security will not be started by iocInit.

After an ioc is initialized the access security database can be changed. The preferred way is via the subroutine record described in the next section. It can also be changed by issuing the following command to the vxWorks shell:

```
asInit
```

It is also possible to reissue asSetFilename before asInit.

1.7 Database Configuration

1.7.1 Access Security Group

Each database record has a field ASG which holds a character string. Any database configuration tool can be used to give a value to this field. If the ASG of a record is not defined or is not equal to any ASG in the configuration file then the record is placed in DEFAULT.

1.7.2 Subroutine Record Support

Two subroutines, which can be attached to a subroutine record, are available (provided with iocCore):

```
asSubInit
asSubProcess
```

If a record is created that attaches to these routines then that record can be used to force the ioc to load a new access configuration database. To change the access configuration:

1 Modify the file specified by the last call to asSetFilename so that it contains the new configuration desired.
2 Write a 1 to the subroutine record VAL field. Note that this can be done via channel access.

The following action is taken:

1 When the value is found to be 1, asInit is called and the value set back to 0.
2 The record is treated as an asynchronous record. Completion occurs when the new access configuration has been initialized or a time-out occurs. If initialization fails the record is placed into alarm with a severity determined by BRSV.

1.7.3 ascii Record Descriptions

Any record descriptions that were created before access security was added have to be modified. A new column specifying the access security level needs to be added. This column must have the value ASL0 or ASL1. If your ascii record definition files are named *Record.ascii the following script will help perform the conversion:

```
#! /bin/sh
for file in *Record.ascii
do
ex - $file <<\EOF
1,$s/DBF_/ASL1 DBF_/
wq
EOF
done
```

Lets assume that this file is named cvtascii (make sure it has execute permission). The following steps can be used to convert the ascii files:

1 execute the command
cvtascii
2 Edit each file and change ASL1 to ASL0 where necessary.

For the standard EPICS supplied records only the fields VAL, RES (reset), and CMD (command) are set to ASL0. You should think hard before you set any other fields to ASL0.

1.8 Example:

Lets design a set of rules for a Linac. Assume the following:

1 Anyone can have read access to all fields at anytime.
2 Linac engineers, located in the injection control or control room, can have write access to most level 0 fields only if the linac is not in operational mode.
3 Operators, located in the injection control or control room, can have write access to most level 0 fields anytime.
4 The operations supervisor, linac supervisor, and the application developers can have write access to all fields but must have some way of not changing something inadvertently.
5 Most records use the above rules but a few (high voltage power supplies, etc.) are placed under tighter control. These will follow rules 1 and 4 but not 2 or 3.
6 IOC channel access clients always have level 1 write privilege.

Most linac ioc records will not have the ASG field defined and will thus be placed in ASG "DEFAULT". The following records will have an ASG defined:

1 `LI:OPSTATE` and any other records that need tighter control have ASG="critical". One such record could be a subroutine record used to cause a new access configuration file to be loaded. `LI_OPSTATE` has the value (0,1) if the linac is (not operational, operational).
2 `LI:lev1permit` has ASG="permit". In order for the opSup, linacSup, or an appDev to have write privilege to everything this record must be set to the value 1.

The following access configuration satisfies the above rules.

```
UAG(op) {op1,op2,superguy}
UAG(opSup) {superguy}
UAG(linac) {waw,nassiri,grelick,berg,fuja,gsm}
UAG(linacSup) {gsm}
UAG(appDev) {nda,kko}
HAG(icr) {silver,phebos,gaea}
HAG(cr) {mars,hera,gold}
HAG(ioc) {ioclic1,ioclic2,ioclid1,ioclid2,ioclid3,ioclid4,ioclid5}
ASG(DEFAULT) {
   INPA(LI:OPSTATE)
   INPB(LI:lev1permit)
   RULE(0,WRITE) {
      UAG(op)
      HAG(icr,cr)
      CALC(”A=1”)
   }
   RULE(0,WRITE) {
      UAG(op,linac,appdev)
      HAG(icr,cr)
      CALC(”A=0”)
   }
   RULE(1,WRITE) {
      UAG(opSup,linacSup,appdev)
      CALC("B=1")
   }
   RULE(1,READ)
   RULE(1,WRITE) {
      HAG(ioc)
   }
}
ASG(permit) {
   RULE(0,WRITE) {
       UAG(opSup,linacSup,appDev)
   }
   RULE(1,READ)
   RULE(1,WRITE) {
      HAG(ioc)
   }
}
ASG(critical) {
   INPB(LI:lev1permit)
   RULE(1,WRITE) {
      UAG(opSup,linacSup,appdev)
      CALC("B=1")
   }
   RULE(1,READ)
   RULE(1,WRITE) {
      HAG(ioc)
   }
}
```

1.9 Performance and Memory Requirements

Performance has not yet been measured but during the tests to measure memory usage no noticeable change in performance during ioc initialization or during Channel Access clients connection was noticed. Unless access privilege is violated the overhead during channel access gets and puts is only an extra comparison.

In order to measure memory usage, the following test was performed:

1 A database consisting of 5000 soft analog records was created.

2 A channel access client (caput) was created that performs ca_puts on each of the 5000 channels. Each time it begins a new set of puts the value increments by 1.

3 A channel access client (caget) was created that has monitors on each of the 5000 channels.

The memory consumption was measured before iocInit, after iocInit, after caput connected to all channels, and after caget connected to all 5000 channels. This was done for APS release 3.11.5 (before access security) and the latest version which include access security. The results are:

	R3.11.5	Latest
Before iocInit	4,244,520	4,860,840
After iocInit	4,995,416	5,964,904
After caput	5,449,780	6,658,868
After caget	8,372,444	9,751,796

Before the database was loaded the memory used was 1,249,692 bytes. Thus most of the memory usage before iocInit resulted from storage for records. The increase since R3.11.5 results from added fields to dbCommon. Fields were added for access security, synchronous time support and for the new caching put support. The other increases in memory usage result from the control blocks needed to support access control. The entire design was based on maximum performance. This resulted in increased memory usage.

Chapter 2: Design Overview

2.1 Summary of Functional Requirements

A brief summary of the Functional Requirements is:

1 Each field of each record type is assigned an access security level.
2 Each record instance is assigned to a unique access security group.
3 Each user is assigned to one or more user access groups.
4 Each node is assigned to a host access group.
5 For each access security group a set of access rules can be defined. Each rule specifies:
   a Access security level
   b READ or READ/WRITE access.
   c An optional list of User Access Groups or * meaning anyone.
   d An optional list of Host Access Groups or * meaning anywhere.
   e Conditions based on values of process variables

An additional requirement is stated in section 4.5 of the requirements manual:

”For interactive channel access clients, provisions must be made to alter the current user without requiring the client program to restart”

2.2 Additional Requirements

Performance

Although the functional requirements doesn’t mention it, a fundamental goal is performance. The design provides almost no overhead during normal database access and moderate overhead for the following: channel access client/server connection, ioc initialization, a change in value of a process variable referenced by an access calculation, and dynamically changing a records access control group. Dynamically changing the user access groups, host access groups, or the rules, however, can be a time consuming operation. This is done, however, by a low priority IOC task and thus does not impact normal ioc operation.

Generic Implementation

Access security should be implemented as a stand alone system, i.e. it should not be imbedded tightly in database or channel access.

No Access Security within an IOC

Within an IOC no access security is invoked. This means that database links and local channel access clients calls are not subject to access control. Also test routines such as dbgf should not be subject to access control.

Defaults

It must be possible to easily define default access rules.

Remote Users

Users who rlogin to a host must be treated different than local host users.

Access Security is Optional

When an IOC is initialized, access security can be turned off.

2.3 Implementation Overview

The implementation provides a library of routines for accessing the security system. This library has no knowledge of channel access or IOC databases, i.e. it is generic. Database access, which is responsible for protecting an IOC database, calls library routines to add each IOC record to one of the access control groups.

Lets briefly discuss the access security system and how database access and channel access interact with it.

Configuration File

User access groups, host access groups, and access security groups are configured via an ascii file.

Access Security Library

The access security library, which is a new EPICS component, consists of the following groups of routines: initialization, group manipulation, client manipulation, access computation, and diagnostic. The initialization routine reads a configuration file and creates a memory resident access control database. The group manipulation routines allow members to be added and removed from access groups. The client routines provide services for clients attached to members.

IOC Database Access Security

This provides an interface between an ioc database and the access security system.

Channel Access Security

Whenever the channel access broadcast server receives a ca_search request and finds the process variable, it calls asAddClient.Whenever it disconnects it calls asRemoveClient. Whenever it issues a get or put to the database it must call asCheckGet or asCheckPut.

Channel access is responsible for implementing the requirement of allowing the user to be changed dynamically.

2.4 Comments

It is likely that the access rules will be defined such that many IOCs will attach to a common process variable. As a result the IOC containing the pv will have many CA clients.

What about password protection and encryption? I maintain that this is a problem to be solved in a level above the access security described in this document. This is the issue of protecting against the sophisticated saboteur.

Chapter 3: Access Security Application Programmer's Interface

3.1 Definitions

```
typedef struct asgMember *ASMEMBERPVT;
typedef struct asgClient *ASCLIENTPVT;
typedef int (*ASINPUTFUNCPTR)(char *buf,int max_size);
typedef enum{
    asClientCOAR/*Change of access rights*/
    /*For now this is all*/
} asClientStatus;
typedef void (*ASCLIENTCALLBACK)(ASCLIENTPVT,asClientSta-
tus);
```

3.2 Initialization

long asInitialize(ASINPUTFUNPTR inputFunction)

This routine reads an access definition file and performs all initialization necessary.

asInitialize can be called multiple times. If an access system already exists the old definitions are removed and the new one initialized. Existing members are placed in the new ASGs.

3.3 Group manipulation

long asAddMember(ASMEMBERPVT *ppvt, char *asgName);

This routine adds a new member to ASG asgName. The calling routine must provide storage for ASMEMBERPVT. Upon successful return *ppvt will be equal to the address of storage used by the access control system. The access system keeps an orphan list for all asgNames not defined in the access configuration.

The caller must provide permanent storage for asgName.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

long asRemoveMember(ASMEMBERPVT *ppvt);

This routine removes a member from an access control group. If any clients are still present it returns an error status of S_asLib_clientExists without removing the member.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

void *asGetMemberPvt(ASMEMBERPVT pvt);

For each member, the access system keeps a pointer that can be used by the caller. This routine returns the value of the pointer.

This routine returns NULL if access security is not active

long asPutMemberPvt(ASMEMBERPVT pvt,void *userPvt);

This routine is used to set the pointer returned by asGetMemberPvt.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

long asChangeGroup(ASMEMBERPVT *ppvt, char *newAsgName);

This routine changes the group for an existing member. The access rights of all clients of the member are recomputed.

The caller must provide permanent storage for newAsgName.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

3.4 Client Manipulation

long asAddClient(ASCLIENTPVT *ppvt,ASMEMBERPVT pvt,int asl,
char *user,char*host);

This routine adds a client to an ASG member. The calling routine must provide storage for ASCLIENTPVT**.** ASMEMBERPVT is the value that was set by calling asAddMember. asl is the access security level.

The caller must provide permanent storage for user and host.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

long asChangeClient(ASCLIENTPVT ppvt,int asl,char *user,char*host);

This routine changes one or more of the values asl, user, and host for an existing client. Again the caller must provide permanent storage for user and host. It is permissible to use the same user and host used in the call to asAddClient with different values.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

long asRemoveClient(ASCLIENTPVT *pvt);

This call removes a client.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

void *asGetClientPvt(ASCLIENTPVT pvt);

For each client, the access system keeps a pointer that can be used by the caller. This routine returns the value of the pointer.

This routine returns NULL if access security is not active.

void asPutClientPvt(ASCLIENTPVT pvt, void *userPvt);

This routine is used to set the pointer returned by asGetClientPvt.

long asRegisterClientCallback(ASCLIENTPVT pvt, ASCLIENTCALLBACK pcallback);

This routine registers a callback that will be called whenever the access privilege of the client changes.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

long asCheckGet(ASCLIENTPVT pvt);

This routine, actually a macro, returns (TRUE,FALSE) if the client (has, doesn't have) get access rights.

long asCheckPut(ASCLIENTPVT pvt);

This routine, actually a macro, returns (TRUE,FALSE) if the client (has, doesn't have) put access rights

3.5 Access Computation

long asComputeAllAsg(void);

This routine calls asComputeAsg for each access security group.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

long asComputeAsg(ASG *pasg);

This routine calculates all CALC entries for the ASG and calls asCompute for each client of each member of the specified access security group.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

long asCompute(ASCLIENTPVT pvt);

This routine computes the access rights of a client. This routine is normally called by the access library itself rather than use code.

This routine returns S_asLib_asNotActive without doing anything if access control is not active.

3.6 Diagnostic

int asDump(void (*member)(ASMEMBERPVT),
void (*client)(ASCLIENTPVT),int verbose);

This routine prints the current access security database. If verbose is 0 (FALSE), then only the information obtained from the access security file is printed.

If verbose is TRUE then additional information is printed. The value of each INP is displayed. The list of members belonging to each ASG and the clients belonging to each member are displayed. If member callback is specified as an argument, then it is called for each member. If client callback is specified, it is called for each access security client.

int asDumpUag(char *uagname)

This routine displays the specified UAG or if uagname is NULL each UAG defined in the access security database.

int asDumpHag(char *hagname)

This routine displays the specified UAG or if uagname is NULL each UAG defined in the access security database.

int asDumpRules(char *asgname)

This routine displays the rules for the specified ASG or if asgname is NULL the rules for each ASG defined in the access security database.

int asDumpMem(char *asgname,void (*memcallback)(ASMEMBERPVT),int clients)

This routine displays the member and, if clients is TRUE, client information for the specified ASG or if asgname is NULL the member and client information for each ASG defined in the access security database. It also calls memcallback for each member if this argument is not NULL.

int asDumpHash(void)

This shows the contents of the hash table used to locate UAGs and HAGs,

Chapter 4: Database Access Security

4.1 Access Level definition

The definition of access level means that a level is defined for each field of each record type.

1 Structure fldDes (dbRecDes.h), which describes the attributes of each field, contains a field access_security _level. In addition definitions exist for the symbols: ASL0 and ASL1.

2 Each field description in an ascii record description contains a field with the value ASLx.

The meanings of the Access Security Level definitions are as follows:

ASL0 .Assigned to fields used during normal operation

ASL1 Assigned to fields that may be sensitive to change. Permission to access this level implies permission for ASL0.

Most record types assign ASL as follows: The fields VAL, RES (Reset), and CMD use the value ASL0. All other fields use ASL1.

4.2 Access Security Group definition

dbCommon contains the fields ASG and ASP. ASG (Access Security Group) is a character string. The value can be assigned via a database configuration tool or else a utility could be provided to assign values during ioc initialization. ASP is an access security private field. It contains the address of an ASGMEMBER.

4.3 Access Client Definition

Struct dbAddr contains a field asPvt, which contains the address of an ASGCLIENT. This definition is also added to struct db_addr so that old database access also supports access security.

4.4 Database Access Library

Two files dbAsLib.c and asCa.c implement the interface between ioc databases and access control. It contains the following routines:

4.4.1 Initialization Routines

int asSetFilename(char *acf)

Calling this routine sets the filename of an access configuration file. The next call to asInit uses this file. If asSetFilename is never called or is passed a NULL filename then the next call to asInit results in access security being disabled. This routine must be called before iocInit if you want access security to be turned on during ioc initialization.

int asInit(ASDBCALLBACK *pcallback)

This routine calls asInitialize. If the current access configuration file, as specified by asSetFilename, is NULL then the routine just returns, otherwise the configuration file is used to create the access configuration database.

This routine is called by iocInit. asInit can also be called at any time to change the access configuration information.

An important part of initialization is starting a channel access client which attaches monitors to all INPx fields. This task takes care of recomputing access permissions whenever a monitor is triggered.

asInit actually spawns a task asInitTask to perform the initialization. This allows asInit to be called from a subroutine called by the process entry of a subroutine record.asInitTask calls taskwdInsert so that if it suspends for some reason taskwd can detect the failure. After initialization all records in the database are made members of the appropriate access control group.

If the caller provides an **ASDBCALLBACK** then when either initialization completes or taskwd detects a failure the users callback routine is called via one of the standard callback tasks.

asInit will return a value of -1 if access initialization is already active. It returns 0 if asInitTask is successfully spawned.

4.4.2 Routines used by Channel Access Server

int asDbGetAsl(void *paddr)

Get Access Security level for the field referenced by a database access structure. The argument is defined as a void * so that both old and new database access can be used.

ASMEMBERPVT asDbGetMemberPvt(void *paddr)

Get ASMEMBERPVT for the field referenced by a database access structure. The argument is defined as a void * so that both old and new database access can be used.

4.4.3 Routine to test asAddClient

int astac(char *pname,char *user,char *host)

This is a routine to test asAddClient. It simulates the calls that are made by Channel Access.

4.4.4 Subroutines attached to a subroutine record

These routines are provided so that a channel access client can force an ioc to load a new access configuration database.

long asSubInit(struct subRecord *prec,int pass)

long asSubProcess(struct subRecord *prec)

These are routines that can be attached to a subroutine record. Whenever a 1 is written to the record, asSubProcess calls asInit. If asInit returns success, it returns with asynchronously. When asInitTask calls the completion routine supplied by asSubProcess, the return status is used to place the record in alarm.

4.4.5 Diagnostic Routines

These routines provide interfaces to the asDump routines described in the previous chapter. They also lock before calling the associated routine and unlock afterwards. Thus they will work correctly even if the access configuration is being changed.

asdbdump(void)

This routine calls asDump with a member callback and with verbose TRUE.

aspuag(char *uagname)

This routine calls asDumpUag.

asphag(char *hagname)

This routine calls asDumpHag.

asprules(char *asgname)

This routine calls asDumpRules.

aspmem(char *asgname,int clients)

This routine calls asDumpMem.

Chapter 5: Channel Access Security

EPICS Access Security is designed to protect Input Output Controllers (IOCs) from unauthorized access via the Channel Access (CA) network transparent communication software system. This chapter describes the interaction between the CA server and the Access Security system. It also briefly describes how the current access rights state is communicated to clients of the EPICS control system via the CA communication system and the CA client interface.

5.1 CA Server Interfaces to the Access Security System

The CA server calls `asAddClient()` and `asRegisterClientCallback()` for each of the channels that a client connects to the server. The routine `asRemoveClient()` is called whenever the client clears (removes) a channel or when the client disconnects.

The server maintains storage for the clients host and user names. The initial value of these strings are supplied to the server when the client connects and can be updated at any time by the client. When these strings change then `asChangeClient()` is called for each of the channels maintained by the server for the client.

The server checks for read access when processing get request and for write access when processing put requests. If access is denied then an exception message is sent to the client.

The server checks for read access when processing requests to register an event callback (monitor) for the client. If there is read access the server always sends an initial update indicating the current value. If there isn't read access the server sends one update indicating no read access and disables subsequent updates.

The server receives asynchronous notification of access rights change via its call back registered with `asRegisterClientCallback()`. When a channel's access rights change the server communicates the current state to the client library. If read access to a channel is lost and there are events (monitors) registered on the channel then the server sends an update to the client for each of them indicating no access and disables future updates for each event. If read access is reestablished to a channel and there are events (monitors) registered on the channel then the server re-enables updates and sends an initial update message to the client for each of them.

5.2 Client Interfaces

Additional details on the channel access client side callable interfaces to access security can be obtained from the "Channel Access Reference Manual".

The client library stores and maintains the current state of the access rights for each channel that it has established. The client library receives asynchronous updates of the current access rights state from the server. It uses this state to check for read access when processing get request and for write access when processing put requests. If a program issues a channel access request that is inconsistent with the client library's current knowledge of the access rights state then access is denied and an error code is returned to the application. The current access rights state as known by the client library can be tested by an applications program with the C macros `ca_read_access()` and `ca_write_access()`.

An application program can also receive asynchronous notification of changes to the access rights state by registering a function to be called back when the client library updates its storage of the access rights state. The application's call back function is installed for this purpose by calling `ca_replace_access_rights_event()`.

If the access rights state changes in the server after a request is queued in the client library but before the request is processed by the server then it is possible that the request will fail in the server. Under these circumstances then an exception will be raised in the client.

The server always sends one update to the client when the event (monitor) is initially registered. If there isn't read access then the status in the arguments to the application program's event call back function indicates no read access and the value in the arguments to the clients event call back is set to zero. If the read access right changes after the event is initially registered then another update is supplied to the application programs call back function.

Chapter 6: Access Control: Implementation Overview

This chapter provides a few aids for reading the access security code. Include file asLib.h describes the control blocks used by the access security library. The following is a picture of how the control blocks are related.

6.1 Basic Structures

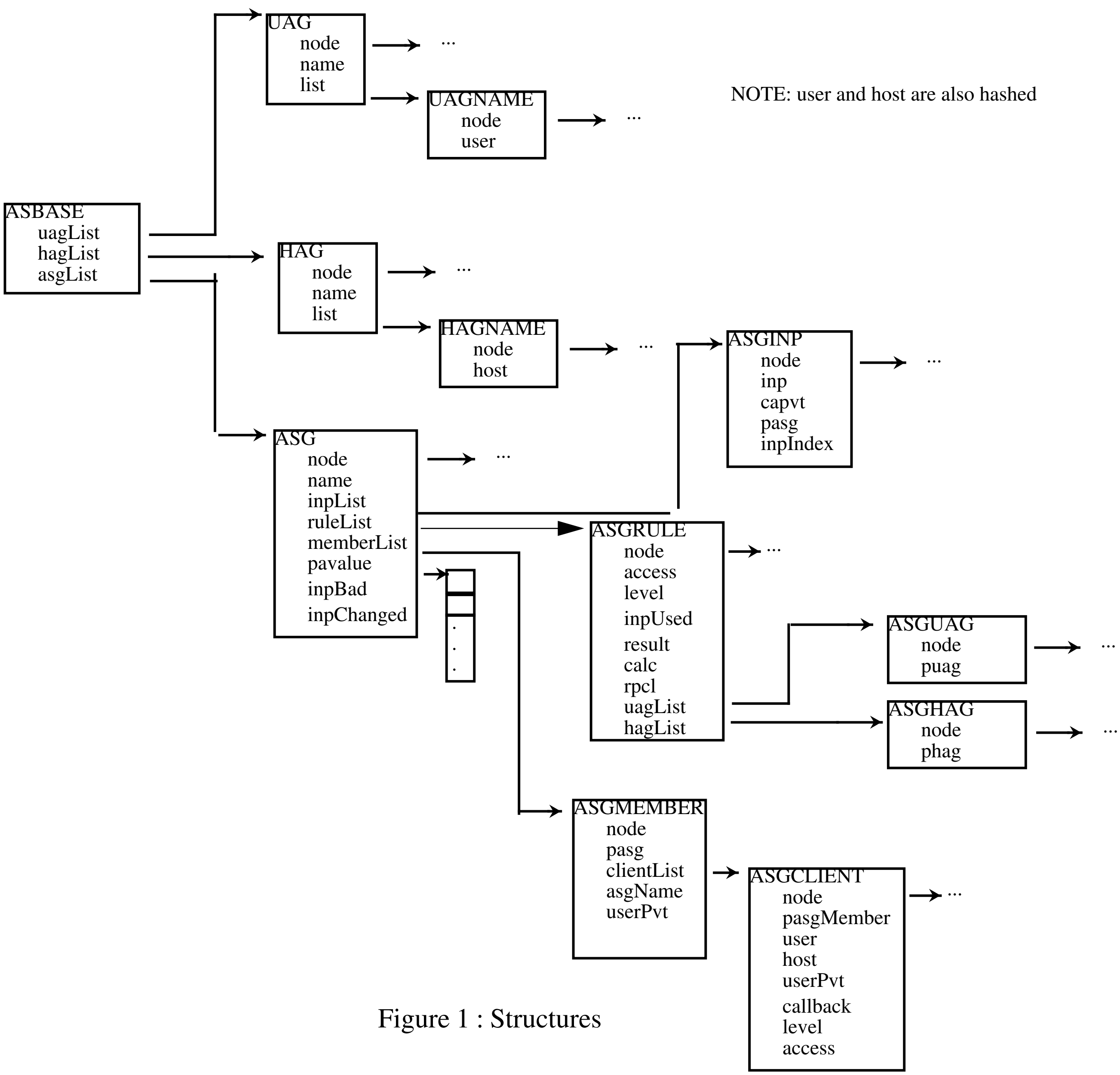

Figure 1 : Structures

6.2 Implementation Overview

The following files form the access security system:

asLib.h	Definitions for the portion of access security that is independent of ioc databases.
asDbLib.h	Definitions for access routines that interface to an ioc database.
asLib_lex.l	Lex and Yacc (actually Epics flex and antelope) are used to parse the access configuration file. This is the lex input file.
asLib.y	This is the yacc input file. Note that it includes asLibRoutines.c, which do most of the work.
asLibRoutines.c	These are the routines that implement access security. This code has no knowledge of the database or channel access. It is a general purpose access security implementation.
asDbLib.c	This contains the code for interfacing access security to the ioc database.
asCa.c	This code contains the channel access client code that implements the INP and CALC definitions in an access security database.
ascheck.c	The Unix program which perform a syntax check on a configuration file.

6.3 Locking

Because it is possible for multiple tasks to simultaneously modify the access security database it is necessary to provide locking. Rather than try to provide low level locking, the entire access security database is locked during critical operations. The only things this should hold up are access initialization, ca searches, ca clears, and diagnostic routines. It should NEVER cause record processing to wait. In addition ca gets and puts should never be delayed. One exception exists. If the ASG field of a record is changed then asChangeGroup is called which locks.

The access database is locked for the following operations:

1 While initializing. This is done by asInitTask.

2 asCaTask locks while calling asComputeAllAsgs

3 asCa callback routines lock except while initializing.

4 asSetFilename

5 asChangeGroup
asAddClient
asChangeClient
asRemoveClient

6 asdbdump
aspuag
asphag
asprules
aspmem

Note that the asDump* routines do not lock. This is done for diagnostic purposes. If for some reason an access database is permanently locked, it is still possible to invoke these diagnostic routines.