Hesiod Name Service
Application Programmer's Guide

Stephen P. Dyer
MIT Project Athena

Version A
8 July 1987

This is the first version, please direct comments and suggestions to Steve Dyer (username dyer) at (25)3-0127.

Table of Contents

1 Introduction

This is the application developers' guide to the use of the *Hesiod* name service. It is designed to complement the manual page, hesiod.3, and the section of the Athena Technical Plan which describes the Hesiod name service.

2 What is Hesiod

The *Hesiod* name service allows an application to retrieve associations between a name, a particular type of service, and information about that named service. Some examples of this are course locker names and system libraries, other RVD or NFS lockers, printer information, such as might be found in /etc/printcap, service and mailbox locations, and service-to-port mappings, such as are found in /etc/services. Much of what *Hesiod* now makes available had heretofore been kept on each workstation and timesharing machine as specialized databases, making the timely and efficient update of these databases very difficult.

Within Project Athena, the data which Hesiod makes available are maintained by the operations staff through the use of the Service Management System (SMS), and distributed several times a day to the authoritative Hesiod name servers which run, at present, on several machines within the E40 cluster and in the student areas.

Hesiod is designed to be used in situations where a small amount of data which changes infrequently needs to be accessed quickly, with little overhead. It is not intended to serve as a general-purpose database system supporting arbitrary queries, or as a repository for information that changes frequently. *Hesiod* provides no facility for an application to update its database, which is refreshed several times a day by the Athena SMS. Because of the limitation imposed by the underlying implementation of Hesiod, based as it is on the Internet domain naming scheme, there is a maximum length of 1024 bytes of data which can be exchanged between the name servers. This imposes limits on both the maximum size of an individual data record, as well as the number of records which can be returned in a single packet in the case of multiple matches. Hesiod was designed to provide applications with a rapid, low-overhead naming service in which a query would return no more than a few matches of limited size. Applications which require more complicated queries or ones which return voluminous data should consider interfacing to the Athena SMS.

3 Use of Hesiod

There are only two subroutines which are usually invoked by the applications programmer when using Hesiod. The subroutine `hes_resolve` is the primary interface into the Hesiod name server. It takes two string arguments, the name to be resolved, known as the *HesiodName*, and a type indicating the type of service associated with this name, known as the *HesiodNameType*. `hes_resolve` returns a pointer to a static array of strings, a la `argv[]`, containing all the data which matched the query, one match per array slot. The array is NULL terminated. A second call to `hes_resolve` will overwrite any previously-returned data, so applications which require data to be maintained across multiple calls to hes_resolve should copy the returned values into data areas they maintain.

Note that a call to hes_resolve may return more than one match. The semantics of using or choosing between multiple matches is dependent on the particular application. In general, however, multiple matches are considered "equivalent", and any of them could be used equally well. This is exploited, for example, by the *attach* command which attaches a remote file system to the workstation. In the case of read-only system libraries, the attach command iterates through all matches, stopping after the first successful attach. Because *Hesiod* is based on the Internet Domain Naming scheme, no interpretation can or should be given to the order in which matches are returned.

If hes_resolve returns NULL, then no data could be found, either because the name server had no matching records, or because of an error. The function hes_error takes no arguments and returns a small integer indicating the type of error, if any, encountered in the last call to hes_resolve.

It is important to emphasize that *Hesiod* knows nothing about the data it stores; any meaning given to the HesiodName, the HesiodNameType and the data returned by Hesiod is imposed completely by the application. The format of the data stored by *Hesiod* is application-specific, and would be defined by the application programmer.

```
#include <hesiod.h>

char *HesiodName, *HesiodNameType;
char **hp;

hp = hes_resolve(HesiodName, HesiodNameType);
if (hp == NULL) {
        err = hes_error();
        switch(err) {
        .
        .
        .
        }
} else {
        /* do your thing with hp */
        while(*hp != NULL) process(*hp++);
}
```

Here is a list of some of the presently-defined HesiodNameTypes, the kind of information provided as a HesiodName, and the applications programs which use such queries.

HesiodName	HesiodNameType	Used By	Info Returned
workstation name	"cluster"	getcluster	cluster information
filesystem name	"filsys"	attach/detach	RVD and NFS info
username	"pobox"	inc/movemail	location and type of mailbox
username	"passwd"	toehold/login	Athena-wide /etc/passwd entry
service	"sloc"	olc/kerberos	Name of machine (location) providing this service

The *Hesiod* library is available on both the RT and VS through the *-lhesiod* flag to the C compiler or loader.

The error values returned by `hes_error` are one of the following:

```
#define HES_ER_UNINIT   -1      /* uninitialized */
#define HES_ER_OK        0      /* no error */
#define HES_ER_NOTFOUND  1      /* Hesiod name not found by server */
#define HES_ER_CONFIG    2      /* local problem (bad config file?) */
#define HES_ER_NET       3      /* network problem or timeout */
```

The most common values returned by `hes_error` are HES_ER_OK, meaning no error, and HES_ER_NOTFOUND, meaning that the desired name was not found in the Hesiod data base. HES_ER_CONFIG indicates a problem with the optional per-machine Hesiod configuration file, `/etc/hesiod.conf`. HES_ER_UNINIT will never be returned by `hes_error`, unless it is called before the first time `hes_resolve` is called. HES_ER_NET indicates that the request never received a response from the Hesiod name server. This can be due to a variety of network problems: for example, the host making the request might be disconnected from the network, an intervening gateway might be down, or simply that all Hesiod name servers failed to respond. No further information about the state of the network is available because the domain system on which Hesiod is based uses datagrams with retries as the communications interface.

HES_ER_NOTFOUND is a negative acknowledgement indicating that the desired name/*HesiodNameType* pair was not found in the Hesiod database. An application receiving this error message can consider this an authoritative response. Of course, this may be due to an omission on the part of operations, or simply reflect a delay between the time Hesiod data was asked to be placed into the database, and the actual Hesiod updates, which will occur several times each day, in the same way as the *userinfod* information is presently propagated.

In the case of a Hesiod error of HES_ER_NET, it may be prudent for an application to assume that this situation is temporary, and that a later call to hes_resolve will either return the desired data or a definitive reply of HES_ER_NOTFOUND. HES_ER_CONFIG indicates a problem with the Hesiod configuration file, a situation which requires intervention by operations and which will not resolve itself spontaneously. No query to the *Hesiod* name server is actually made, so no conclusion can be drawn about the validity of the name to be resolved. The standard Athena distribution of the *Hesiod* library does not require a configuration file; its built-in defaults suffice, so this situation should not be encountered frequently.

A general design strategy for applications using Hesiod is to have a contingency plan in place in case *Hesiod* does not respond, is configured incorrectly or does not know the name. This may be built-in to the application, such as new versions of *lpr* which revert to using the old clustertab and printcap libraries if Hesiod printer information is not available. Another popular scheme, exploited by the MH application *inc* and the EMACS tool, *movemail*, is to allow the value of an environment variable, in this case, `MAILHOST`, to override the call to Hesiod to retrieve a person's mailhost, using his username as the key. Thus, a user can temporarily "hard-wire-in" appropriate values to allow applications to proceed. Not every application can be programmed in such a fashion, but it is prudent to try to design applications with this in mind.

4 How Do I Use Hesiod for Project X?

The Hesiod database already contains data for several *HesiodNameTypes*; see the Athena Technical Plan for a description of the types and the data returned by Hesiod. Other applications might wish to make use of Hesiod using their own *HesiodNameTypes*. If they require information which is appropriate to be managed centrally by Athena operations, all that is required is a list of the Hesiod names, the unique Hesiod name type associated with them, and the data associated with each name. Right now, developers should contact Steve Dyer to incorporate the data into the Hesiod database. Later on, the data will be managed centrally by the Athena operations staff through the use of the Athena SMS.