NISTIR 89-4042

Internal Structure of the Guide to Available Mathematical Software

Ronald F. Boisvert
Sally E. Howe
Jeanne L. Springmann

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
(Formerly National Bureau of Standards)
Center for Computing and Applied Mathematics
Gaithersburg, MD 20899

March 1989

NISTIR 89-4042

Internal Structure of the Guide to Available Mathematical Software

Ronald F. Boisvert
Sally E. Howe
Jeanne L. Springmann

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
(Formerly National Bureau of Standards)
Center for Computing and Applied Mathematics
Gaithersburg, MD 20899

March 1989

National Bureau of Standards became the National Institute of Standards and Technology on August 23, 1988, when the Omnibus Trade and Competitiveness Act was signed. NIST retains all NBS functions. Its new programs will encourage improved use of technology by U.S. industry.

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary

Ernest Ambler, Acting Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Raymond G. Kammer, Acting Director

Abstract

The purpose of the NIST Guide to Available Mathematical Software (GAMS) project is to provide convenient documentation tools for users and maintainers of scientific computer software. The main components of this effort are a detailed tree-structured, problem-oriented classification scheme for mathematical and statistical software, a printed catalog based upon this classification scheme which integrates information about all available software, an on-line interactive version of this catalog, and a relational database containing all information upon which the on-line and off-line catalogs rely, along with associated maintenance programs. This report presents a detailed specification of the internal structure of the GAMS database and the programs used to manipulate it. This information is useful to those who wish to implement the GAMS systems on their own computer systems.

Disclaimer

Certain commercial products are identified in this report in order to adequately document the existing GAMS system. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

Acknowledgements

The GAMS project team at NIST is currently comprised of Ron Boisvert, Elsie Clark, Sally Howe, David Kahaner, and Jeanne Springmann. We are indebted to may others who have provided advice and help with programming and data collection over the years.

Contents

List of Tables

Chapter 1

Introduction

The Guide to Available Mathematical Software (GAMS) project seeks to provide convenient documentation tools for users and maintainers of scientific software at NIST. The main components of this effort are a detailed tree-structured, problem-oriented classification scheme for mathematical and statistical software, a printed catalog based upon this classification scheme which integrates information about all available software [2], an on-line interactive version of this catalog, and a relational database containing all information upon which the on-line and off-line catalogs rely, along with associated maintenance programs. An overview of the project is given in [1]. The purpose of this report is to describe in some detail the implementation of the GAMS database and the computer programs associated with it.

In the GAMS project, software is organized into collections called *application packages*, which we will often simply refer to as *packages*. A package may be either a collection of subprograms, a collection of programs, or a collection of commands from an interactive system. About 30 packages are currently represented in our database. Some packages may be partitioned into smaller collections called *sublibraries*. Each sublibrary has the organizational structure of a package. The smallest user-callable unit within a package is called a *module*. Thus, a module may be either a subprogram, a program, or a command in an interactive system. Some of the software packages described in the current GAMS database are listed in Table 1.1. More than 4800 individual software modules are currently represented.

The organization of software into packages, sublibraries, and modules is primarily of interest to software maintainers. Software users need to know which modules can be used to solve specific problems. Thus, GAMS also provides an organization of software by functionality. The basis of this organization is a tree-structured, problem-oriented classification scheme for mathematical and statistical software [1].

To implement the database we use the RIM database management system [5,6]. RIM is a Fortran-based system which is available on a number of computing systems including Cyber NOS and Vax VMS. It includes an SQL-like interactive query system and a Fortran interface. The GAMS programs gain access to the database through the

Table 1.1: Packages Represented in the GAMS Database

BMDP
Collected Algorithms of the ACM
DATAPLOT
ELLPACK
IMSL Library
MAGEV Library
MATLAB
MINITAB
NAG Library
NIST Core Math Library (CMLIB)
PORT Library
ROSEPAK
Standards Time Series and Regression Package (STARPAC)

Fortran-callable subprograms of the RIM applications program interface.

In the first part of this report we present a detailed description of the format of the GAMS database. In the second part we describe various programs and procedures which have been built to manipulate this database. Finally, in the third part we describe the format in which the GAMS data is transported to non-RIM sites.

Chapter 2

The GAMS Database

The GAMS data are represented using the relational model [3]. In a relational model the data are represented in terms of a collection of tables called *relations*. The columns of the tables are called *attributes*. Each *row* represents a logical record of data.

There are ten relations in the GAMS database:

1. *AP*
   Contains *machine-independent* data describing application packages.

2. *APX*
   Contains data describing each (package,computer) pair.

3. *APHIST*
   Provides historical data on each (package,computer,version) triple.

4. *SUBLIB*
   Contains *machine-independent* data describing sublibraries.

5. *SUBLIBX*
   Contains data describing each (sublibrary,computer) pair.

6. *MODULE*
   Contains *machine-independent* data describing individual modules.

7. *MODULEX*
   Contains data describing each (module,computer) pair.

8. *COMPUTER*
   Contains data describing the computers on which the software is located.

9. *MOD\N*
   Contains the GAMS classification scheme.

10. *MOD\T*
    Contains pointers which determine the tree structure of the GAMS classification scheme, and pointers that specify the modules classified at each node.

To represent an application package in GAMS one adds one row to the AP relation to describe the package. In addition, one row is added to the MODULE relation for each module in the package. Finally, if the package is partitioned into sublibraries, one row is added to the SUBLIB relation for each of the package's sublibraries.

The data in the AP, MODULE, and SUBLIB relations are independent of any particular computer. However, we wish to also represent data describing particular *implementations* of each package on one or many computer systems. It is for this reason that the relations APX, MODULEX, and SUBLIBX are introduced. For each row in AP there may be many rows in the APX relation, each corresponding to the implementation of a version of the package on a particular computer system. For each row in APX there is a set of rows in MODULEX describing the implementation of modules in the package on the given computer. (Note that this implies that the MODULE relation contains the union of modules available in all implementations of a given package.) If the package is partitioned into sublibraries there is also a set of rows in the SUBLIBX relation. In this way the GAMS database can effectively represent implementations of application packages on many computer systems.

A precise description of the attributes within each relation can be found in the following sections. Some attributes are common to many relations. These include the names of packages (attribute AP), computers (attribute COMP), modules (attribute MODULE), and sublibraries (attribute SUBLIB). We do not assume that any of these names are unique. As a result, we have associated a unique identifier (an integer) with each of these items. These are the attributes named AP#, COMP#, MOD#, and SUBLIB#, respectively. Finally, for each relation we will indicate which attribute (or set of attributes) may be used as a key for the relation. A *key* has the property that it uniquely specifies a row of the relation (there are no duplicate keys).

When the string "-0-" appears as the first three characters of a text field in the database, then that field is considered empty.

2.1 The AP Relation

The AP relation contains machine-independent data which describes each package. Exactly one row appears in the AP relation for each package known to the database. The attribute AP# is a key for the AP relation.

The attributes of the AP relation are:

1. **AP#** (Integer; positive)
   The unique identifier for this package.

2. **AP** (Text, 12 characters; upper case)
   The name of this package.

3. **TYPE** (Integer; 1, 2, 3, 4, 5, or 6)
   Indicates how the package is organized: 1 ⇒ subprogram library (not divided into sublibraries), 2 ⇒ partitioned subprogram library (library divided into sublibraries), 3 ⇒ homogeneous collection of stand-alone programs (i.e., with a common input syntax), 4 ⇒ collection of commands in an interactive system, 5 ⇒ interactive program, 6 ⇒ heterogeneous collection of stand-alone programs. Examples of each type: (1) IMSL, (2) CMLIB, (3) BMDP, (4) Dataplot, (5) MATLAB, (6) Collected Algorithms of the ACM. The difference between 4 and 5 is that the individual commands within Dataplot have been classified using the GAMS Classification Scheme, while those within MATLAB have not.

4. **PORT** (Text, 1 character; E, H, M, or P)
   Indicates restrictions on library usage and ease of transporting the software to other machines: `E` ⇒ portable (in the public domain, written in a commonly available subset of the programming language, free of dependencies on the arithmetic of a specific machine and source code is available); `H` ⇒ portable some conversion required (while most of the code is portable, certain localized portions use proprietary software or machine-dependent constructs); `M` ⇒ machine-specific (uses special features of a particular machine); `P` ⇒ proprietary (use of this software is governed by a licensing agreement).

5. **DESC** (Text, variable length; free format)
   A brief description of this package.

6. **LANG** (Text, 12 characters; upper case)
   The computer language in which the package is coded.

7. **DEVELOP** (Text, variable length; free format)
   The name and the address of the organization where the library was developed and the name of a contact there. Example: `National Institute of Standards and Technology (NIST), Boulder, CO (J. Donaldson)`

8. **CITATION** (Text, variable length; free format)
   A reference to a monograph or technical article which describes the package. Example: `W. B. Gone, WISHPACK, Gnome Press, 1973`

9. **DISTRIB** (Text, variable length; free format)
   The name and address of the organization which is currently distributing the software.

2.2 The APX Relation

The APX relation contains information about the implementations of packages on each computer system where they are available. Each row corresponds to the implementation of a version of a package on a particular computer system. For example, if a given package is available on three computer systems, there will be three rows in the APX relation corresponding to this package; they will have the same AP#, but different COMP#. More precisely, the attribute pair (AP#,COMP#) is a key for the APX relation.

Only one version of a given package per computer is representable in the current database. The definition of "computer" is somewhat liberal, however, and some apparent violations of this principle are, in fact, allowed; see the comments in the description of the COMPUTER relation for details.

A name substitution feature is assumed active in several fields of the APX relation (ACCESS, SOURCE, TEST, SAMPLE, LIBDOC, and MODDOC). These fields give commands which can be used on the given computer system to retrieve items such as on-line documentation and source code for the given package. If such commands depend on the particular module in question, then the string `<MOD>` may be used to denote the module name in these fields. Programs processing these attributes will substitute the module name for this substring if information about a particular module is being processed. One can also use the strings `<AP>` and `<SBL>` to denote the current package and sublibrary, respectively. If more than one command is required, then they should be separated by the two characters \$; programs which print these fields will replace these characters by an end-of-line.

The attributes of the APX relation are:

1. **AP#** (Integer; positive)
   The unique identifier of this package. This determines the row of the AP relation that provides machine-independent information about this package.

2. **AP** (Text, 12 characters; upper case)
   The name of the package corresponding to AP#.

3. **COMP#** (Integer; positive)
   The unique identifier for the computer on which this package has been implemented.

4. **COMP** (Text, 6 characters; upper case)
   The name of the computer corresponding to COMP#.

5. **SUPP** (Text, 1 character; 1, 2, or 3)
   The level of support provided users of the library on this computer: 1 ⇒ full support; 2 ⇒ limited support; 3 ⇒ no formal support.

6. **ACCESS** (Text, variable length; free format)
   Command(s) which access this library on the given computer. Name substitution may be used. Example: `INVOKE,GETLIB,IMSL.`

7. **VER#** (Integer; positive)
   A unique identifier for this implementation of the package.

8. **VER** (Text, 6 characters; free format)
   The version name, number, or date, of this implementation. Examples: `Mark 9`

9. **LIBDOC** (Text, variable length; free format)
   Command(s) which retrieve detailed documentation for this implementation of the package on the named computer. Name substitution may be used. Example: `INVOKE,GETDOC,IMSL.`

10. **MODDOC** (Text, variable length; free format)
    Command(s) which retrieve detailed documentation for a module in this package on the named computer. Name substitution may be used. Example: `INVOKE,GETDOC,IMSL,<MOD>.`

11. **CITATION** (Text, variable length; free format)
    A reference to a monograph or technical article which describes this particular implementation of the package. Example: `W. B. Gone, Using WISHPACK on CSCS, User Services Document, 1975.`

12. **SAMPLE** (Text, variable length; free format)
    A system command which will retrieve a sample usage of modules in this package. Name substitution may be used.

13. **SOURCE** (Text, variable length; free format)
    A system command on the named computer which can be used to retrieve the source for modules in the package. Name substitution may be used. Examples: `COPY [DATAPAC.SOURCE]<MOD>.FOR *.*`

14. **TESTS** (Text, variable length; free format)
    A system command which will retrieve test programs for modules in this package. Name substitution may be used.

Three redundant data items are found in this relation: AP, COMP, and VER. These are symbolic identifiers from the AP, COMPUTER, and APHIST relations, respectively, associated with the unique identifiers for APs, computers, and package versions (AP#, COMP#, and VER#). These attributes are repeated in this relation in order to simplify retrieval.

2.3 The APHIST Relation

This relation is used to keep a record of which versions of each application package are or have been implemented on each computer. For each row in the APX relation there will be one or more rows in the APHIST relation. Each row corresponds to the implementation of a particular version of a package on a single computer. The dates the version was installed and superceded are noted. If the date superceded is null, then that particular version is currently active. The attribute triple (AP#,COMP#,VER#) is a key for the APHIST relation.

The attributes of the APHIST relation are:

1. **AP#** (Integer; positive)
   The unique identifier for the package.

2. **AP** (Text, 12 characters; upper case)
   The name of the package corresponding to AP#.

3. **VER#** (Integer; positive)
   The unique identifier associated with this version of the package on the named computer.

4. **VER** (Text, 6 characters; free format)
   The name of the version associated with VER#.

5. **COMP#** (Integer; positive)
   The unique identifier for the computer on which this version of the package has been implemented.

6. **COMP** (Text, 6 characters; upper case)
   The name of the computer corresponding to COMP#.

7. **DATEINT** (Integer, 1 unit)
   The date this version of the package was introduced. Format is yymmdd.

8. **DATEDEL** (Integer, 1 unit)
   The date this version of the package was superceded. Format is yymmdd. If this is null, then this version is currently active.

9. **CITATION** (Text, variable length; free format)
   A reference to a monograph or technical article which describes this particular implementation the package. Example: `W. B. Gone, Using WISHPACK on CSCS, User Services Document, 1975.`

2.4 The SUBLIB Relation

This relation provides a machine-independent description of all of the sublibraries known to GAMS. There is one row in SUBLIB for each sublibrary, and the attribute SUBLIB# is a key. The attribute AP is redundant here (it can be determined from the AP relation using AP#), but has been added to simplify queries.

The attributes of the SUBLIB relation are:

1. **SUBLIB#** (Integer; positive)
   A unique identifier for this sublibrary.

2. **SUBLIB** (Text, 12 characters; upper case)
   The name of the sublibrary.

3. **AP#** (Integer; positive)
   The unique identifier associated with the package which contains this sublibrary.

4. **AP** (Text, 12 characters; upper case)
   The name of the package corresponding to AP#.

5. **PORT** (Text, 1 character; E, H, M, or P)
   Indicates restrictions on library usage and ease of transporting the software to other machines. (See AP relation.)

6. **DESC** (Text, variable length; free format)
   A brief description of the sublibrary.

7. **DEVELOP** (Text, variable length; free format)
   The name (and, optionally, the address) of the organization where the sublibrary was developed, and the name of a contact there.

8. **DISTRIB** (Text, variable length; free format)
   The name and address of the organization which is currently distributing the software.

2.5 The SUBLIBX Relation

The SUBLIBX relation contains information about the implementations of sublibraries on each computer system where they are available. Each row corresponds to the implementation of a version of a sublibrary on a single computer system. For example, if a given sublibrary is available on three computer systems, there will be three rows in the SUBLIBX relation corresponding to this sublibrary; they will have the same SUBLIB#, but different COMP#. More precisely, the attribute pair (SUBLIB#,COMP#) forms a key for SUBLIBX.

The attributes of the SUBLIBX relation are:

1. **SUBLIB#** (Integer; positive)
   The unique identifier for the sublibrary. This determines the row of the SUBLIB relation that provides machine-independent information about this sublibrary.

2. **SUBLIB** (Text, 12 characters; upper case)
   The name of the sublibrary corresponding to SUBLIB#.

3. **COMP#** (Integer; positive)
   The unique identifier for the computer on which this sublibrary has been implemented.

4. **COMP** (Text, 6 characters; upper case)
   The name of the computer corresponding to COMP#.

5. **SUPP** (Text, 1 character; 1, 2, or 3)
   The level of support provided users of the sublibrary on the computer. (See APX relation.)

6. **VER#** (Integer; positive)
   The unique identifier associated with the version of the package which contains this sublibrary.

7. **VER** (Text, 6 characters; free format)
   The version name corresponding to VER#.

8. **SUBLIBDC** (Text, variable length; free format)
   A command on the given computer which retrieves detailed documentation for this sublibrary. Name substitution may be used (see the APX relation). Example: `INVOKE,GETDOC,CMLIB,EISPACK`.

9. **CITATION** (Text, variable length; free format)
   A reference to a monograph or technical article which describes the sublibrary.

Two redundant attributes are found in the SUBLIBX relation: COMP and VER. COMP may be determined from the COMPUTER relation using COMP#, and VER may be determined from the APHIST relation using VER#. They have been repeated here to simplify queries.

2.6 The MODULE Relation

This relation provides machine-independent data describing each module. There is one row per module, and the attribute MOD# serves as a key.

The attributes of the MODULE relation are:

1. **MOD#** (Integer; positive)
   A unique identifier for the module.

2. **MODULE** (Text, 12 characters; upper case)
   The module name corresponding to MOD#. If the module name is longer than 12 characters, the remainder are stored at the beginning of the description field.

3. **AP** (Text, 12 characters; upper case)
   The package name corresponding to AP#.

4. **CLASSES** (Text, variable length; fixed format)
   A list of all nodes in the GAMS classification scheme at which this module is classified. The allowable node names are given in the MOD\N relation. Each node name in the list must be followed by a slash (/). Associated with each class is a flag which indicates whether this module is designated by its authors as an easy-to-use version of another module for the problems in that class. If a module is easy-to-use for a given class, then the node name is preceded by an asterisk (*). Example: `G1a1a/K1b1a1/*L8g1a/`

5. **DESC** (Text, variable length; free format)
   A brief description of the module. If the module name was longer than 12 characters, the remainder of the name is stored here from the beginning of the field to a \$.

6. **HOWTO** (Text, variable length; free format)
   Gives a statement that invokes the module. If the module is a subprogram, then the call statement is given. In the case of function subprograms an assignment statement is given, with the variable assigned indicating the type of output (I for integer, S for single precision real, D for double precision real, C for complex, L for logical, and H for character). If the module is a command in an interactive system, then the command is given (if more than one command is required, they should be separated by the two characters \$. If the module is a stand-alone program this field is undefined. Examples: `CALL CCHDC(A,LDA,P,WORK,JPVT,JOB,INFO)` and `S = SASUM (N,NX,INCX)`

7. **ASSOCMOD** (Text, variable length; fixed format)
   The names of modules almost always used in association with this module in order to complete its task. Takes the form of a list of module names (upper case) separated by blanks. The list may be followed by comments in free format beginning with the two characters \$. Examples: `SGEFA SGECO` and `BVALU $ for evaluation.`

8. **AP#** (Integer; positive)
   The unique identifier associated with the package containing this module.

9. **SUBLIB#** (Integer; positive)
   The unique identifier for the sublibrary containing this module. (If the module is in a package which is not partitioned into sublibraries, then the field contains -0-.)

10. **SUBLIB** (Text, 12 character; upper case)
    The name of the sublibrary corresponding to SUBLIB#. (If the module is in an application package which is not partitioned into sublibraries, then the field contains -0-.)

Three redundant attributes are found in the MODULE relation: AP, SUBLIB, and CLASSES. AP may be determined from the AP relation using AP# and SUBLIB may be determined from the SUBLIB relation using SUBLIB#. Except for the "easy-to-use" designations, the information contained in CLASSES can be obtained from the MOD\T relation using MOD#. These have been repeated here to simplify queries.

2.7 The MODULEX Relation

The MODULEX relation contains information about the implementations of individual modules on each computer system where they are available. Each row corresponds to the implementation of a version of a module on a single computer system. For example, if a given module is available on three computer systems, there will be three rows in the MODULEX relation corresponding to this module; they will have the same MODULE#, but different COMP#. More precisely, the attribute pair (MOD#,COMP#) forms a key for MODULEX.

The attributes of the MODULEX relation are:

1. **MOD#** (Integer; positive)
   A unique identifier for this module. This determines the row of the MODULE relation that provides machine-independent information about this module.

2. **MODULE** (Text, 12 characters; upper case)
   The module name corresponding to MOD#.

3. **COMP#** (Integer; positive)
   The unique identifier for the computer on which this module is installed.

4. **COMP** (Text, 6 characters; uppercase)
   The name of the computer corresponding to COMP#.

5. **PREC** (Text, 1 character; D, H, O, or S)
   The precision in which the primary results are returned by the module on the named computer: D ⇒ extra precision (double precision in FORTRAN) S ⇒ standard precision (single precision in FORTRAN) O ⇒ other precision H ⇒

half precision. Note that precision is independent of the data type used in the computation, i.e., integer, real, or complex.

6. **ALTMOD** (Text, variable length; fixed format)
   A list of names of identical modules in this package in other precisions. The list is composed of items of the form NAME/P, separated by commas. NAME is the name of the alternate module, and P is one of the letters S, D, H, or O denoting single, double, half, or other precision, respectively. Example: `DGEFA/D,HGEFA/H`

7. **COMMENT** (Text, variable length; free format)
   Machine-specific remarks. This can be used to provide information about estimated execution times, the status of vectorization on a supercomputer, etc.

8. **AP#** (Integer; positive)
   The unique identifier of the package which contains this module.

9. **AP** (Text, 12 characters; upper case)
   The name of the package corresponding to AP#.

Three redundant attributes are found in the MODULE relation: MODULE, AP, and COMP. MODULE may be determined from the MODULE relation using MOD#, AP may be determined from the AP relation using AP# and COMP may be determined from the COMPUTER relation using COMP#. They have been retained in this relation to simplify queries.

2.8 The COMPUTER Relation

This relation provides background data describing the various computers on which the software catalog in GAMS is implemented. There is exactly one row for each computer, and COMP# is a key. Note that one may use a fairly liberal interpretation of the word computer if necessary. For example, the NOS and NOS/VE subsystems of a single Cyber 855 may be considered to be distinct computers, with distinct versions of each package implemented on them. If more than one Fortran compiler is available on a given system, then each machine–compiler pair may be considered to be a distinct computer for Fortran library implementation purposes.

The attributes of the COMPUTER relation are:

1. **COMP#** (Integer; positive)
   The unique identifier for the computer.

2. **COMP** (Text, 6 characters; upper case)
   A name associated with the computer. It is convenient, though not necessary, to make this name unique. Examples: `CAMVAX`, `205`

3. **LOCATION** (Text, variable length; free format)
   The organizational and/or physical location of this computer.

4. **CONTACT** (Text, variable length; free format)
   The name, address, and phone number of a systems programmer with responsibility for mathematical and statistical software on the given computer.

5. **OPSYS** (Text, variable length; free format)
   The name and version of the operating system running on the computer.

6. **COMPILER** (text,variable length; free format)
   The name of the Fortran compiler(s) primarily used on the computer.

2.9 The MOD\N Relation

The MOD\N relation contains the text of the GAMS classification scheme. The classification scheme is tree-structured. The position of a class in the scheme is encoded in its node name. Two textual descriptions of the class are given, one which is very succinct and another which is more complete. The attribute NODE# is a key for the MOD\N relation.

The attributes of the MOD\N relation are:

1. **NODE#** (Integer; positive)
   A unique identifier associated with this class.

2. **NODE** (Text, 12 characters; fixed format)
   The short name for this class. Alternating single letters and integers. The first character must be an upper case letter and all remaining characters must be lower case. The parts of the class name show the path from the root to this node in the classification tree. Examples: `E2a` and `L10g1a`.

3. **TEXT** (Text, variable length; free format)
   A phrase which succinctly describes this class. This phrase may assume that the context of the class (i.e. its parent and siblings) are apparent to the reader. Example: `Gridded data`.

4. **VERBOSE** (Text, variable length; free format)
   A phrase which completely describes this class. In contrast to TEXT, one should not assume that the context of the node is apparent to the reader. Example: `Interpolation of two-dimensional gridded data`.

2.10 The MOD\T Relation

The MOD\T relation is used to represent the tree structure of the classification scheme and also to show which modules are classified at which nodes. Each row of this relation is an ordered pair (sup,inf) that denotes a (parent,child) relationship between two classes or a class and a module. There is one row in MOD\T for each row in MOD\N which does not represent a leaf of the classification tree, and there is at least one row for each row in the MODULE relation. (There is more than one row for modules which are classified at more than one node). Although the information encoded in this relation is already implicitly available in the MOD\N and MODULE relations, it is made available here in an abbreviated form in order to facilitate faster processing.

The attributes of the MOD\T relation are:

1. **SUP** (Integer; positive)
   The unique identifier associated with a class in MOD\N.

2. **INF** (Integer; non-zero)
   The unique identifier for a child of the class denoted by SUP. If INF > 0, then it is the unique identifier associated with a class in MOD\N. In this case, a row in this relation (SUP,INF) denotes that class INF is a subclass of class SUP. If INF < 0, then $-$INF is the unique identifier associated with a module in the MODULE relation. In this case, a row in this relation (SUP,INF) denotes that module number $-$INF is classified in class SUP.

2.11 Database Implementation

To implement the database we use the RIM database management system (RTIRIM) [6]. RTIRIM has been implemented for a wide variety of computer systems. It includes an SQL-like interactive query system and a Fortran interface.

The following text can be used to define the schema of the GAMS database to RTIRIM. Note that two additional relations are defined. ATTDEF is used to store information about the attributes in the GAMS database. Its attributes are ATTDESC (a description of the attribute), ATTFORM (the format of the attribute), and ATTEX (an example). DEAD\N is used to save rows of MOD\N which have been deleted using the program REVISETREE (see next section); this information is later used by the program RECLASSIFY.

```
DEFINE GAMS

ATTRIBUTES

 NODE#          INT            1   KEY
 NODE           TEXT          12   KEY
 TEXT           TEXT         VAR        FORMAT 20
 VERBOSE        TEXT         VAR        FORMAT 20
 SUP            INT            1   KEY  FORMAT  4
 INF            INT            1   KEY  FORMAT  6
 COMP#          INT            1   KEY  FORMAT  4
 COMP           TEXT           6   KEY
 LOCATION       TEXT         VAR        FORMAT 20
 CONTACT        TEXT         VAR        FORMAT 20
 OPSYS          TEXT         VAR        FORMAT 20
 COMPILER       TEXT         VAR        FORMAT 20
 AP#            INT            1   KEY  FORMAT  4
 AP             TEXT          12   KEY
 TYPE           INT            1        FORMAT  4
 PORT           TEXT           1        FORMAT  4
 DESC           TEXT         VAR        FORMAT 20
 LANG           TEXT          12
 DEVELOP        TEXT         VAR        FORMAT 20
 CITATION       TEXT         VAR        FORMAT 20
 DISTRIB        TEXT         VAR        FORMAT 20
 SUPP           TEXT           1        FORMAT  4
 ACCESS         TEXT         VAR        FORMAT 20
 VER#           INT            1        FORMAT  4
 VER            TEXT           6
 LIBDOC         TEXT         VAR        FORMAT 20
 MODDOC         TEXT         VAR        FORMAT 20
 SAMPLE         TEXT         VAR        FORMAT 20
 SOURCE         TEXT         VAR        FORMAT 20
 TESTS          TEXT         VAR        FORMAT 20
 DATEINT        INT            1
 DATEDEL        INT            1
 SUBLIB#        INT            1   KEY  FORMAT  8
 SUBLIB         TEXT          12   KEY
 SUBLIBDC       TEXT         VAR        FORMAT 20
 MOD#           INT            1   KEY
 MODULE         TEXT          12   KEY
 CLASSES        TEXT         VAR        FORMAT 20
```

```
 HOWTO          TEXT          VAR          FORMAT 20
 ASSOCMOD       TEXT          VAR          FORMAT 20
 PREC           TEXT            1          FORMAT  4
 ALTMOD         TEXT          VAR          FORMAT 20
 COMMENT        TEXT          VAR          FORMAT 20
 ATTNAM         TEXT            8
 ATTDESC        TEXT          VAR          FORMAT 20
 ATTFORM        TEXT          VAR          FORMAT 20
 ATTEX          TEXT          VAR          FORMAT 20

RELATIONS

 MOD\N     WITH NODE#     NODE      TEXT      VERBOSE
 MOD\T     WITH SUP       INF
 COMPUTER  WITH COMP#     COMP      LOCATION  CONTACT      +
                OPSYS     COMPILER
 AP        WITH AP#       AP        TYPE      PORT         +
                DESC      LANG      DEVELOP   CITATION     +
                DISTRIB
 APX       WITH AP#       AP        COMP#     COMP         +
                SUPP      ACCESS    VER#      VER          +
                LIBDOC    MODDOC    CITATION  SAMPLE       +
                SOURCE    TESTS
 APHIST    WITH AP#       AP        VER#      VER          +
                COMP#     COMP      DATEINT   DATEDEL      +
                CITATION
 SUBLIB    WITH SUBLIB#   SUBLIB    AP#       AP           +
                PORT      DESC      DEVELOP   DISTRIB
 SUBLIBX   WITH SUBLIB#   SUBLIB    COMP#     COMP         +
                SUPP      VER#      VER       SUBLIBDC     +
                CITATION
 MODULE    WITH MOD#      MODULE    AP        CLASSES      +
                DESC      HOWTO     ASSOCMOD  AP#          +
                SUBLIB#   SUBLIB
 MODULEX   WITH MOD#      MODULE    COMP#     COMP         +
                PREC      ALTMOD    COMMENT   AP#          +
                AP
 DEAD\N    WITH NODE#     NODE
 ATTDEF    WITH ATTNAM    ATTDESC   ATTFORM   ATTEX

END
```

Chapter 3

GAMS Programs and Procedures

There are five principle programs for manipulating the GAMS database:

1. GAMS Interactive Consultant (GAMSIC)
   Allows users to interactively search for mathematical and statistical software for solving problems of interest to them. This is an on-line version of the GAMS catalog [2]. It is the only GAMS program designed to be called by users.

2. MODIFY
   Adds and deletes modules, sublibraries, or packages from the GAMS database.

3. UPDATE
   An interactive program for preparing input data for MODIFY.

4. REVISETREE
   Adds and deletes subtrees from the GAMS Classification Scheme.

5. RECLASSIFY
   An interactive program used to specify changes in module classifications.

An additional set of programs aids in the production of the printed GAMS catalog [2].

3.1 The GAMS Interactive Consultant

The GAMS Interactive Consultant (GAMSIC) is a program designed to help a computer user locate mathematical and statistical software for solving particular computational problems. When using GAMSIC a person traverses a tree-structured classification scheme of mathematical and statistical problems. The user is alerted when software is available for solving the problem described by the current tree node, and the user may then obtain a short summary describing each of these software modules. These summaries also give information about how to obtain detailed documentation on each module; this documentation must be obtained outside the GAMSIC system.

Table 3.1: Files Required by GAMSIC

Name	Unit	Type	Description
GAMS1.DAT		input	
GAMS2.DAT		input	GAMS/RIM database
GAMS3.DAT		input	
Standard input	IUNIT	input	user's terminal
ICHELP.TXT	HUNIT	input	help text
Standard output	OUNIT	output	user's terminal

GAMSIC provides users with complete information on its own usage. When GAMSIC begins execution a short message giving news of the GAMS project is displayed and the user is told to type a question mark (?) for help. Various help texts may be retrieved by typing ?name , where name is the title of the help text.

The command structure of GAMSIC is quite simple. A user is always positioned at a particular node of the problem classification tree. When first positioned at a node the user is given a list of all children of the current node and a count of the number of modules available for the associated problem. Typing a minus sign (-) moves to the parent of the current node, while typing a plus sign (+) followed by one or more characters moves to the child of the current node whose name is obtained by appending the given characters to the current node's name. (For example, when at node L8a, typing +2 moves one to the node L8a2). If modules have been classified at the current node, then entering a carriage return obtains a description of the first one; subsequent carriage returns retrieve information about additional modules classified there, if any. At any time a user may go immediately to any class by simply typing its name.

Optional commands are also available which allow users to declare that they are only interested in modules with certain specific properties. These properties include portability, precision, membership in a given library, or availability on a particular computer system.

GAMSIC is implemented in Fortran 77. Access to GAMS data is obtained by calling Fortran subroutines of the RIM applications program interface. The files required by GAMSIC are given in Table 3.1.

The file ICHELP.TXT contains a collection of text records. Each record may contain many lines, but must begin with a line containing the characters CCCCname beginning in column 1, where name is the name of the record. GAMSIC processes the command ?name by searching the ICHELP.TXT file for a record of the given name; this record, if found, is copied to the standard output file. Three records with fixed names must appear in this file:

1. DBNAME
   the RIM name for the GAMS database.

2. NEWS
   a news item which is displayed whenever GAMSIC begins execution.

3. HELP
   the help text to be printed when a user types a question mark ?.

3.2 GAMS Maintenance Procedures

In this section we describe how some typical maintenance tasks can be performed on the GAMS/RIM database. There are two principle means for changing the database: interactively through the RIM Query Language or indirectly through the GAMS maintenance programs listed above. Regardless of which method is chosen, updating the database should always be a four-step process:

1. Make a temporary copy of the database files:

   ```
   GAMS1.DAT
   GAMS2.DAT
   GAMS3.DAT.
   ```

   Note that RIM is quite particular about the names of the database files, i.e., the copies must also be named GAMS1.DAT, GAMS2.DAT, and GAMS3.DAT. The easiest way to arrange this is to put the copies in a different directory than the originals.

2. Make the required changes to the temporary database files.

3. Use the RIM Query Language and/or GAMSIC to check whether the changes succeeded.

4. Replace the original GAMS/RIM database files by the temporary ones.

3.2.1 Fixing Errors in the Data

Errors in the AP, APX, SUBLIB, SUBLIBX, MODULE, and MODULEX relations (excluding the CLASSES attribute) are best fixed using the RIM Query Language. Spelling errors, for example, can easily be remedied using the CHANGE or EDIT commands. Changing a given attribute in many rows of a relation can often be accomplished efficiently with these commands.

In some cases it may be more convenient to use the more extensive facilities of a text editor to edit data outside of the RIM system. The UNLOAD command can be used

to move all data from a given relation to a text file in a form suitable for subsequent reloading. This file may be edited to effect any desired changes in the data. Upon re-entering the RIM system one can delete all rows in the previously unloaded relation and then use the INPUT command to load the edited data.

There are several types of modifications to the GAMS/RIM database that should never be attempted with the RIM Query System. These include adding or deleting modules, sublibraries, and packages, reclassifying existing modules, and modifying the GAMS classification scheme. (In particular, the following attributes should never be modified: MOD#, CLASSES, NODE#, NODE, SUP, INF.) Such changes affect more than one relation and necessary changes are often overlooked when done by hand. The GAMS maintenance programs listed in section 3 have been provided for this purpose. In the following sections we describe how they can be used to perform these tasks.

3.2.2 Adding and Deleting Packages and Modules

This is the maintenance task performed most often. Two programs must be run to do this: UPDATE and MODIFY. UPDATE is run interactively to specify the additions and deletions to be made to the database. It prompts the user for all necessary data and checks them for validity. Alternately, module data to be entered in UPDATE can be prepared first in a file in a fixed format (see the Appendix); upon request UPDATE will read from this file rather than prompt the user for each data item.

When processing module data from a file, UPDATE reports on the success or failure of each task performed. This output goes to the screen and to an output log file, UPDATE.LOG. If some input data items were in error, the input file should be corrected and UPDATE rerun.

UPDATE does not change the database itself; instead it writes a data file named MODCOM.DAT which contains directives and data in a fixed format. Errors made while entering data with UPDATE, such as spelling errors in module descriptions, can be fixed by editing the MODCOM.DAT file. (Caution: MODCOM.DAT has a fixed format; see the Appendix.) The companion program MODIFY can then be run to read this data file and effect the changes in the database.

3.2.3 Changing the Classification Scheme

Two types of revisions to the classification scheme are distinguished. Simple changes such as fixing typographical errors or clarifying the wording in the node descriptions are best made by editing the MOD\N relation with the RIM Query System. More substantial changes such as the addition or deletion of new nodes must be made using the GAMS program REVISETREE.

REVISETREE runs in batch mode, i.e., one first prepares a data file containing REVISETREE commands and then runs REVISETREE specifying this input file. REVISETREE has two basic commands: one to add nodes and one to delete nodes. Both

work only on subtrees; that is, one adds an entire subtree or deletes an entire subtree. Each operation must result in a valid tree-structured classification scheme. REVISETREE will neither add nodes that already exist in the classification scheme, nor add nodes that do not have an existing parent in the classification scheme. Thus, the order of commands in the input data file is important.

After REVISETREE is run, the program RECLASSIFY can be used to update the classifications of modules which were affected by the revision.

3.2.4 Reclassifying Modules

One should never attempt to reclassify modules except through the program RECLASSIFY. This is because several relations are affected by a reclassification and getting it right using the RIM Query System itself is not easy.

There are two reasons for reclassifying a module: to fix an error in its existing classification(s), or to reclassify a module affected by a REVISETREE operation. In the first case both the existing class and the new class come from the *current* classification scheme. In the second case the existing class is from a *previous* scheme, while the new class is from the *current* scheme. The RECLASSIFY program asks which of these cases applies.

One can reclassify particular modules if their names are known, or one can choose to reclassify all the modules in a given class. RECLASSIFY is interactive; the user is presented with the names, short descriptions, and existing class(es) of selected modules, and the requested changes are applied to the database immediately.

3.3 GAMS Maintenance Programs

In this section we describe the Fortran programs that are used to maintain the GAMS database in more detail. Each of these programs accesses the database using the Fortran-callable subprograms of the RIM applications program interface.

3.3.1 MODIFY

MODIFY is used to add modules, sublibraries, and libraries to the GAMS database, as well as to delete them. Input to MODIFY is a file containing a set of addition and deletion commands in a rigid format. There are thirteen basic commands; note that each command name must begin with an asterisk in column one.

1. `*ADD AP`
   Adds summary data describing an application package. Relations affected: AP.

2. `*ADD APX`
   Adds summary data describing the implementation of an application package on a particular computer. Relations affected: APX.

3. *ADD APHIST
   Adds historical data describing implementation of an application package on a particular machine. Relations affected: APHIST.

4. *ADD SUBLIB
   Adds summary data describing a sublibrary. Relations affected: SUBLIB.

5. *ADD SUBLIBX
   Adds summary data describing the implementation of a sublibrary on a particular computer. Relations affected: SUBLIBX.

6. *ADD MODULE
   Adds data describing a particular software module. Relations affected: MODULE, MOD\T.

7. *ADD MODULEX
   Adds data describing the implementation of a software module on a particular computer. Relations affected: MODULE, MOD\T.

8. *DEL AP
   Deletes summary data describing an application package. The data are not deleted if there exist any implementations of this package (i.e., rows in APX) or any sublibraries or modules (i.e., rows in SUBLIB or MODULE) associated with this package in the current database. Relations affected: AP.

9. *DEL APX
   Deletes data describing the implementation of a package on a particular computer. The data are not deleted if there exist any sublibraries or modules (i.e., rows in SUBLIBX or MODULEX) associated with this package implementation. Relations affected: APX, APHIST

10. *DEL SUBLIB
    Deletes summary data describing a sublibrary. The data are not deleted if there exist any implementations of this sublibrary (i.e., rows in SUBLIBX) or modules (i.e., rows in MODULE) associated with this sublibrary in the current database. Relations affected: SUBLIB

11. *DEL SUBLIBX
    Deletes data describing the implementation of a sublibrary on a particular computer. The data are not deleted if there exist any modules (i.e., rows in MODULEX) associated with this sublibrary implementation. Relations affected: SUBLIBX.

Table 3.2: Files Required by MODIFY

Name	Unit	Type	Description
GAMS1.DAT		input	
GAMS2.DAT		input	GAMS/RIM database
GAMS3.DAT		input	
Standard input	IUNIT	input	user's terminal
MODCOM.DAT	FUNIT	input	user's MODIFY commands & data
Standard output	OUNIT	output	user's terminal
MODIFY.LOG	LUNIT	output	log file

12. *DEL MODULE
    Deletes summary data describing a module. The data are not deleted if there exist any implementations of this module (i.e. rows in MODULEX) associated with this module in the current database. Relations affected: MODULE, MOD\T

13. *DEL MODULEX
    Deletes data describing the implementation of a module on a particular computer. Relations affected: MODULEX.

Each delete command is followed by data giving the name of the particular item to be deleted. Each add command is followed by a variable number of lines containing the data to be loaded into the database. The format of these data lines is given in the Section 3.7. MODIFY does little or no checking of its input data lines.

The files required by MODIFY are listed in Table 3.2. (The user is prompted for the actual name of the file MODCOM.DAT at runtime.)

3.3.2 UPDATE

UPDATE is a preprocessor which can be used to generate the MODCOM.DAT file which is the input to MODIFY. In comparison to MODIFY, which is intended to run in batch mode, UPDATE is a friendly interactive program with help facilities and some degree of input data checking.

When UPDATE begins execution, the user is presented with a menu of six add and delete commands. When a choice is made the user is prompted for each data item required to prepare input for MODIFY. If the user types a question mark (?) in response to any of the prompts then a short description of the data item is displayed along with examples. UPDATE checks the data for consistency and prompts the user to re-enter bad data.

The add commands allow modules, sublibraries, or application packages to be added as new items, or, if they already exist in the data base, to be added as imple-

Table 3.3: Files Required by UPDATE

Name	Unit	Type	Description
GAMS1.DAT		input	
GAMS2.DAT		input	GAMS/RIM database
GAMS3.DAT		input	
Standard input	IUNIT	input	user's terminal
MODIN.DAT	MUNIT	input	user's module data (optional)
UPDATEHELP.DAT	HUNIT	input	help text
Standard output	OUNIT	output	user's terminal
MODCOM.DAT	FUNIT	output	MODIFY commands and data
UPDATE.LOG	LUNIT	output	log file

mentations on a new computer. In addition one may add a new version of an existing package. When adding packages or modules, the user has the option of prestoring data in a file for UPDATE to read. The format of this file is specified in Section 3.8.

The delete commands prompt the user for the name of the item to delete. The user may type the word ALL to denote all items of the indicated type, e.g., all modules, all sublibraries, or all computers. The order of deletions is important:

- All package, sublibrary, or module implementations must be deleted before the corresponding package, sublibrary, or module can be deleted.
- All modules in a given sublibrary must be deleted before the sublibrary itself is deleted.
- All sublibraries and modules in a given package must be deleted before the package itself is deleted.

UPDATE does not check whether these precedence rules are followed; violating them will result in error messages when MODIFY runs. The user is prompted to verify each delete command.

The files required by UPDATE are given in Table 3.3. (The actual names to be used for files MODIN.DAT and MODCOM.DAT are specified by the user at run time.)

3.3.3 REVISETREE

REVISETREE is used to add or delete nodes from the GAMS classification scheme. It is primarily batch-oriented, taking its commands from a file which has a specific format. There are two commands; note that each command name must begin with an asterisk in column one.

Table 3.4: Files Required by REVISETREE

Name	Unit	Type	Description
GAMS1.DAT		input	
GAMS2.DAT		input	GAMS/RIM database
GAMS3.DAT		input	
Standard input	IUNIT	input	user's terminal
REVCMD.DAT	FUNIT	input	user's REVISETREE commands
Standard output	OUNIT	output	user's terminal
REVISE.LOG	LUNIT	output	log file

1. *DEL NODES
   Deletes the named classification node and all descendant nodes. The node name is given on the line immediately following the command; it must begin in column one. Relations affected: MOD\N, MOD\T.

2. *ADD NODES
   Adds one or more classification nodes. Each classification has three parts: a name, a short description, and a long description. Each part starts on a new line; the descriptions may be continued for up to six lines and must be terminated with the character #. The node name must begin in column one and must use the alternating letter-number scheme described in [1]. The short description should be appropriate for use when the entire scheme is printed in outline form, while the long description should allow the node to be recognizable when seen out of context. Relations affected: MOD\N, MOD\T.

REVISETREE checks for various error conditions before attempting to add or delete nodes. A node is not added if there is no existing parent in the classification scheme or if a node with the same name already exists. Appropriate error messages are printed on standard output to report these occurrences.

A special RIM relation named DEAD\N is used by REVISETREE to save rows of the MOD\N relation which have been deleted from the classification scheme. This relation is subsequently used by the program RECLASSIFY to determine which modules must be reclassified as a result of node deletions in REVISETREE.

The files required by REVISETREE are given in Table 3.4. (The user is prompted for the actual name of the file REVCMD.DAT at runtime.)

3.3.4 RECLASSIFY

RECLASSIFY is an interactive program used to change the classification of one or more modules in the GAMS database, as well as to check the consistency of classification

Table 3.5: Files Required by RECLASSIFY

Name	Unit	Type	Description
GAMS1.DAT		input	
GAMS2.DAT		input	GAMS/RIM database
GAMS3.DAT		input	
Standard input	IUNIT	input	user's terminal
ICHELP.DAT	HUNIT	input	help text
Standard output	OUNIT	output	user's terminal
TEMP.DAT	CUNIT	output/input	scratch file

data in the database (the latter operation is referred to as verifying the GAMS tree).

When RECLASSIFY begins execution, the user is presented with a menu. The user may choose to reclassify a particular module, to reclassify all modules in a given class, or to verify the GAMS tree. The class name may be from the current GAMS tree or one previously deleted by REVISETREE. When reclassifying by class, the user has the option of whether to reclassify only modules in the named class or modules in the named class and its descendent classes.

For each module the user is presented with the module name, a short description, and a list of its current classifications; the user then may give a set of new classifications for the module. Optionally, the user can decide not to reclassify the module.

GAMS tree verification involves checking whether each module in the database is classified only at nodes currently known to the database. If the verification is successful, then one is sure that all modules affected by the deletion of nodes by REVISETREE have been reclassified under the new scheme; if the process fails the names of modules which require reclassification are printed. Note that this process is quite time-consuming since the classifications of each module in the database must be checked. When the GAMS tree is successfully verified, then one should delete all rows from the DEAD\N relation (see section 3.3.3).

The files required by RECLASSIFY are given in Table 3.5.

3.4 Portability of the GAMS System

The GAMS system is portable only in a limited sense. GAMS uses the RIM database management system, and hence this software must also be available. At NIST we use RTIRIM [6]. This software is currently available on a large variety of systems. There is no guarantee that GAMS will run under any other version of RIM.

Two steps are required to move GAMS from one system to another: moving the database and moving the Fortran programs which perform GAMS query and mainte-

nance functions. RIM databases may easily be moved between computer systems of different types using the RIM Query System's OUTPUT/UNLOAD and INPUT commands. The former commands can be used to build a text file which contains the database schema and all data in a machine-independent form suitable for subsequent loading using the INPUT command.

The GAMS Fortran programs are coded in portable Fortran 77. The machine-dependent parts have been localized in a few subroutines to ease the transfer to different machines. These subprograms are:

1. Initialization routines
   One of these subprograms exists for each main program (GAMSIC, MODIFY, UPDATE, RECLASSIFY, REVISETREE), and a call to this routine is the first executable statement in each of these programs. This routine is used to open all files, set machine-dependent constants, and perform any other machine-specific preprocessing.

2. Finalization routines
   One of these subprograms exists for each main program, and a call to this routine is the last executable statement in each of these programs. These routines are used to perform machine-dependent postprocessing.

3. Subroutine GETLN
   This subprogram is used to read an 80-character record from an interactive user's terminal.

4. GAMS/RIM Interface Routines
   The GAMS programs interact with the RIM database through RIM-supplied Fortran-callable subprograms; this is the "RIM applications program interface". Unfortunately, the RIM subprograms are coded in Fortran 66, where CHARACTER variables are not available. Thus, character variables in the GAMS programs must be passed to RIM subroutines where they are declared with a numeric data type. This cannot be done directly, and the method for doing so is machine-dependent. Thus, we provide a collection of short GAMS/RIM interface routines which perform this translation for each system. The interface routines have the same names as their RIM counterparts, except that the first two characters (always RM) are replaced with ZZ.

The machine-dependent constants which must be set in the initialization routines are given in Table 3.6.

In the following sections we describe the status of our conversion efforts.

3.4.1 Cyber NOS Systems

At NIST, users access the GAMS Interactive Consultant on a Cyber 180/855 (NOS 2.4, FTN5 compiler). Because we maintain GAMS on a Vax, we have not attempted

Table 3.6: Machine-Dependent Variables in GAMS Programs

Variable	Description
NCPW	Number of characters which can be stored in a single word.
NWPI	Number of words in an integer variable.
NWPR	Number of words in a real variable.
NWPD	Number of words in a double precision variable.
CUNIT	Unit number of a scratch file. (i/o)
FUNIT	Unit number of the command file. (i/o)
HUNIT	Unit number of the help file. (input)
IUNIT	Unit number of standard input (user's terminal).
LUNIT	Unit number of the log file. (output)
MUNIT	Unit number of an input file for UPDATE.
OUNIT	Unit number of standard output (user's terminal).

to install any of the GAMS maintenance programs under NOS; subtle changes will undoubtably be necessary to implement the maintenance programs there.

GAMS programs assume that lower-case characters are representable as single characters and will not operate if they are instead represented in the Cyber's 6/12 ASCII code. Thus, all GAMS programs and data must be first converted to upper case before they can be used under NOS.

The NOS implementation of the machine-dependent subprograms have several unique features. The initialization and finalization routines call COMPASS routines to determine the current terminal session mode (NORMAL or ASCII), set the user to ASCII, and finally to restore the previous mode before exit. This is required to allow users to type lowercase characters in spite of the fact that GAMSIC only recognizes uppercase characters under NOS. The subroutine GETLN must be modified so that a user can type carriage return as a response to GAMSIC. Under NOS, a null line triggers an end-of-file condition, and hence the file INPUT must be closed and reopened in this case. The GAMS/RIM interface routines must communicate with RIM in character variables whose length is a multiple of 10; temporary variables are set up in these routines to effect the translation.

3.4.2 Cyber NOS/VE Systems

At NIST, users access the GAMS Interactive Consultant on a Cyber 180/855 (NOS/VE 1.3.1, FORTRAN compiler). Because we maintain GAMS on a Vax, we have not attempted to install any of the GAMS maintenance programs under NOS/VE; changes will undoubtably be necessary to implement the maintenance programs there.

Since NOS/VE uses the ASCII character set there are no problems associated with upper/lower case characters as there are under NOS.

3.4.3 Sperry EXEC Systems

GAMS was originally developed on a Sperry 1100/82 EXEC system (ASCII Fortran) using RIM (Version 6) as distributed by Boeing Computer Services. We no longer maintain a Sperry implementation, and BOEING RIM (version 7) is not being prepared for Sperry systems. Thus, those interested in GAMS for a Sperry computer must use RIM version 6. Our experience with this system would indicate that the conversion to this system would not be difficult at this time. In particular, the GAMS/RIM interface routines are not required on this machine.

3.4.4 Vax VMS Systems

All GAMS development and maintenance at NIST is done on a Vax 11/785 running VMS 4.7. Thus, porting GAMS to another VMS site should pose no problems. When GAMS is sent to a Vax VMS site, a tape is made, in backup format, of all files in [GAMS] (see section 3.5).

The only non-standard features in the GAMS machine-dependent subprograms occur in the GAMS/RIM interface routines, where character variables must be passed to RIM subprograms using the %REF primitive so that the RIM subprograms can declare them using a numeric data type.

3.5 GAMS Implementation Under Vax VMS

All GAMS maintenance and development work at NIST is performed on a Vax VMS system. This section provides information about the files and procedures used there. When the GAMS system is distributed copies of these items are made available on a VMS backup tape.

We have assumed that the Vax system administrator has defined two global symbols:

- RIM : executes the RIM Interactive Query System
- RIMSHR : the RIM Application Programmer's Interface Library

RIMSHR is a sharable image library; it precludes the use of the common block /RIMCOM/ or the subroutine RMSTAT in the GAMS programs. We also assume that the directory [GAMS] resides on DISK$USER, i.e., its fully-qualified name is DISK$USER:[GAMS].

3.5.1 Directory GAMS

All files required by GAMS have been placed in the directory [GAMS] or its subdirectories. Found here are the publicly-accessible copies of the GAMS RIM database and VMS CCL procedures which execute each of the GAMS programs. These files are listed in Table 3.7.

Table 3.7: Files in Directory [GAMS]

Name	Description
GAMS1.DAT	the GAMS RIM database (part 1)
GAMS2.DAT	the GAMS RIM database (part 2)
GAMS3.DAT	the GAMS RIM database (part 3)
COMFILE.DAT	GAMS database in RIM UNLOAD format
GAMS.COM	Runs the Interactive Consultant
MODIFY.COM	Runs MODIFY
RECLAS.COM	Runs RECLASSIFY
REVTREE.COM	Runs REVISETREE
UPDATE.COM	Runs UPDATE
LIB.DIR	GAMS RIM Fortran subroutine library
PROC.DIR	Utility CCL procedures
PROG.DIR	The GAMS access and maintenance programs

In order for RIM to access the database files, the protection on the files GAMS1.DAT, GAMS2.DAT, and GAMS3.DAT files must be *read* and *write* for all users. Because of this the GAMS database administrator must maintain an additional back-up copy of the database files with exclusive write protection. The protection on the remaining files in this directory is *read* and *execute*. The protection on the user's working directory must be *read*, *write*, and *execute* so that RIM can write temporary files there.

On our system a global symbol, GAMS, has been set up, vis.,

```
GAMS :== "@DISK$USER:[GAMS]GAMS.COM"
```

which runs the GAMS Interactive Consultant.

In addition to these files, there are three subdirectories (see Table 3.7) which contain the source, object, and executables of all GAMS programs, as well as numerous helpful CCL procedures. These are described in the following sections.

3.5.2 Subdirectory [GAMS.PROG]

This subdirectory contains a subdirectory for each principle GAMS program (GAMSIC, UPDATE, MODIFY, RECLASSIFY, REVISETREE). Each of these subdirectories contains the following:

- A .FOR file containing the program source.
- A .EXE file containing the executable version of the program.
- A .COM file which can be used to build the .EXE file by compiling its source and linking it to all necessary libraries.

- A .TXT file containing help text for use by the program, if necessary.

Two libraries are used by the .COM file which builds each program: a library of GAMS utilities ([GAMS.LIB]GAMSLIB.OLB), and the RIM Application Programmer's Interface library (the global symbol RIMSHR). In addition, any required machine-dependent routines are loaded from the directory

[GAMS.LIB.MACHDP.VAX].

3.5.3 Subdirectory [GAMS.LIB]

This subdirectory is used to maintain a collection of Fortran subprograms which are used by the GAMS programs. The contents of this directory are

- GAMSLIB.OLB
  A precompiled library of the GAMS Fortran utility routines.
- MAKEGAMSLIB.COM
  A CCL procedure to generate GAMSLIB.OLB from its source.
- SOURCE.DIR
  A directory containing the Fortran source for GAMSLIB.OLB. Each subprogram occupies a single .FOR file in this directory.
- MACHDP.DIR
  A directory containing all machine-dependent parts of the GAMS programs. This directory contains subdirectories for each machine implementation of GAMS (currently only Vax VMS and Cyber NOS are provided. Within these directories are found the source for the items described in Section 3.4.

3.5.4 Subdirectory [GAMS.PROC]

This subdirectory contains a collection of CCL procedures which are used in maintaining the database at NIST. The following is a brief outline of what each one does. Note that here, the private, most current copy of the GAMS database is maintained in the files

[SPRING.GAMS]DB1.DAT, DB2.DAT, DB3.DAT.

All updates are applied to these files, which are periodically copied to the publicly-accessible versions,

[GAMS]GAMS1.DAT, GAMS2.DAT, GAMS3.DAT.

- MAKETEMPDB
  Copies the files [SPRING.GAMS]DB1.DAT, DB2.DAT into the current directory, renaming them GAMS1.DAT, GAMS2.DAT, GAMS3.DAT.

- SAVETEMPDB
  Saves the files named GAMS1.DAT, GAMS2.DAT, GAMS3.DAT in the current directory in the files [SPRING.GAMS]DB1.DAT, DB2.DAT.

- REPLACE
  Updates the publicly-accessible GAMS database files. In particular,
  [GAMS]GAMS1.DAT, GAMS2.DAT, GAMS3.DAT
  are replaced by the contents of
  [SPRING.GAMS]DB1.DAT, DB2.DAT, DB3.DAT,
  respectively.

- RIM
  Makes a copy of the database files [SPRING.GAMS]DB1.DAT, DB2.DAT into the current directory, renaming them GAMS1.DAT, GAMS2.DAT, GAMS3.DAT, enters the RIM Query System, allowing these database files to be edited. After the editing session, the procedure, at the user's request, will copy the updated database files back to DB1.DAT, DB2.DAT, DB3.DAT and add entries in the file [SPRING.GAMS]DB.LOG which records changes made to the GAMS database.

- NEWLINK
  Updates the GAMS utility library [GAMS.LIB]GAMSLIB.OLB. Specifically, a file with suffix .FOR specified by the user is copied from the current directory to the directory [GAMS.LIB.SOURCE], the file is then compiled and the object module is included in the library [GAMS.LIB]GAMSLIB.OLB. All GAMS programs are then relinked using the new library.

- UNLOAD
  Runs MAKETEMPDB.COM, then enters the RIM Query System and executes RIM commands to unload the database to the file COMFILE.DAT in the current directory. The file [GAMS.PROC]UNLOAD.DAT contains the RIM commands which perform the unloading operation.

3.6 Programs Under Development

Another application of the GAMS database has been the development of a printed catalog of mathematical and statistical software [2]. Various programs have been written to query the database and produce formatted sections of this document. Versions with output formatted as a standard text file and instrumented with commands for the LaTeX text-formatting system [4] have been produced. These programs include:

- GCS (GAMS Classification Scheme)
  Lists the GAMS classification scheme.

- MBC (Modules by Class)
  Lists modules descriptions in order of the GAMS classification scheme.
- APDICT (AP Dictionary)
  Lists detailed information about each application package in sorted order.
- MODICT (Module Dictionary)
  Lists detailed information about each module in sorted order.

3.7 MODIFY Input Data Format

MODIFY accepts ten different commands (indicated by an asterisk in column 1, each followed by lines of data in a fixed format. The templates below illustrate the format of these data lines. Data fields are denoted by the database attribute names to which they correspond (in lower case). In addition, the following abbreviations are used:

```
   P  =  prec
   Q  =  port
   S  =  supp
XXXX  =  mod#
  YY  =  sublib#
  ZZ  =  ap#
   N  =  comp#
```

Variable-length text attributes which may occupy more than one line of input are shown ending with the character #; this character must be included to indicate the end of the field. In each case below the first two lines are used to indicate column positions and are not part of MODIFY input data.

1. Adding modules

```
000000000111111111122222222223333333333444444444455555
123456789012345678901234567890123456789012345678901234
*ADD MODULE
module
sublib      YYap
classes
howto#
assocmod#

*ADD MODULEX
XXXXmodule      Ncomp  P
ZZap
```

```
altmod#
comment#
```

2. Adding sublibraries

```
000000000111111111122222222233333333334444444444555555
123456789012345678901234567890123456789012345678901234
*ADD SUBLIB
sublib      ap          Q
desc#
develop#
distrib#

*ADD SUBLIBX
YYsublib      Ncomp
 Over 1 S
sublibdc#
citation#
```

3. Adding libraries

```
000000000111111111122222222233333333334444444444555555
123456789012345678901234567890123456789012345678901234
*ADD AP
ap          SQlang
desc#
develop#
citation#
distrib#

*ADD APX
ZZap           ver   Ncomp  S
citation#
access#
libdoc#
moddoc#
sample#
source#
tests#

*ADD APHIST
ZZap       ver    Ncomp
```

```
dateint
citation#
```

4. Deleting modules:

```
000000000111111111122222222233333333334444444444555555
123456789012345678901234567890123456789012345678901234
*DEL MODULEX
XXXXmodule
 Ncomp
*DEL MODULE
XXXXmodule
```

5. Deleting sublibraries:

```
000000000111111111122222222233333333334444444444555555
123456789012345678901234567890123456789012345678901234
*DEL SUBLIBX
YYsublib      Ncomp
*DEL SUBLIB
YYsublib
```

6. Deleting libraries:

```
000000000111111111122222222233333333334444444444555555
123456789012345678901234567890123456789012345678901234
*DEL APX
ZZap          ver    Ncomp
datedel
*DEL AP
ZZap
```

3.8 UPDATE Input Data Format

Normally, UPDATE prompts the user for information about each new module which is to be added to the database. Sometimes it is more convenient to prepare this information and place it in a file ahead of time. UPDATE gives the user the option of doing this, and in this appendix we specify the format of such a data file.

Data for four different relations may appear in such a file: AP, APX, MODULE, and MODULEX. For each package to be added there must be exactly one *ADD AP record which provides all machine-independent data about the package. This may

be followed by one or more *ADD APX records, each of which provides data about the implementation of the package on a particular computer. Similarly, for each module to be added there must be exactly one *ADD MODULE record which provides all machine-independent data about the module. This may be followed by one or more *ADD MODULEX records, each of which provides data about the implementation of the module on a particular computer. All modules in a given file must be contained in a single application package, and at most one AP should be represented in any given file.

The format of the *ADD MODULE and *ADD MODULEX records are given below. Fields are designated by the database attribute names to which they correspond. Variable length input fields (indicated by the # character) may occupy up to six lines and must end with the character #. (A null entry, designated by only a '#' on the line, is allowed.) Note that the first two lines are used to indicate column positions and are not part of UPDATE input data.

```
000000000111111111122222222223333333333444444444455555
123456789012345678901234567890123456789012345678901234
*ADD AP
ap
desc#
port
type
develop#
citation#
distrib#
*ADD APX
comp
ver
dateint
support
access#
libdoc#
moddoc#
sample#
source#
test#
*ADD MODULE
module
classes#
desc#
howto#
assocmod#
altmod#
```

```
*ADD MODULEX
comp
prec
altmod#
comment#
```

Chapter 4

Format of Unloaded GAMS Database

The GAMS database is maintained using the RIM relational database management system [6]. One way to extract data from a RIM database is to *unload* it. An unloaded database is a simple text file which contains all the data in a format suitable for automatic reloading. (This provides a facility for moving RIM databases among machines of different types.) The purpose of this part is to briefly describe the format of an unloaded GAMS database file. This may be of interest to those who wish to obtain GAMS data but do not have access to the RIM system.

4.1 The Unloaded GAMS Database

A relational database is a collection of tables; the rows of the table correspond to data records and the columns correspond to fields. In a relational system the tables are called *relations* and the fields are called *attributes*. The database *schema* is a description of all the relations and their attributes.

The unloaded GAMS database file contains two parts: a schema definition followed by the raw data. The raw data is composed of the contents of each row of each relation. The following template indicates the overall organization of this file. (Everything up to the NOCHECK statement is the schema.

```
*(SET SEMI=NULL)
*(SET DOLLAR=NULL)
DEFINE GAMS
OWNER "MODIFY "
ATTRIBUTES
list of attributes
RELATIONS
list of relations
```

```
RULES
PASSWORDS
list of passwords
END
NOCHECK
For each relation:
   LOAD name
   rows of the named relation
   END
*(SET SEMI=;)
*(SET DOLLAR=$)
```

This file is in a format suitable for reading and processing by the RIM system. The text in uppercase are actually RIM commands. The text for each line begins in column two (except the asterisks, which are in column one), and no line contains more than 80 characters. A line whose last non-blank character is a plus sign (+) is continued on the next line. Continuation lines may have characters in column one.

The RELATIONS section of the schema indicates which attributes may be found in each relation. Each relation definition starts on a new line and may be continued on one or more lines. The format of each entry is, roughly,

relation WITH *list*

where *relation* is the relation name and *list* is a list of attribute names. Each item is separated by one or more blanks. Both relation and attribute names are at most eight characters in length.

The ATTRIBUTES section of the schema gives the properties of each attribute. These are given in a fixed format, with each attribute definition occupying one physical line. For those not using RIM only three items are of interest: name (columns 2-9, left justified), type (columns 16-19 left justified), and size (columns 29-31, right justified). Type can be one of INT (integer) or TEXT. For attributes of type TEXT, size is either an integer indicating the number of characters for a fixed-length attribute or the character string VAR indicating that the attribute is of variable length. Attributes of type INT always have size 1. The following relations are of the most interest to those outside of NIST without access to RIM:

- MOD\N : the GAMS Classification Scheme
- AP : data about packages
- MODULE : data about modules (programs, subprograms, etc.)

Data for each relation follows the schema. Relations are loaded in the same order as the relation definitions in the RELATIONS section of the schema. Each row of data begins on a new line and may be continued on one or more lines. Within a row,

data items are separated by one or more blanks and appear in the same order that they are specified in the RELATIONS section of the schema. Attributes of type TEXT are enclosed in double-quotes (trailing blanks are truncated on fixed-length TEXT attributes). Consecutive double-quotes ("") are used to represent a double-quote character within a text field. Attributes of type INT occupy 10 character positions (right justified). The character string -0- indicates that no value has been specified for the attribute.

As an example, consider the relation MODULE. In the schema it is defined as

```
MODULE   WITH MOD#     MODULE   AP       CLASSES      +
              DESC     HOWTO    ASSOCMOD AP#          +
              SUBLIB#  SUBLIB
```

with the corresponding attribute definitions

```
MOD#          INT            1   KEY
MODULE        TEXT          12   KEY
AP            TEXT          12   KEY
CLASSES       TEXT         VAR        FORMAT 20
DESC          TEXT         VAR        FORMAT 20
HOWTO         TEXT         VAR        FORMAT 20
ASSOCMOD      TEXT         VAR        FORMAT 20
AP#           INT            1   KEY  FORMAT  4
SUBLIB#       INT            1   KEY  FORMAT  8
SUBLIB        TEXT          12   KEY
```

The following is a valid entry for a row of the MODULE relation in the unloaded database file.

```
        16 "CCBRT" "CMLIB" "C2/" "Complex cube root of com+
plex argument." "C=CCBRT(Z)" -0-          1          9 "FNL+
IB"
```

From this one finds the following value for each attribute

MOD#	16
MODULE	CCBRT
AP	CMLIB
CLASSES	C2/
DESC	Complex cube root of complex argument.
HOWTO	C=CCBRT(Z)
ASSOCMOD	-0-
AP#	1
SUBLIB#	9
SUBLIB	FNLIB

4.2 Sample Unloaded Database File

```
*(SET SEMI=NULL)
*(SET DOLLAR=NULL)
 DEFINE GAMS
 OWNER "MODIFY  "
 ATTRIBUTES
 NODE#          INT            1   KEY
 NODE           TEXT          12   KEY
 TEXT           TEXT         VAR        FORMAT 20
 VERBOSE        TEXT         VAR        FORMAT 20
 SUP            INT            1   KEY  FORMAT  4
 INF            INT            1   KEY  FORMAT  6
 COMP#          INT            1   KEY  FORMAT  4
 COMP           TEXT           6   KEY
 LOCATION       TEXT         VAR        FORMAT 20
 CONTACT        TEXT         VAR        FORMAT 20
 OPSYS          TEXT         VAR        FORMAT 20
 COMPILER       TEXT         VAR        FORMAT 20
 AP#            INT            1   KEY  FORMAT  4
 AP             TEXT          12   KEY
 TYPE           INT            1        FORMAT  4
 PORT           TEXT           1        FORMAT  4
 DESC           TEXT         VAR        FORMAT 20
 LANG           TEXT          12
 DEVELOP        TEXT         VAR        FORMAT 20
 CITATION       TEXT         VAR        FORMAT 20
 DISTRIB        TEXT         VAR        FORMAT 20
 SUPP           TEXT           1        FORMAT  4
 ACCESS         TEXT         VAR        FORMAT 20
 VER#           INT            1        FORMAT  4
 VER            TEXT           6
 LIBDOC         TEXT         VAR        FORMAT 20
 MODDOC         TEXT         VAR        FORMAT 20
 SAMPLE         TEXT         VAR        FORMAT 20
 SOURCE         TEXT         VAR        FORMAT 20
 TESTS          TEXT         VAR        FORMAT 20
 DATEINT        INT            1
 DATEDEL        INT            1
 SUBLIB#        INT            1   KEY  FORMAT  8
 SUBLIB         TEXT          12   KEY
```

```
SUBLIBDC        TEXT          VAR         FORMAT 20
MOD#            INT             1   KEY
MODULE          TEXT           12   KEY
CLASSES         TEXT          VAR         FORMAT 20
HOWTO           TEXT          VAR         FORMAT 20
ASSOCMOD        TEXT          VAR         FORMAT 20
PREC            TEXT            1         FORMAT  4
ALTMOD          TEXT          VAR         FORMAT 20
COMMENT         TEXT          VAR         FORMAT 20
ATTNAM          TEXT            8
ATTDESC         TEXT          VAR         FORMAT 20
ATTFORM         TEXT          VAR         FORMAT 20
ATTEX           TEXT          VAR         FORMAT 20
RELATIONS
MOD\N     WITH NODE#     NODE      TEXT      VERBOSE
MOD\T     WITH SUP       INF
COMPUTER  WITH COMP#     COMP      LOCATION  CONTACT      +
               OPSYS     COMPILER
AP        WITH AP#       AP        TYPE      PORT         +
               DESC      LANG      DEVELOP   CITATION     +
               DISTRIB
APX       WITH AP#       AP        COMP#     COMP         +
               SUPP      ACCESS    VER#      VER          +
               LIBDOC    MODDOC    CITATION  SAMPLE       +
               SOURCE    TESTS
APHIST    WITH AP#       AP        VER#      VER          +
               COMP#     COMP      DATEINT   DATEDEL      +
               CITATION
SUBLIB    WITH SUBLIB#   SUBLIB    AP#       AP           +
               PORT      DESC      DEVELOP   DISTRIB
SUBLIBX   WITH SUBLIB#   SUBLIB    COMP#     COMP         +
               SUPP      VER#      VER       SUBLIBDC     +
               CITATION
MODULE    WITH MOD#      MODULE    AP        CLASSES      +
               DESC      HOWTO     ASSOCMOD  AP#          +
               SUBLIB#   SUBLIB
MODULEX   WITH MOD#      MODULE    COMP#     COMP         +
               PREC      ALTMOD    COMMENT   AP#          +
               AP
DEAD\N    WITH NODE#     NODE
ATTDEF    WITH ATTNAM    ATTDESC   ATTFORM   ATTEX
RULES
```

```
 PASSWORDS
 MPW FOR MOD\N     IS "NONE    "
 RPW FOR MOD\N     IS "NONE    "
 MPW FOR MOD\T     IS "NONE    "
 RPW FOR MOD\T     IS "NONE    "
 MPW FOR COMPUTER IS "NONE    "
 RPW FOR COMPUTER IS "NONE    "
 MPW FOR AP        IS "NONE    "
 RPW FOR AP        IS "NONE    "
 MPW FOR APX       IS "NONE    "
 RPW FOR APX       IS "NONE    "
 MPW FOR APHIST    IS "NONE    "
 RPW FOR APHIST    IS "NONE    "
 MPW FOR SUBLIB    IS "NONE    "
 RPW FOR SUBLIB    IS "NONE    "
 MPW FOR SUBLIBX   IS "NONE    "
 RPW FOR SUBLIBX   IS "NONE    "
 MPW FOR MODULE    IS "NONE    "
 RPW FOR MODULE    IS "NONE    "
 MPW FOR MODULEX   IS "NONE    "
 RPW FOR MODULEX   IS "NONE    "
 MPW FOR DEAD\N    IS "NONE    "
 RPW FOR DEAD\N    IS "NONE    "
 MPW FOR ATTDEF    IS "NONE    "
 RPW FOR ATTDEF    IS "NONE    "
 END
 NOCHECK
 LOAD MOD\N
          1 "GAMS" "Classification Scheme" -0-
          2 "A" "Arithmetic, error analysis" -0-
          3 "A1" "Integer" "Integer arithmetic"
          4 "A2" "Rational" "Rational arithmetic"
          5 "A3" "Real" "Real arithmetic"
          6 "A3a" "Single precision" "Single precision real+
 arithmetic"
          7 "A3b" "Double precision" "Double precision real+
 arithmetic"
          8 "A3c" "Extended precision" "Extended precision +
real arithmetic"
          9 "A3d" "Extended range" "Extended range real ari+
thmetic"
         10 "A4" "Complex" "Complex arithmetic"
```

```
         11 "A4a" "Single precision" "Single precision comp+
lex arithmetic"
         12 "A4b" "Double precision" "Double precision comp+
lex arithmetic"
         13 "A4c" "Extended precision" "Extended precision +
complex arithmetic"
         14 "A4d" "Extended range" "Extended range complex +
arithmetic"
         15 "A5" "Interval" "Interval arithmetic"
        736 "A5a" "Real" "Real interval arithmetic"
        737 "A5b" "Complex" "Complex interval arithmetic"
        732 "A6" "Change of representation" -0-
        733 "A6a" "Type conversion" -0-
        734 "A6b" "Base conversion" -0-
        735 "A6c" "Decomposition, construction" -0-

                              *
                              *
                              *
 END
 LOAD MODULE
          8 "ICEIL" "PORT" "C1/" "Finds the smallest intege+
r greater than or equal to x. Input is real, output is inte+
ger." "I = ICEIL (X)" -0-          5 -0- -0-
         16 "CCBRT" "CMLIB" "C2/" "Complex cube root of com+
plex argument." "C=CCBRT(Z)" -0-          1          9 "FNL+
IB"
         23 "A02ABF" "NAG" "A4b/" "Modulus of a complex num+
ber." "D = A02ABF (XR, XI)" -0-          4 -0- -0-
        490 "C05NCF" "NAG" "F2a/" "Finds a zero of a system+
 of N nonlinear functions in N variables by a modification +
of Powell's hybrid method. Derivatives of the functions are+
 not required. (Comprehensive version of C05NBF.)" "CALL C0+
5NCF (FCN,N,X,FVEC,XTOL,MAXFEV,ML,MU,EPSFCN,DIAG,MODE,FACTO+
R,NPRINT, NFEV FJAC,LDFJAC,R,LR,QTF,W,IFAIL)" -0-          4 +
 -0- -0-
        520 "Q1DAX" "CMLIB" "H2a1a1/" "Flexible subroutine +
for the automatic evaluation of definite integrals of a use+
r-defined function of one variable. Special features includ+
e randomization, singularity weakening, restarting, specifi+
cation of an initial mesh (optional), and output of smalles+
t and largest integrand values." "CALL Q1DAX(F,A,B,EPS,R,E,+
```

```
NINT,RST,W,NMAX,FMIX,FMAX,KF,IFLAG)" -0-          1         49 +
 "Q1DA"
         32 "BFQAD" "CMLIB" "H2a2a1/E3/K6/" "Integrates fun+
ction times derivative of B-spline from X1 to X2. The B-spl+
ine is in ""B"" representation." "CALL BFQAD(F,T,BCOEF,N,K,+
ID,X1,X2,TOL,QUAD,IERR,WORK)" -0-          1          3 "BS+
PLINE"

                              *
                              *
                              *
 END
*(SET SEMI=;)
*(SET DOLLAR=$)
```

References

[1] R. F. Boisvert, S. E. Howe, and D. K. Kahaner. GAMS—a framework for the management of scientific software. *ACM Trans. Math. Softw.*, 11:313–355, 1985.

[2] R. F. Boisvert, S. E. Howe, and D. K. Kahaner. *Guide to Available Mathematical Software.* NBSIR 84-2824, National Bureau of Standards, 1984. Available as PB 84-171305 from the National Technical Information Service (NTIS), Springfield, VA 22161.

[3] D. D. Chamberlin. Relational database management systems. *ACM Comp. Surveys*, 8:43–66, 1976.

[4] L. Lamport. LaTeX: *A Document Preparation System.* Addison-Wesley, Reading, MA, 1986.

[5] *BCS-RIM—Relational Information Management System User Guide.* Boeing Computer Services Co., 7980 Gallows Ct., Vienna, VA, 1985.

[6] *RTIRIM User's Guide and Reference Manual.* RIM Technology Inc., 11661 SE First St., Bellevue, WA, 1985.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** (See instructions)	1. PUBLICATION OR REPORT NO. NIST 89-4042	2. Performing Organ. Report No.	3. Publication Date MARCH 1989

4. TITLE AND SUBTITLE

Internal Structure of the Guide to Available Mathematical Software

5. AUTHOR(S)

Ronald F. Boisvert, Sally E. Howe and Jeanne L. Springmann

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The purpose of the NIST Guide to Available Mathematical Software (GAMS) project is to provide convenient documentation tools for users and maintainers of scientific computer software. The main components of this effort are a detailed tree-structured, problem-oriented classification scheme for mathematical and statistical software, a printed catalog based upon this classification scheme which integrates information about all available software, an on-line interactive version of this catalog, and a relational database containing all information upon which the on-line and off-line catalogs rely, along with associated maintenance programs. This report presents a detailed specification of the internal structure of the GAMS database and the programs used to manipulate it. This information is useful to those who wish to implement the GAMS systems on their own computer systems.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

documentation system; mathematical software; on-line documentation; relational database; software catalog; statistical software

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

55

15. Price

$14.95