\mathbf{GAMS} — A User's Guide

Tutorial by Richard E. Rosenthal

@techpower

@techpower

@techpower

Table of Contents

1	Intro	oduction	1
	1.1	Motivation	1
	1.2	Basic Features of GAMS	1
		1.2.1 General Principles	1
		1.2.2 Documentation	2
		123 Portability	$\frac{1}{2}$
		124 User Interface	$\frac{1}{2}$
		125 Model Library	$\frac{2}{2}$
	13	Organization of the Book	2
	1.0		5
2	A G.	AMS Tutorial by Richard E. Rosenthal	5
	2.1	Introduction	5
	2.2	Structure of a GAMS Model	7
	2.3	Sets	8
	2.4	Data	9
		2.4.1 Data Entry by Lists	0
		2.4.2 Data Entry by Tables	1
		2.4.3 Data Entry by Direct Assignment	1
	2.5	Variables	2
	2.6	Equations	2
		2.6.1 Equation Declaration	3
		2.6.2 GAMS Summation (and Product) Notation	3
		2.6.3 Equation Definition	4
	2.7	Objective Function	5
	2.1	Model and Solve Statements	5
	2.0	Display Statements	6 6
	2.9	The 'le l up m' Detebase	0 6
	2.10	2.10.1 Assignment of Variable Bounds and (on Initial Values	0 6
		2.10.1 Assignment of variable bounds and/or initial values	0
	0.11	2.10.2 Transformation and Display of Optimal values	(
	2.11	GAMS Output	8
		2.11.1 Echo Prints	8
		2.11.2 Error Messages	9
		2.11.3 Reference Maps	1
		2.11.4 Equation Listings $\ldots \ldots \ldots$	2
		2.11.5 Model Statistics	3
		2.11.6 Status Reports	3
		2.11.7 Solution Reports	4
	2.12	Summary	5
3	GAN	AS Programs 2	7
-	3.1	Introduction	7
	3.2	The Structure of GAMS Programs	.7
	0.2	3.2.1 Format of GAMS Input	7
		3.2.2 Classification of GAMS Statements	8
			J

		3.2.3	Organization of GAMS Programs	. 28
	3.3	Data T	vpes and Definitions	. 29
	34	Langua	age Items	29
	0.1	3/1	Characters	. 20
		9.4.9	Deserved Words	. 50
		0.4.2		. 30
		3.4.3	Identifiers	. 30
		3.4.4	Labels	. 31
		3.4.5	Text	. 31
		3.4.6	Numbers	. 32
		3.4.7	Delimiters	. 32
		3.4.8	Comments	. 32
	3 5	Summa	۲۷	33
	0.0	Summ		
Λ	Set	Definiti	ons	35
4	4 1	Introdu	ution	
	4.1	introdu		. 55
	4.2	Simple	Sets	. 35
		4.2.1	The Syntax	. 35
		4.2.2	Set Names	. 35
		4.2.3	Set Elements	. 36
		4.2.4	Associated Text	. 36
		4.2.5	Sequences as Set Elements	. 37
		426	Declarations for Multiple Sets	37
	12	Tho Al	ise Statement: Multiple Names for a Sat	
	4.0	Cubaat	and Demain Checking	. 50
	4.4	Subset		. 30
	4.5	Multi-o	limensional Sets	. 39
		4.5.1	One-to-one Mapping	. 39
		4.5.2	Many-to-many Mapping	. 40
	4.6	Summa		41
		Summe	xi j · · · · · · · · · · · · · · · · · ·	. 41
	1.0	Samma		. 41
5	Dat	a Entry	: Parameters, Scalars & Tables	. 41 43
5	Dat 5.1	a Entry Introdu	: Parameters, Scalars & Tables	• 41 43 • 43
5	Dat 5.1 5.2	a Entry Introdu Scalars	: Parameters, Scalars & Tables	. 41 43 . 43 . 43
5	Dat 5.1 5.2	a Entry Introdu Scalars 5.2.1	Parameters, Scalars & Tables	. 41 43 . 43 . 43 . 43
5	Dat 5.1 5.2	a Entry Introdu Scalars 5.2.1 5.2.2	: Parameters, Scalars & Tables action	. 41 43 . 43 . 43 . 43 . 43 . 44
5	Dat 5.1 5.2	a Entry Introdu Scalars 5.2.1 5.2.2 Param	: Parameters, Scalars & Tables action	. 41 43 . 43 . 43 . 43 . 43 . 44 . 44
5	Dat 5.1 5.2 5.3	a Entry Introdu Scalars 5.2.1 5.2.2 Parame 5.2.1	: Parameters, Scalars & Tables iction	. 41 43 . 43 . 43 . 43 . 43 . 44 . 44
5	Dat 5.1 5.2 5.3	a Entry Introdu Scalars 5.2.1 5.2.2 Param 5.3.1	: Parameters, Scalars & Tables Iction	. 41 43 . 43 . 43 . 43 . 43 . 44 . 44 . 44
5	Dat 5.1 5.2 5.3	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2	: Parameters, Scalars & Tables Iction	 41 43 43 43 43 44 44 44 44
5	Dat 5.1 5.2 5.3	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3	: Parameters, Scalars & Tables action	. 41 43 . 43 . 43 . 43 . 44 . 44 . 44 . 44
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Parama 5.3.1 5.3.2 5.3.3 Tables	: Parameters, Scalars & Tables action	 41 43 43 43 43 44 44 44 44 45 45
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Parama 5.3.1 5.3.2 5.3.3 Tables 5.4.1	: Parameters, Scalars & Tables action	 41 43 43 43 43 44 44 44 44 45 45 45
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Parama 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2	: Parameters, Scalars & Tables action	 41 43 43 43 43 44 44 44 44 45 45 45 45 46
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Parama 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3	: Parameters, Scalars & Tables action	$\begin{array}{cccccccccccccccccccccccccccccccccccc$
5	Dat . 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Parame 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4	: Parameters, Scalars & Tables action	$\begin{array}{c} & 41 \\ & 43 \\ & 43 \\ & 43 \\ & 43 \\ & 44 \\ & 44 \\ & 44 \\ & 44 \\ & 44 \\ & 45 \\ & 45 \\ & 45 \\ & 45 \\ & 45 \\ & 46 \\ & 46 \\ & 47 \end{array}$
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5	: Parameters, Scalars & Tables action The Syntax An Illustrative Example eters The Syntax An Illustrative Examples Parameter Data for Higher Dimensions The Syntax An Illustrative Example Continued Tables Tables with more than Two Dimensions Condensing Tables	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 46\\ & 47\\ \end{array}$
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6	: Parameters, Scalars & Tables nction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 46\\ & 47\\ & 47\\ & 47\end{array}$
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6	: Parameters, Scalars & Tables nction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 46\\ & 47\\ & 47\\ & 48\\ \end{array}$
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony	: Parameters, Scalars & Tables Inction The Syntax An Illustrative Example eters The Syntax An Illustrative Examples Parameter Data for Higher Dimensions The Syntax An Illustrative Example Continued Tables Tables with more than Two Dimensions Condensing Tables Handling Long Row Labels	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\end{array}$
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1	: Parameters, Scalars & Tables Inction The Syntax An Illustrative Example eters The Syntax An Illustrative Examples Parameter Data for Higher Dimensions The Syntax An Illustrative Example Continued Tables Tables with more than Two Dimensions Condensing Tables Handling Long Row Labels ms The Syntax	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ \end{array}$
5	Dat 5.1 5.2 5.3 5.4	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2	: Parameters, Scalars & Tables Iction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ \end{array}$
5	Dat 5.1 5.2 5.3 5.4 5.5 5.6	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa	: Parameters, Scalars & Tables Iction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 49\end{array}$
5	Dat . 5.1 5.2 5.3 5.4 5.5 5.6	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa	: Parameters, Scalars & Tables Iction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 47\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 49\end{array}$
5	Dat . 5.1 5.2 5.3 5.4 5.5 5.6 Dat .	a Entry Introdu Scalars 5.2.1 5.2.2 Param 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa a Manig	: Parameters, Scalars & Tables netion	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 445\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 51\end{array}$
5	Dat 5.1 5.2 5.3 5.4 5.5 5.6 Dat 6.1	a Entry Introdu Scalars 5.2.1 5.2.2 Param 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa a Manig Introdu	: Parameters, Scalars & Tables Iction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 51\\ & 51\end{array}$
5	Dat 5.1 5.2 5.3 5.4 5.5 5.6 Dat 6.1 6.2	a Entry Introdu Scalars 5.2.1 5.2.2 Parama 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa a Manig Introdu	: Parameters, Scalars & Tables Inction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 47\\ & 48\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 51\\ & 51\\ & 51\\ \end{array}$
5	Dat 5.1 5.2 5.3 5.4 5.5 5.6 Dat 6.1 6.2	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa a Manij Introdu The As 6.2.1	: Parameters, Scalars & Tables nction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 45\\ & 46\\ & 47\\ & 46\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 51\\ & 51\\ & 51\\ & 51\\ \end{array}$
5	Dat 5.1 5.2 5.3 5.4 5.5 5.6 Dat 6.1 6.2	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa a Manig Introdu The As 6.2.1 6.2.2	: Parameters, Scalars & Tables nction	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 47\\ & 46\\ & 46\\ & 47\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 51\\ & 51\\ & 51\\ & 51\\ & 51\\ \end{array}$
5	Dat. 5.1 5.2 5.3 5.4 5.5 5.6 Dat. 6.1 6.2	a Entry Introdu Scalars 5.2.1 5.2.2 Paramo 5.3.1 5.3.2 5.3.3 Tables 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 5.4.6 Acrony 5.5.1 5.5.2 Summa a Manig Introdu The As 6.2.1 6.2.2 6.2.3	: Parameters, Scalars & Tables Inction The Syntax An Illustrative Example eters The Syntax An Illustrative Examples Parameter Data for Higher Dimensions The Syntax An Illustrative Example Continued Tables Tables with more than Two Dimensions Condensing Tables Handling Long Row Labels ms The Syntax Illustrative Example ary collations with Parameters Inction Scalar Assignments Lindexed Assignments Lindexed Assignments Lindexed Example	$\begin{array}{c} & 41\\ & 43\\ & 43\\ & 43\\ & 43\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 44\\ & 45\\ & 45\\ & 45\\ & 45\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 46\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 48\\ & 51\\ & 51\\ & 51\\ & 51\\ & 51\\ & 51\\ & 51\\ \end{array}$

		6.2.4	Assignments Over Subsets	. 52
		6.2.5	Issues with Controlling Indices	. 52
		6.2.6	Extended Range Identifiers in Assignments	. 53
		6.2.7	Acronyms in Assignments	. 53
	6.3	Expres	ssions	. 53
		6.3.1	Standard Arithmetic Operations	. 53
		6.3.2	Indexed Operations	. 54
		633	Functions	55
		634	Extended Bange Arithmetic and Error Handling	. 00 63
	64	Summe	arv	. 00 64
	0.1	Summe	ary	. 01
7	Vari	ables		65
	7.1	Introdu	uction	. 65
	7.2	Variab	le Declarations	. 65
		7.2.1	The Syntax	. 65
		722	Variable Types	. 00 66
		723	Styles for Variable Declaration	. 00 66
	73	Variah	Je Attributes	. 00 67
	1.0	731	Bounds on Variables	. 01 67
		729	Eving Variables	. 01 67
		722	Activity Levels of Variables	. 01 68
	74	Variah	Activity Levels of Variables	. 00 60
	1.4	variab.	Assigning Values to Variable Attributes	. 00 60
		(.4.1	Assigning values to variable Attributes	. 68
		(.4.2	Variable Attributes in Assignments	. 68
		(.4.3 C	Displaying variable Attributes	. 69
	7.5	Summa	ary	. 69
8	Equ	ations		71
0	8 1	Introdu	uction	71
	8.2	Equati	ion Declarations	· 11 71
	0.2	8 2 1	The Syntax	· 11 71
		822	An Illustrative Exemple	· 11 71
	09	Equati	In must any e mainple	· 11 79
	0.0	Q 2 1		· 12 79
		0.0.1	An Illustrative Exemple	· 14
		8.3.2		. 12
		8.3.3	Scalar Equations	. (3
		8.3.4		. (3
	0.4	8.3.5	Using Labels Explicitly in Equations	. 73
	8.4	Expres	ssions in Equation Definitions	. 74
		8.4.1	Arithmetic Operators in Equation Definitions	. 74
		8.4.2	Functions in Equation Definitions	. 74
	~ -	8.4.3	Preventing Undefined Operations in Equations	. 74
	8.5	Data E	Handling Aspects of Equations	. 75
0	Mod	lol and	Solvo Statomonte	77
9	0.1	Introdu		77
	0.2	The M	[ode] Statement	. 11 77
	9.4	0.9.1		· // 77
		9.2.1	Classification of Modela	. 11 70
		9.4.4	Viassinication of Models	. 18
	0.9	9.2.3 The O	Model Attributes	. 19
	9.3	ine Sc		. 83
		9.3.1		. 83
		9.3.2	Requirements for a Valid Solve Statement	. 84
	o . t	9.3.3	Actions Triggered by the Solve Statement	. 84
	9.4	Progra	ams with Several Solve Statements	. 85

		9.4.2	Sensitivity or Scenario Analysis	85
		9.4.3	Iterative Implementation of Non-Standard Algorithms	86
	9.5	Making	g New Solvers Available with GAMS	87
10	GAN	AS Out	put 8	39
	10.1	Introdu	iction	39
	10.2	The Ill	ustrative Model	39
	10.3	Compil	ation Output	90
		10.3.1	Echo Print of the Input File	90
		10.3.2	The Symbol Reference Map	91
		10.3.3	The Symbol Listing Map	93
		10.3.4	The Unique Element Listing - Map	93
		10.3.5	Useful Dollar Control Directives	93
	10.4	Execut	ion Output	94
	10.5	Output	Produced by a Solve Statement	95
		10.5.1	The Equation Listing	95
		10.5.2	The Column Listing	96
		10.5.3	The Model Statistics	96
		10.5.4	The Solve Summary	97
		10 5 5	Solver Report)0
		10.5.6	The Solution Listing)1
		10.5.7	Report Summary)9)9
		10.5.1	File Summary	ງ2 ງງ
	10.6	Frror F	Price Summary	ງ2 ງງ
	10.0	10.6.1	Compilation France 10	ว∠ าว
		10.0.1	Complication Errors 10	ງວ Դ4
		10.0.2)4)5
		10.0.3		ງວ ວະ
	10.7	10.0.4		ງວ ງຂ
	10.7	Summa	ury	90
11	Con	ditional		
			Exprogetone Aggreginments and Eguations	17
		Introdu	Expressions, Assignments and Equations)7)7
	11.1 11.9	Introdu	Expressions, Assignments and Equations 10 action)7)7)7
	11.1 11.2	Introdu Logical	Expressions, Assignments and Equations 10 action 10 Conditions 10 Numerical Expressions as Logical Conditions 10)7)7)7
	11.1 11.2	Introdu Logical 11.2.1	Expressions, Assignments and Equations 10 inction 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Expressions as Logical Conditions 10)7)7)7)7
	11.1	Introdu Logical 11.2.1 11.2.2	Expressions, Assignments and Equations 10 action 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Learning Operators 10)7)7)7)7)7
	11.1 11.2	Introdu Logical 11.2.1 11.2.2 11.2.3	Expressions, Assignments and Equations 10 action 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Operators 10	07 07 07 07 07
	11.1 11.2	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4	Expressions, Assignments and Equations 10 action 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions 10 Set Membership 10 Logical Conditions 10 Logical Operators 10 Logical Operato)7)7)7)7)7)7)8)8
	11.1 11.2	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5	Expressions, Assignments and Equations 10 inction 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Conditions Involving Acronyms 10)7)7)7)7)7)7)8)8)9
	11.1	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6	Expressions, Assignments and Equations 10 inction 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10)7)7)7)7)7)7)7)7)7)7)7)7)7)7
	11.1 11.2	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7	Expressions, Assignments and Equations 10 inction 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10)7)7)7)7)7)7)7)7)7)7)7)7)7)7
	11.1 11.2	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8	Expressions, Assignments and Equations 10 iction 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11)7)7)7)7)7)7)7)7)7)7)7)7)7)7
	11.1 11.2 11.3	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do	Expressions, Assignments and Equations 10 iction 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 Intervention 11)7)7)7)7)7)7)7)7)7)7)7
	11.1 11.2 11.3	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1	Expressions, Assignments and Equations 10 inction 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 An Example 11	\mathbf{b}_{10}
	11.1 11.2 11.3	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2	Expressions, Assignments and Equations10inction10Conditions10Numerical Expressions as Logical Conditions10Numerical Relationship Operators10Logical Operators10Set Membership10Logical Conditions Involving Acronyms10Numerical Values of Logical Conditions10Mixed Logical Conditions - Operator Precedence10Mixed Logical Conditions - Examples11Jlar Condition12Nested Dollar Conditions13	07 07 07 07 07 07 07 07 07 08 09 09 09 09 09 09 10 10
	11.1 11.2 11.3 11.4	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit	Expressions, Assignments and EquationsICaction10Conditions10Numerical Expressions as Logical Conditions10Numerical Relationship Operators10Logical Operators10Set Membership10Logical Conditions Involving Acronyms10Numerical Values of Logical Conditions10Mixed Logical Conditions - Operator Precedence10Mixed Logical Conditions - Examples11Dilar Condition12Nested Dollar Conditions13Interval Assignments14	$\mathbf{b}7$ \mathbf{b}
	11.1 11.2 11.3 11.4	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1	Expressions, Assignments and Equations 10 action 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 An Example 11 Nested Dollar Conditions 11 Dollar on the Left 11	$\mathbf{b}7$ \mathbf
	11.1 11.2 11.3 11.4	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1 11.4.2	Expressions, Assignments and EquationsICaction10Conditions10Numerical Expressions as Logical Conditions10Numerical Relationship Operators10Logical Operators10Set Membership10Logical Conditions Involving Acronyms10Numerical Values of Logical Conditions10Mixed Logical Conditions - Operator Precedence10Mixed Logical Conditions - Examples11Jalar Condition12Nested Dollar Conditions12Dollar on the Left13Dollar on the Right14	$\mathbf{b}7$ \mathbf
	11.1 11.2 11.3 11.4	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1 11.4.2 11.4.3	Expressions, Assignments and EquationsICaction10Conditions10Numerical Expressions as Logical Conditions10Numerical Relationship Operators10Logical Operators10Set Membership10Logical Conditions Involving Acronyms10Numerical Values of Logical Conditions10Mixed Logical Conditions - Operator Precedence10Mixed Logical Conditions - Examples11Illar Condition11Nested Dollar Conditions11Ional Assignments11Dollar on the Left11Dollar on the Right11Filtering Controlling Indices in Indexed Operations11	$\mathbf{b}7$ \mathbf{b}
	11.1 11.2 11.3 11.4	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1 11.4.2 11.4.3 11.4.4	Expressions, Assignments and Equations 10 action 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 Dilar Condition 12 Nested Dollar Conditions 11 Dollar on the Left 11 Dollar on the Right 12 Filtering Controlling Indices in Indexed Operations 13 Filtering Sets in Assignments 14	$\mathbf{b}7$ \mathbf{b}
	11.1 11.2 11.3 11.4	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The De 11.3.1 11.3.2 Condit 11.4.1 11.4.2 11.4.3 11.4.4 Condit	Expressions, Assignments and Equations 10 nction 11 Conditions 11 Numerical Expressions as Logical Conditions 11 Numerical Relationship Operators 11 Logical Operators 11 Logical Operators 11 Logical Conditions Involving Acronyms 11 Numerical Values of Logical Conditions 11 Numerical Values of Logical Conditions 11 Mixed Logical Conditions - Operator Precedence 11 Mixed Logical Conditions - Examples 11 Dilar Condition 12 Nested Dollar Conditions 11 Nested Dollar Conditions 11 Dollar on the Left 12 Dollar on the Right 13 Filtering Controlling Indices in Indexed Operations 13 Filtering Sets in Assignments 13 Ional Indexed Operations 14	\mathbf{b}_{11}^{0}
	11.1 11.2 11.3 11.4	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1 11.4.2 11.4.3 11.4.4 Condit 11.5.1	Expressions, Assignments and Equations 10 nction 11 Conditions 11 Numerical Expressions as Logical Conditions 11 Numerical Relationship Operators 11 Logical Operators 11 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Numerical Values of Logical Conditions 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 Dilar Condition 12 An Example 13 Nested Dollar Conditions 14 Dollar on the Left 15 Dollar on the Right 14 Filtering Controlling Indices in Indexed Operations 15	$\mathbf{b}7$ \mathbf{b}
	11.1 11.2 11.3 11.4 11.5 11.6	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1 11.4.2 11.4.3 11.4.4 Condit 11.5.1 Condit	Expressions, Assignments and Equations 10 iction 10 Conditions 10 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 An Example 11 Nested Dollar Conditions 11 Dollar on the Left 11 Dollar on the Right 11 Filtering Controlling Indices in Indexed Operations 11 Filtering Cont	$\mathbf{b}7$ \mathbf{b}
	11.1 11.2 11.3 11.4 11.5 11.6	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1 11.4.2 11.4.3 11.4.4 Condit 11.5.1 Condit 11.6.1	Expressions, Assignments and Equations 10 inction 11 Conditions 11 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Logical Operators 10 Set Membership 10 Logical Conditions Involving Acronyms 10 Numerical Values of Logical Conditions 10 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 An Example 11 Nested Dollar Conditions 11 Dollar on the Left 11 Dollar on the Right 11 Filtering Controlling Indices in Indexed Operations 11 Filtering Controlling Indices in Indexed Operations 11 Filtering Controlling Indices in Indexed Operations 12 Filtering Controlling Indices in Indexed Operations 13 Filtering Controlling Indices in Indexed Operations 14 Filtering Controlling Indices in Indexed Operations 15 Ional Indexed Operations 15 Ional Equations 15	$\mathbf{b7}$
	11.1 11.2 11.3 11.4 11.5 11.6	Introdu Logical 11.2.1 11.2.2 11.2.3 11.2.4 11.2.5 11.2.6 11.2.7 11.2.8 The Do 11.3.1 11.3.2 Condit 11.4.1 11.4.2 11.4.3 11.4.4 Condit 11.5.1 Condit 11.6.1 11.6.2	Expressions, Assignments and Equations 10 nction 11 Conditions 11 Numerical Expressions as Logical Conditions 10 Numerical Relationship Operators 10 Logical Operators 10 Set Membership 11 Logical Conditions Involving Acronyms 11 Numerical Values of Logical Conditions 11 Numerical Values of Logical Conditions 11 Mixed Logical Conditions - Operator Precedence 10 Mixed Logical Conditions - Examples 11 Dilar Condition 12 Nested Dollar Conditions 11 Nested Dollar Conditions 11 Dollar on the Left 12 Dollar on the Right 13 Filtering Controlling Indices in Indexed Operations 14 Filtering Controlling Indices in Indexed Operations 15 Filtering Controlling Indices in Indexed Operations 14 Intering Controlling Indices in Indexed Operations 15 Filtering Controlling Indices in Indexed Operations 15 Foldar Operators within the Algebra 15 Dollar Operators within the Algebra 17	$\mathbf{b7}$

12 Dyn	amic Sets 117
12.1	Introduction
12.2	Assigning Membership to Dynamic Sets
	12.2.1 The Syntax
	12.2.2 Illustrative Example
	12.2.3 Dynamic Sets with Multiple Indices
	12.2.4 Assignments over the Domain of Dynamic Sets
	12.2.5 Equations Defined over the Domain of Dynamic Sets
12.3	Using Dollar Controls with Dynamic Sets
	12.3.1 Assignments
	12.3.2 Indexed Operations
	12.3.3 Equations
	12.3.4 Filtering through Dynamic Sets
12.4	Set Operations
	12.4.1 Set Union
	12.4.2 Set Intersection
	12.4.3 Set Complement
	12.4.4 Set Difference
12.5	Summary 121
12.0	Sammary
13 Sets	as Sequences: Ordered Sets 123
13.1	Introduction
13.2	Ordered and Unordered Sets
13.3	Ord and Card
	13.3.1 The Ord Operator
	13.3.2 The Card Operator
13.4	Lag and Lead Operators
13.5	Lags and Leads in Assignments
10.0	13.5.1 Linear Lag and Lead Operators - Reference
	13.5.2 Linear Lag and Lead Operators - Assignment 126
	13.5.2 Circular Lag and Lead Operators
13.6	Lags and Leads in Equations
10.0	13.6.1 Linear Lag and Load Operators Domain Control
	13.6.2 Linear Lag and Lead Operators – Domain Control
	13.6.2 Circular Lag and Load Operators
197	
15.7	Summary
14 The	Display Statement
1/1	Introduction 131
14.1	The Syntax
14.2	An Example
14.0	The Label Order in Displays 139
14.4	$\frac{114.1}{1} \text{Example} $
14.5	Display Controls
14.0	14.5.1 Clobal Digplay Controls
	14.5.1 Global Display Control 124
	14.5.2 Local Display Control
	14.5.5 Display Statement to Generate Data in List Format
15 The	Put Writing Facility
15 I He	Introduction 197
15.0	The Symtax 197
10.2	пе бущах
10.0 15 4	An Example
10.4	15/1 Defining Files 199
	15.4.9 Assigning Files
	10.4.2 Assigning r lies

		15.4.4 Appending to a File	140
	15.5	Page Format	140
	15.6	Page Sections	141
		15.6.1 Accessing Various Page Sections	42
		15.6.2 Paging	42
	15.7	Positioning the Cursor on a Page	42
	15.8	System Suffixes	43
	15.9	Output Items	43
		15.9.1 Text Items	44
		15.9.2 Numeric Items	145
		15.9.2 Set Value Items	1/15
	15 10	Clobal Itam Formatting	145
	10.10	15 10 1 Field Justification	145
			140
	18 11	15.10.2 Field Width	140
	10.11		140
	15.12	Additional Numeric Display Control	147
		15.12.1 Illustrative Example	147
	15.13	Cursor Control	148
		15.13.1 Current Cursor Column	148
		15.13.2 Current Cursor Row	149
		15.13.3 Last Line Control	149
	15.14	Paging Control	150
	15.15	Exception Handling	150
	15.16	Source of Errors Associated with the Put Statement	150
		15.16.1 Syntax Errors	150
		15.16.2 Put Errors	150
	15.17	Simple Spreadsheet/Database Application	151
		15.17.1 An Example	151
16	Prog	ramming Flow Control Features 1	53
16	Prog 16.1	ramming Flow Control Features 1 Introduction 1	. 53 153
16	Prog 16.1 16.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1	. 53 153 153
16	Prog 16.1 16.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1	. 53 153 153 153
16	Prog 16.1 16.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1	. 53 153 153 153 153
16	Prog 16.1 16.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1	. 53 153 153 153 154 154
16	Prog 16.1 16.2 16.3	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1	. 53 153 153 153 154 154
16	Prog 16.1 16.2 16.3	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1	. 53 153 153 153 154 154 154
16	Prog 16.1 16.2 16.3	ramming Flow Control Features1Introduction1The Loop Statement116.2.1 The Syntax116.2.2 Examples1The If-Elseif-Else Statement116.3.1 The Syntax116.3.2 Examples1The While Statement1	. 53 153 153 154 154 154 155 155
16	Prog 16.1 16.2 16.3 16.4	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.4 The Syntax 1 16.4.1 The Syntax 1	53 153 153 153 154 154 154 155 156 156
16	Prog 16.1 16.2 16.3 16.4	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.4 The Syntax 1 16.4.1 The Syntax 1 16.4.2 Examples 1	53 153 153 154 154 154 155 156 156 156
16	Prog 16.1 16.2 16.3 16.4	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.3 Examples 1 16.4.4 Examples 1 16.4.5 Examples 1 16.4.6 Examples 1 17 1 18 1 19 1 10 1 11 1 12 1	53 153 153 153 154 154 154 155 156 156 156
16	Prog 16.1 16.2 16.3 16.4 16.5	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 The While Statement 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.5 1 The Syntax 1	53 153 153 153 154 154 155 156 156 156 156
16	Prog 16.1 16.2 16.3 16.4 16.5	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1	53 153 153 154 154 155 156 156 156 156 156
16	Prog 16.1 16.2 16.3 16.4 16.5	ramming Flow Control Features 1. Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.5.1 The Syntax 1 16.5.2 Examples 1	53 153 153 154 154 154 156 156 156 156 156 157 157
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec	ramming Flow Control Features 1. Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.3.4 The Syntax 1 16.3.5 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1	 53 153 153 153 154 154 155 156 156 156 157 59
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1	ramming Flow Control Features 1. Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.3 The Syntax 1 16.5.4 Examples 1 17 1 18 1	.53 153 153 153 154 154 155 156 156 156 157 157 .59 159
16 17	Prog 16.1 16.2 16.3 16.4 16.5 Spec : 17.1 17.2	ramming Flow Control Features1.Introduction1The Loop Statement116.2.1 The Syntax116.2.2 Examples1The If-Elseif-Else Statement116.3.1 The Syntax116.3.2 Examples1The While Statement116.4.1 The Syntax116.4.2 Examples1The For Statement116.5.1 The Syntax116.5.2 Examples1ial Language Features1Introduction1Special MIP Features1	.53 153 153 154 154 155 156 156 156 156 157 157 157 157
16 17	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1 17.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.3.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1 16.5.2 Examples 1 17 Phe For Statement 1 16.5.2 Examples 1 16.5.2 Examples 1 17 Phe For Statement 1 16.5.2 Examples 1 17 Phe For Statement 1 17 Phe For Statement 1 17 Phe For Statement 1 18 Language Features 1 19 Columb Features 1 10 Throduction 1 17 Phe Features 1	.53 153 153 153 154 154 155 156 156 156 156 157 157 157 159 159
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1 17.2	ramming Flow Control Features 1. Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.3.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1 16.5.2 Examples 1 17.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Trupe 1 (SOS1) 1	.53 153 153 154 154 155 156 156 156 156 156 157 157 159 159 159
16 17	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1 17.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.4 The Syntax 1 16.3.5 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.5.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1 16.5.2 Examples 1 17.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Type 1 (SOS1) 1 17.2.3 Special Order Sets of Type 1 (SOS1) 1 17.2.3 Special Order Sets of Type 2 (SOS2) 1	.53 153 153 154 154 154 155 156 156 156 156 157 157 159 159 159 159
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec : 17.1 17.2	ramming Flow Control Features 1. Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 The While Statement 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.2 Examples 1 16.4.2 Examples 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1 16.5.2 Examples 1 17.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Type 1 (SOS1) 1 17.2.3 Special Order Sets of Type 2 (SOS2) 1 17.2.4 SemicContinuous Wariables 1	.53 153 153 154 154 154 156 156 156 156 156 157 .59 159 159 159 159 160 161
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec : 17.1 17.2	ramming Flow Control Features 1. Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.4 The Syntax 1 16.4.5 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 17.4 For Statement 1 16.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 117.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Type 1 (SOS1) 1 17.2.3 Special Order Sets of Type 2 (SOS2) 1 17.2.4 Semi-Continuous Variables 1	53 153 153 153 154 154 155 156 156 156 157 59 159 159 159 159 159 159 159 159
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1 17.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.3.2 Examples 1 16.3.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1 16.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 117.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Type 1 (SOS1) 1 17.2.3 Special Order Sets of Type 2 (SOS2) 1 17.2.4 Semi-Continuous Variables 1 17.2.5 Semi-Integer Variables 1 17.2.6 Semi-Integer Variables 1 17.2.6 Semi-Integer Variables 1 </td <td> 53 153 153 153 154 154 155 156 156 157 159 159 159 159 159 159 160 161 161 </td>	 53 153 153 153 154 154 155 156 156 157 159 159 159 159 159 159 160 161 161
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1 17.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.2 Examples 1 16.3.4 The Syntax 1 16.3.5 Examples 1 16.4.2 Examples 1 16.4.2 Examples 1 16.4.2 Examples 1 16.5.1 The Syntax 1 16.5.2 Examples 1 17.5 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1 16.5.2 Examples 1 17.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Type 1 (SOS1) 1 17.2.3 Special Order Sets of Type 2 (SOS2) 1 17.2.4 Semi-Continuous Variables 1 17.2.5 Semi-Integer Variables 1 17.2.6 Setting Priorities for Branching 1 17.2.6 Setting Priorities for Branching 1	53 153 153 153 154 154 155 156 156 156 157 159 159 159 159 159 159 160 161 162 162
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1 17.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 16.3.1 The Syntax 1 16.3.2 Examples 1 17he While Statement 1 16.4.1 The Syntax 1 16.4.2 Examples 1 16.4.1 The Syntax 1 16.5.2 Examples 1 17he For Statement 1 16.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 117.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Type 1 (SOS1) 1 17.2.2 Special Order Sets of Type 2 (SOS2) 1 17.2.4 Semi-Continuous Variables 1 17.2.5 Semi-Integer Variables 1 17.2.6 Setting Priorities for Branching 1 17.2.6 Setting Priorities fo	 53 153 153 154 154 155 156 156 156 156 157 59 159 159 160 161 162 162 162
16	Prog 16.1 16.2 16.3 16.4 16.5 Spec 17.1 17.2	ramming Flow Control Features 1 Introduction 1 The Loop Statement 1 16.2.1 The Syntax 1 16.2.2 Examples 1 The If-Elseif-Else Statement 1 16.3.1 The Syntax 1 16.3.2 Examples 1 The While Statement 1 16.4.1 The Syntax 1 16.4.2 Examples 1 The For Statement 1 16.4.2 Examples 1 The For Statement 1 16.5.1 The Syntax 1 16.5.2 Examples 1 16.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 116.5.2 Examples 1 117.2.1 Types of Discrete Variables 1 17.2.2 Special Order Sets of Type 1 (SOS1) 1 17.2.3 Special Order Sets of Type 2 (SOS2) 1 17.2.4 Semi-Continuous Variables 1 17.2.5 Semi-Integer Variables 1 17.2.6 Setting Priorities for Branching 1 17.2.6 Viewere Variables	53 153 153 154 154 155 156 156 156 156 157 59 159 159 159 159 159 159 160 161 162 162 162 162

	17.3.3Equation Scaling	
$\mathbf{A}_{]}$	Appendix	166
A	A Glossary	167
В	B The GAMS Model Library	173
С	C The GAMS Call C.1 The Generic "no frills" GAMS Call	175
	C.1.1 Specifying Options through the Command Line	
	C.2 List of Command Line Parameters	
D	D Dollar Control Options	201
	D.1 Introduction	
	D.1.1 Syntax	201
	D.3 Detailed Description of Dollar Control Options	
E	E The Option Statement	241
	E.1 Introduction	
	E.1.1 The Syntax	241
	E.3 Detailed Description of Options	
\mathbf{F}	F The Save and Restart Feature	247
	F.1 Introduction \dots I.D. () E. (
	F.2 The Save and Restart Feature	
	F.2.2 Restarting from the Work File	
	F.3 Ways in which a Work File is Useful	
	F.3.1 Separation of Model and Data	
	F.3.2 Incremental Program Development	
	F.3.4 Multiple Scenarios	
	F.3.5 The GAMS Runtime License	
G	G Secure Work Files	253
	G.1 Introduction	
	G.3 Secure Work Files	
	G.4 Access Control Commands	
	G.5 Advanced Use of Access Control	
	G.6 Limitations and Future Requirements	
н	H Compressed and Encrypted Input Files	259 259
	H.2 A First Example	
	H.3 The CEFILES Gamslib Model	
	H.4 The ENCRYPT GAMSLIB Model	
Ι	I The GAMS Grid Computing Facility	263
	1.1 Introduction 1.1 1.2 Basic Concepts	· · · · · · · · · · · · · · · · · · ·
	I.3 A First Example	
	I.4 Advanced use of Grid Features	

		.4.1 Very Long Job Durations	66
	I.5	Summary of Grid Features	67
		.5.1 Grid Handle Functions	58
		.5.2 Grid Model Attributes	58
		.5.3 Grid Solution Retrieval	<u> 59</u>
		.5.4 Grid Directory	<u> </u>
	I.6	Architecture and Customization	<u> </u>
		.6.1 Grid Submission Testing	70
	I.7	Glossary and Definitions	71
J	Extr	asic Functions 27	73
J	Extr J.1	27 Introduction	7 3 73
J	Extr J.1 J.2	27 27 27 27 27 27 27 27 27 27 27 27 27 2	7 3 73 73
J	Extr J.1 J.2 J.3	sic Functions 27 Introduction 27 Fitpack Library 27 Piecewise Polynomial Library 27	73 73 73 74
J	Extr J.1 J.2 J.3 J.4	sic Functions 27 Introduction 27 Fitpack Library 27 Piecewise Polynomial Library 27 Stochastic Library 27	73 73 73 74 75
J	Extr J.1 J.2 J.3 J.4 J.5	nsic Functions 27 Introduction 27 Fitpack Library 27 Piecewise Polynomial Library 27 Stochastic Library 27 LINDO Sampling Library 27	73 73 73 74 75 77
J	Extr J.1 J.2 J.3 J.4 J.5 J.6	sic Functions27Introduction27Fitpack Library27Piecewise Polynomial Library27Stochastic Library27LINDO Sampling Library27Irigonometric Library28	73 73 73 74 75 77 30

Index

285

List of Tables

2.1 2.2 2.3	Data for the transportation problem (adapted from Dantzig, 1963) 6 The basic components of a GAMS model 8 Permissible variable types 1
3.1 3.2 3.3 3.4	Organization of GAMS programs 29 Legal characters 30 Reserved words and symbols 30 Rules for constructing identifiers and labels 30
4.1	Examples of the compact representation of sets
$ \begin{array}{r} 6.1 \\ 6.2 \\ 6.3 \end{array} $	GAMS functions 6 Special symbols for extended arithmetic 6 Exponentiation and Division 6
7.1	Variable types and default bounds
$8.1 \\ 8.2$	Classification of functions with endogenous arguments
$9.1 \\ 9.2$	Model Attributes mainly used before solve 82 Model Attributes mainly used after solve 83
10.1	List of GAMS data types
$11.1 \\ 11.2 \\ 11.3$	Truth table of logical operators 108 Operator precedence 109 Examples of logical conditions 110
$\begin{array}{c} 14.1 \\ 14.2 \end{array}$	Default values for .lo and .up subtypes
C.1	GAMS command line parameters
D.1	Dollar control options
E.1	GAMS options
J.1 J.2 J.3 J.4	Fitpack functions 273 Piecewise polynomial functions 274 Random number generators 276 Distribution functions 276
J.5 J.6 J.7	Normal distribution functions 277 LINDO sampling functions 278 Trigonometric functions 286

1

Introduction

1.1 Motivation

Substantial progress was made in the 1950s and 1960s with the development of algorithms and computer codes to solve large mathematical programming problems. The number of applications of these tools in the 1970s was less then expected, however, because the solution procedures formed only a small part of the overall modeling effort. A large part of the time required to develop a model involved data preparation and transformation and report preparation. Each model required many hours of analyst and programming time to organize the data and write the programs that would transform the data into the form required by the mathematical programming optimizers. Furthermore, it was difficult to detect and eliminate errors because the programs that performed the data operations were only accessible to the specialist who wrote them and not to the analysts in charge of the project.

GAMS was developed to improve on this situation by:

- > Providing a high-level language for the compact representation of large and complex models
- > Allowing changes to be made in model specifications simply and safely
- > Allowing unambiguous statements of algebraic relationships
- > Permitting model descriptions that are independent of solution algorithms

1.2 Basic Features of GAMS

1.2.1 General Principles

The design of GAMS has incorporated ideas drawn from relational database theory and mathematical programming and has attempted to merge these ideas to suit the needs of strategic modelers. Relational database theory provides a structured framework for developing general data organization and transformation capabilities. Mathematical programming provides a way of describing a problem and a variety of methods for solving it. The following principles were used in designing the system:

- 1. All existing algorithmic methods should be available without changing the user's model representation. Introduction of new methods, or of new implementations of existing methods, should be possible without requiring changes in existing models. Linear, nonlinear, mixed integer, mixed integer nonlinear optimizations and mixed complementarity problems can currently be accommodated.
- 2. The optimization problem should be expressible independently of the data it uses. This separation of logic and data allows a problem to be increased in size without causing an increase in the complexity of the representation.

3. The use of the relational data model requires that the allocation of computer resources be automated. This means that large and complex models can be constructed without the user having to worry about details such as array sizes and scratch storage.

1.2.2 Documentation

The GAMS model representation is in a form that can be easily read by people and by computers. This means that the GAMS program itself is the documentation of the model, and that the separate description required in the past (which was a burden to maintain, and which was seldom up-to-date) is no longer needed. Moreover, the design of GAMS incorporates the following features that specifically address the user's documentation needs:

- ➤ A GAMS model representation is concise, and makes full use of the elegance of the mathematical representation.
- ➤ All data transformations are specified concisely and algebraically. This means that all data can be entered in their most elemental form and that all transformations made in constructing the model and in reporting are available for inspection.
- Explanatory text can be made part of the definition of all symbols and is reproduced whenever associated values are displayed.
- \succ All information needed to understand the model is in one document.

Of course some discipline is needed to take full advantage of these design features, but the aim is to make models more accessible, more understandable, more verifiable, and hence more credible.

1.2.3 Portability

The GAMS system is designed so that models can be solved on different types of computers with no change. A model developed on a small personal computer can later be solved on a large mainframe. One person can develop a model that is later used by others, who may be physically distant from the original developer. In contrast to previous approaches, only one document need be moved — the GAMS statement of the model. It contains all the data and logical specifications needed to solve the model.

1.2.4 User Interface

Portability concerns also have implications for the user interface. The basic GAMS system is file-oriented, and no special editor or graphical input and output routines exist. Rather than burden the user with having to learn yet another set of editing commands, GAMS offers an open architecture in which each user can use his word processor or editor of choice. This basic user interface facilitates the integration of GAMS with a variety of existing and future user environments.

1.2.5 Model Library

When architects begin to design a new building, they develop the new structure by using ideas and techniques that have been tested in previous structures. The same is true in other fields: design elements from previous projects serve as sources of ideas for new developments.

From the early stages of the development of GAMS we have collected models to be used in a library of examples. Many of these are standard textbook examples and can be used in classes on problem formulation or to illustrate points about GAMS. Others are models that have been used in policy or sector analysis and are interesting for both the methods and the data they use. All the substantive models in the library are described in the open literature. A collection of models is now included with all GAMS systems, along with a database to help users locate examples that cover countries, sectors, or topics of interest to them.

The syntax used to introduce features in the various chapters are presented using the Backus-Naur form (BNF) notation where:

- [] denotes that the enclosed construct is optional,
- {} denotes that the enclosed construct may be repeated zero or more times, and
- denotes that there is an *or* operator across the arguments on both sides of the symbol.

1.3 Organization of the Book

Some introductions to software systems are like reference manuals: they describe each command in detail. Others take you step by step through a small number of examples. This book uses elements of both approaches. The text is divided into three parts. The first part (Chapters 1 and 2) is introductory. Chapter 2 is a self-contained tutorial that guides you through a single example, a small transportation model, in some detail: you can quickly investigate the flavor of GAMS by reading it.

The second part (Chapters 3 to 17) comprises the meat of the book. The components of the GAMS language are introduced in an ordered way, interspersed with detailed examples that are often drawn from the model library. All models from the model library are enclosed in square parenthesis (for example, [TRNSPORT]). Some specialized material has deliberately been omitted in this process because the primary aim is to make GAMS accessible to the widest possible audience, especially those without extensive experience with computers or mathematical programming systems. Some familiarity with quantitative methods and mathematical representations is assumed.

The third part consists of specialized discussions of advanced topics and can be studied as needed. Users with large, complex, or expensive models will find much useful material in this part.

 $\mathbf{2}$

A GAMS Tutorial by Richard E. Rosenthal

2.1 Introduction

The introductory part of this book ends with a detailed example of the use of GAMS for formulating, solving, and analyzing a small and simple optimization problem. Richard E. Rosenthal of the Naval Postgraduate School in Monterey, California wrote it. The example is a quick but complete overview of GAMS and its features. Many references are made to other parts of the book, but they are only to tell you where to look for more details; the material here can be read profitably without reference to the rest of the book.

The example is an instance of the transportation problem of linear programming, which has historically served as a 'laboratory animal' in the development of optimization technology. [See, for example, Dantzig $(1963)^1$.] It is a good choice for illustrating the power of algebraic modeling languages like GAMS because the transportation problem, no matter how large the instance at hand, possesses a simple, exploitable algebraic structure. You will see that almost all of the statements in the GAMS input file we are about to present would remain unchanged if a much larger transportation problem were considered.

In the familiar transportation problem, we are given the supplies at several plants and the demands at several markets for a single commodity, and we are given the unit costs of shipping the commodity from plants to markets. The economic question is: how much shipment should there be between each plant and each market so as to minimize total transport cost?

The algebraic representation of this problem is usually presented in a format similar to the following.

Indices: i = plants

i = markets

Given Data:

 $a_i =$ supply of commodity of plant i (in cases)

 $b_j =$ demand for commodity at market j

 $c_{ij} = \text{cost per unit shipment between plant } i \text{ and market } j (\$/\text{case})$

Decision Variables:

 x_{ij} = amount of commodity to ship from plant *i* to market *j* (cases),

where $x_{ij} \ge 0$, for all i, j

Constraints:

Observe supply limit at plant i :	$\sum_{j} x_{ij} \le a_j$	for all i	(cases)
Satisfy demand at market j :	$\sum_{i}^{s} x_{ij} \ge b_j$	for all j	(cases)
Objective Function:	Minimize $\sum_{i} \sum_{j} c_{ij} x_{ij}$		(K)

Note that this simple example reveals some modeling practices that we regard as good habits in general and that are consistent with the design of GAMS. First, all the entities of the model are identified (and grouped) by type. Second, the ordering of entities is chosen so that no symbol is referred to before it is defined. Third, the

¹Dantzig, George B. (1963). Linear Programming and Extensions, Princeton University Press, Princeton N.J.

units of all entities are specified, and, fourth, the units are chosen to a scale such that the numerical values to be encountered by the optimizer have relatively small absolute orders of magnitude. (The symbol \$K here means thousands of dollars.)

The names of the types of entities may differ among modelers. For example, economists use the terms *exogenous* variable and *endogenous variable* for *given data* and *decision variable*, respectively. In GAMS, the terminology adopted is as follows: indices are called **sets**, given data are called **parameters**, decision variables are called **variables**, and constraints and the objective function are called **equations**.

The GAMS representation of the transportation problem closely resembles the algebraic representation above. The most important difference, however, is that the GAMS version can be read and processed by the computer.

Plants	Shipping Di	stances to Mari	kets (1000 miles)	Supplies
	New York	Chicago	Topeka	
Seattle	2.5	1.7	1.8	350
San Diego	2.5	1.8	1.4	600
Demands	325	300	275	

Table 2.1: Data for the transportation problem (adapted from Dantzig, 1963)

As an instance of the transportation problem, suppose there are two canning plants and three markets, with the data given in table 2.1. Shipping distances are in thousands of miles, and shipping costs are assumed to be \$90.00 per case per thousand miles. The GAMS representation of this problem is as follows:

```
Sets
    i canning plants / seattle, san-diego /
    j markets / new-york, chicago, topeka /;
Parameters
```

```
a(i)
           capacity of plant i in cases
            seattle
                        350
       /
            san-diego
                        600
           demand at market j in cases
     b(j)
            new-york
                        325
                        300
            chicago
            topeka
                        275
                            /;
Table d(i,j) distance in thousands of miles
                  new-york
                                 chicago
                                               topeka
    seattle
                     2.5
                                   1.7
                                                 1.8
    san-diego
                     2.5
                                   1.8
                                                 1.4
                                                      :
Scalar f freight in dollars per case per thousand miles /90/ ;
Parameter c(i,j) transport cost in thousands of dollars per case ;
          c(i,j) = f * d(i,j) / 1000;
Variables
     x(i,j)
             shipment quantities in cases
             total transportation costs in thousands of dollars ;
     z
Positive Variable x ;
Equations
     cost
                 define objective function
     supply(i)
                 observe supply limit at plant i
     demand(j)
                 satisfy demand at market j ;
cost ..
               z =e= sum((i,j), c(i,j)*x(i,j));
supply(i) ..
               sum(j, x(i,j)) =l= a(i) ;
```

```
demand(j) .. sum(i, x(i,j)) =g= b(j);
Model transport /all/;
Solve transport using lp minimizing z;
Display x.l, x.m;
```

If you submit a file containing the statements above as input to the GAMS program, the transportation model will be formulated and solved. Details vary on how to invoke GAMS on different of computers, but the simplest ('no frills') way to call GAMS is to enter the word GAMS followed by the input file's name. You will see a number of terse lines describing the progress GAMS is making, including the name of the file onto which the output is being written. When GAMS has finished, examine this file, and if all has gone well the optimal shipments will be displayed at the bottom as follows.

	new-york	chicago	topeka
seattle	50.000	300.000	
san-diego	275.000		275.000

You will also receive the marginal costs (simplex multipliers) below.

	chicago	topeka
seattle		0.036
san-diego	0.009	

These results indicate, for example, that it is optimal to send nothing from Seattle to Topeka, but if you insist on sending one case it will add .036 \$K (or \$36.00) to the optimal cost. (Can you prove that this figure is correct from the optimal shipments and the given data?)

2.2 Structure of a GAMS Model

For the remainder of the tutorial, we will discuss the basic components of a GAMS model, with reference to the example above. The basic components are listed in table 2.2.

There are optional input components, such as edit checks for bad data and requests for customized reports of results. Other optional advanced features include saving and restoring old models, and creating multiple models in a single run, but this tutorial will discuss only the basic components.

Before treating the individual components, we give a few general remarks.

- 1. A GAMS model is a collection of statements in the GAMS Language. The only rule governing the ordering of statements is that an entity of the model cannot be referenced before it is declared to exist.
- 2. GAMS statements may be laid out typographically in almost any style that is appealing to the user. Multiple lines per statement, embedded blank lines, and multiple statements per line are allowed. You will get a good idea of what is allowed from the examples in this tutorial, but precise rules of the road are given in the next Chapter.
- 3. When you are a beginning GAMS user, you should terminate every statement with a semicolon, as in our examples. The GAMS compiler does not distinguish between upper-and lowercase letters, so you are free to use either.
- 4. Documentation is crucial to the usefulness of mathematical models. It is more useful (and most likely to be accurate) if it is embedded within the model itself rather than written up separately. There are at least two ways to insert documentation within a GAMS model. First, any line that starts with an asterisk in column 1 is disregarded as a comment line by the GAMS compiler. Second, perhaps more important, documentary text can be inserted within specific GAMS statements. All the lowercase words in the transportation model are examples of the second form of documentation.

Inputs:

• Sets

Declaration Assignment of members

- Data (Parameters, Tables, Scalars) Declaration Assignment of values
- Variables

Declaration Assignment of type

- Assignment of bounds and/or initial values (optional)
- Equations

Declaration Definition

- Model and Solve statements
- Display statement (optional)

Table 2.2: The basic components of a GAMS model

- 5. As you can see from the list of input components above, the creation of GAMS entities involves two steps: a declaration and an assignment or definition. *Declaration* means declaring the existence of something and giving it a name. *Assignment* or *definition* means giving something a specific value or form. In the case of equations, you must make the declaration and definition in separate GAMS statements. For all other GAMS entities, however, you have the option of making declarations and assignments in the same statement or separately.
- 6. The names given to the entities of the model must start with a letter and can be followed by up to thirty more letters or digits.

2.3 Sets

Sets are the basic building blocks of a GAMS model, corresponding exactly to the indices in the algebraic representations of models. The Transportation example above contains just one **Set** statement:

Sets i canning plants / seattle, san-diego / j markets / new-york, chicago, topeka / ;

The effect of this statement is probably self-evident. We declared two sets and gave them the names i and j. We also assigned members to the sets as follows:

 $i = \{$ Seattle, San Diego $\}$

 $j = \{$ New York, Chicago, Topeka $\}$.

You should note the typographical differences between the GAMS format and the usual mathematical format for listing the elements of a set. GAMS uses slashes '/' rather than curly braces '{}' to delineate the set simply

Outputs:

- Echo Print
- Reference Maps
- Equation Listings
- Status Reports
- Results

because not all computer keyboards have keys for curly braces. Note also that multiword names like 'New York' are not allowed, so hyphens are inserted.

The lowercase words in the **sets** statement above are called *text*. Text is optional. It is there only for internal documentation, serving no formal purpose in the model. The GAMS compiler makes no attempt to interpret the text, but it saves the text and '*parrots*' it back to you at various times for your convenience.

It was not necessary to combine the creation of sets i and j in one statement. We could have put them into separate statements as follows:

Set i canning plants / seattle, san-diego / ;
Set j markets / new-york, chicago, topeka / ;

The placement of blank spaces and lines (as well as the choice of upper- or lowercase) is up to you. Each GAMS user tends to develop individual stylistic conventions. (The use of the singular set is also up to you. Using set in a statement that makes a single declaration and sets in one that makes several is good English, but GAMS treats the singular and plural synonymously.)

A convenient feature to use when you are assigning members to a set is the asterisk. It applies to cases when the elements follow a sequence. For example, the following are valid **set** statements in GAMS.

Set	t	time periods	/1991*2000/;
Set	m	machines	/mach1*mach24/;

Here the effect is to assign

 $t = \{1991, 1992, 1993, \dots, 2000\}$ $m = \{ \text{mach}_1, \text{mach}_2, \dots, \text{mach}_{24} \}.$

Note that set elements are stored as character strings, so the elements of t are not numbers.

Another convenient feature is the **alias** statement, which is used to give another name to a previously declared set. In the following example:

Alias (t,tp);

the name tp is like a t' in mathematical notation. It is useful in models that are concerned with the interactions of elements within the same set.

The sets i, j, t, and m in the statements above are examples of static sets, i.e., they are assigned their members directly by the user and do not change. GAMS has several capabilities for creating dynamic sets, which acquire their members through the execution of set-theoretic and logical operations. Dynamic sets are discussed in Chapter 12, page117. Another valuable advanced feature is multidimensional sets, which are discussed in Section 4.5, page 39.

2.4 Data

The GAMS model of the transportation problem demonstrates all of the three fundamentally different formats that are allowable for entering data. The three formats are:

• Lists

- Tables
- Direct assignments

The next three sub-sections will discuss each of these formats in turn.

2.4.1 Data Entry by Lists

The first format is illustrated by the first Parameters statement of the example, which is repeated below.

Parameters

```
a(i) capacity of plant i in cases
/ seattle 350
san-diego 600 /
b(j) demand at market j in cases
/ new-york 325
chicago 300
topeka 275 /;
```

This statement has several effects. Again, they may be self-evident, but it is worthwhile to analyze them in detail. The statement declares the existence of two parameters, gives them the names **a** and **b**, and declares their *domains* to be **i** and **j**, respectively. (A domain is the set, or tuple of sets, over which a parameter, variable, or equation is defined.) The statement also gives documentary text for each parameter and assigns values of **a**(**i**) and **b**(**j**) for each element of **i** and **j**. It is perfectly acceptable to break this one statement into two, if you prefer, as follows.

```
Parameters a(i) capacity of plant i in cases
    / seattle 350
    san-diego 600 / ;
Parameters b(j) demand at market j in cases
    / new-york 325
    chicago 300
    topeka 275 / ;
```

Here are some points to remember when using the list format.

- 1. The list of domain elements and their respective parameter values can be laid out in almost any way you like. The only rules are that the entire list must be enclosed in slashes and that the element-value pairs must be separated by commas or entered on separate lines.
- 2. There is no semicolon separating the element-value list from the name, domain, and text that precede it. This is because the same statement is being used for declaration and assignment when you use the list format. (An element-value list by itself is not interpretable by GAMS and will result in an error message.)
- 3. The GAMS compiler has an unusual feature called *domain checking*, which verifies that each domain element in the list is in fact a member of the appropriate set. For example, if you were to spell 'Seattle' correctly in the statement declaring Set i but misspell it as 'Seatle' in a subsequent element-value list, the GAMS compiler would give you an error message that the element 'Seatle' does not belong to the set i.
- 4. Zero is the default value for all parameters. Therefore, you only need to include the nonzero entries in the element-value list, and these can be entered in any order .
- 5. A scalar is regarded as a parameter that has no domain. It can be declared and assigned with a Scalar statement containing a *degenerate* list of only one value, as in the following statement from the transportation model.

Scalar f freight in dollars per case per thousand miles /90/;

If a parameter's domain has two or more dimensions, it can still have its values entered by the list format. This is very useful for entering arrays that are sparse (having few non-zeros) and super-sparse (having few distinct non-zeros).

2.4.2 Data Entry by Tables

Optimization practitioners have noticed for some time that many of the input data for a large model are derived from relatively small tables of numbers. Thus, it is very useful to have the table format for data entry. An example of a two-dimensional table (or matrix) is provided the transportation model:

Table d(i,j)	distance in thou	sands of miles	
	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4 ;

The effect of this statement is to declare the parameter d and to specify its domain as the set of ordered pairs in the Cartesian product of i and j. The values of d are also given in this statement under the appropriate heading. If there are blank entries in the table, they are interpreted as zeroes.

As in the list format, GAMS will perform domain checking to make sure that the row and column names of the table are members of the appropriate sets. Formats for entering tables with more columns than you can fit on one line and for entering tables with more than two dimensions are given in Chapter 5, page 43.

2.4.3 Data Entry by Direct Assignment

The direct assignment method of data entry differs from the list and table methods in that it divides the tasks of parameter declaration and parameter assignment between separate statements. The transportation model contains the following example of this method.

```
Parameter c(i,j) transport cost in thousands of dollars per case ; 
 c(i,j) = f * d(i,j) / 1000 ;
```

It is important to emphasize the presence of the semicolon at the end of the first line. Without it, the GAMS compiler would attempt to interpret both lines as parts of the same statement. (GAMS would fail to discern a valid interpretation, so it would send you a terse but helpful error message.)

The effects of the first statement above are to declare the parameter c, to specify the domain (i,j), and to provide some documentary text. The second statement assigns to c(i,j) the product of the values of the parameters f and d(i,j). Naturally, this is legal in GAMS only if you have already assigned values to f and d(i,j) in previous statements.

The direct assignment above applies to all (i,j) pairs in the domain of c. If you wish to make assignments for specific elements in the domain, you enclose the element names in quotes. For example,

c('Seattle','New-York') = 0.40;

is a valid GAMS assignment statement.

The same parameter can be assigned a value more than once. Each assignment statement takes effect immediately and overrides any previous values. (In contrast, the same parameter may not be declared more than once. This is a GAMS error check to keep you from accidentally using the same name for two different things.)

The right-hand side of an assignment statement can contain a great variety of mathematical expressions and built-in functions. If you are familiar with a scientific programming language such as FORTRAN or C, you will have no trouble in becoming comfortable writing assignment statements in GAMS. (Notice, however, that GAMS has some efficiencies shared by neither FORTRAN nor C. For example, we were able to assign c(i,j) values for all (i,j) pairs without constructing 'do loops'.)

The GAMS standard operations and supplied functions are given later. Here are some examples of valid assignments. In all cases, assume the left-hand-side parameter has already been declared and the right-hand-side parameters have already been assigned values in previous statements.

```
e = m*csquared;
w = 1/lamda;
eoq(i) = sqrt(2*demand(i)*ordcost(i)/holdcost(i));
t(i) = min(p(i), q(i)/r(i), log(s(i)));
euclidean(i,j) = qrt(sqr(xi(i) - xi(j) + sqr(x2(i) - x2(j)));
present(j) = future(j)*exp(-interest*time(j));
```

The summation and product operators to be introduced later can also be used in direct assignments.

2.5 Variables

The decision variables (or endogenous variables) of a GAMS-expressed model must be declared with a Variables statement. Each variable is given a name, a domain if appropriate, and (optionally) text. The transportation model contains the following example of a Variables statement.

```
Variables
    x(i,j) shipment quantities in cases
    z total transportation costs in thousands of dollars;
```

This statement results in the declaration of a shipment variable for each (i,j) pair. (You will see in Chapter 8, page 71, how GAMS can handle the typical real-world situation in which only a subset of the (i,j) pairs is allowable for shipment.)

The z variable is declared without a domain because it is a scalar quantity. Every GAMS optimization model must contain one such variable to serve as the quantity to be minimized or maximized.

Once declared, every variable must be assigned a type. The permissible types are given in table 2.3.

Variable Type	Allowed Range of Variable
<pre>free(default)</pre>	$-\infty$ to $+\infty$
positive	0 to $+\infty$
negative	$-\infty$ to 0
binary	0 or 1
integer	$0, 1, \ldots, 100$ (default)

Table 2.3: Permissible variable types

The variable that serves as the quantity to be optimized must be a scalar and must be of the **free** type. In our transportation example, z is kept free by default, but x(i,j) is constrained to non-negativity by the following statement.

```
Positive variable x ;
```

Note that the domain of \mathbf{x} should not be repeated in the type assignment. All entries in the domain automatically have the same variable type.

Section 2.10 describes how to assign lower bounds, upper bounds, and initial values to variables.

2.6 Equations

The power of algebraic modeling languages like GAMS is most apparent in the creation of the equations and inequalities that comprise the model under construction. This is because whenever a group of equations or inequalities has the same algebraic structure, all the members of the group are created simultaneously, not individually.

2.6.1 Equation Declaration

Equations must be declared and defined in separate statements. The format of the declaration is the same as for other GAMS entities. First comes the keyword, Equations in this case, followed by the name, domain and text of one or more groups of equations or inequalities being declared. Our transportation model contains the following equation declaration:

Equations cost define objective function supply(i) observe supply limit at plant i demand(j) satisfy demand at market j ;

Keep in mind that the word Equation has a broad meaning in GAMS. It encompasses both equality and inequality relationships, and a GAMS equation with a single name can refer to one or several of these relationships. For example, cost has no domain so it is a single equation, but supply refers to a set of inequalities defined over the domain i.

2.6.2 GAMS Summation (and Product) Notation

Before going into equation definition we describe the summation notation in GAMS. Remember that GAMS is designed for standard keyboards and line-by-line input readers, so it is not possible (nor would it be convenient for the user) to employ the standard mathematical notation for summations.

The summation notation in GAMS can be used for simple and complex expressions. The format is based on the idea of always thinking of a summation as an operator with two arguments: Sum(index of summation, summand) A comma separates the two arguments, and if the first argument requires a comma then it should be in parentheses. The second argument can be any mathematical expression including another summation.

As a simple example, the transportation problem contains the expression

Sum(j, x(i,j))

that is equivalent to $\sum_{j} x_{ij}$.

A slightly more complex summation is used in the following example:

Sum((i,j), c(i,j)*x(i,j))

that is equivalent to $\sum_i \sum_j c_{ij} x_{ij}$.

The last expression could also have been written as a nested summation as follows:

```
Sum(i, Sum(j, c(i,j)*x(i,j)))
```

In Section 11.3, page 110, we describe how to use the *dollar* operator to impose restrictions on the summation operator so that only the elements of i and j that satisfy specified conditions are included in the summation. Products are defined in GAMS using exactly the same format as summations, replacing Sum by Prod. For example,

prod(j, x(i, j))

is equivalent to: $\Pi_i x_{ij}$.

Summation and product operators may be used in direct assignment statements for parameters. For example,

```
scalar totsupply total supply over all plants;
totsupply = sum(i, a(i));
```

2.6.3 Equation Definition

Equation definitions are the most complex statements in GAMS in terms of their variety. The components of an equation definition are, in order:

- 1. The name of the equation being defined
- 2. The domain
- 3. Domain restriction condition (optional)
- 4. The symbol '...'
- 5. Left-hand-side expression
- 6. Relational operator: =1=, =e=, or =g=
- 7. Right-hand-side expression

The transportation example contains three of these statements.

cost .. z =e= sum((i,j), c(i,j)*x(i,j)); supply(i) .. sum(j, x(i,j)) =l= a(i); demand(j) .. sum(i, x(i,j)) =g= b(j);

Here are some points to remember.

> The power to create multiple equations with a single GAMS statement is controlled by the domain. For example, the definition for the **demand** constraint will result in the creation of one constraint for each element of the domain j, as shown in the following excerpt from the GAMS output.

```
DEMAND(new-york)..X(seattle,new-york) + X(san-diego,new-york)=G=325 ;
DEMAND(chicago)..X(seattle,chicago) + X(san-diego,chicago) =G=300 ;
DEMAND(topeka)..X(seattle,topeka) + X(san-diego,topeka) =G=275 ;
```

- > The key idea here is that the definition of the demand constraints is exactly the same whether we are solving the toy-sized example above or a 20,000-node real-world problem. In either case, the user enters just one generic equation algebraically, and GAMS creates the specific equations that are appropriate for the model instance at hand. (Using some other optimization packages, something like the extract above would be part of the input, not the output.)
- > In many real-world problems, some of the members of an equation domain need to be omitted or differentiated from the pattern of the others because of an exception of some kind. GAMS can readily accommodate this loss of structure using a powerful feature known as the *dollar* or '*such-that*' operator, which is not illustrated here. The domain restriction feature can be absolutely essential for keeping the size of a real-world model within the range of solvability.
- \succ The relational operators have the following meanings:
 - =1= less than or equal to
 - =g= greater than or equal to
 - =e= equal to
- ➤ It is important to understand the difference between the symbols '=' and '=e='. The '=' symbol is used only in direct assignments, and the '=e=' symbol is used only in equation definitions. These two contexts are very different. A direct assignment gives a desired value to a parameter before the solver is called. An equation definition also describes a desired relationship, but it cannot be satisfied until after the solver is called. It follows that equation definitions must contain variables and direct assignments must not.

- Variables can appear on the left or right-hand side of an equation or both. The same variable can appear in an equation more than once. The GAMS processor will automatically convert the equation to its equivalent standard form (variables on the left, no duplicate appearances) before calling the solver.
- ➤ An equation definition can appear anywhere in the GAMS input, provided the equation and all variables and parameters to which it refers are previously declared. (Note that it is permissible for a parameter appearing in the equation to be assigned or reassigned a value after the definition. This is useful when doing multiple model runs with one GAMS input.) The equations need not be defined in the same order in which they are declared.

2.7 Objective Function

This is just a reminder that GAMS has no explicit entity called the *objective function*. To specify the function to be optimized, you must create a variable, which is free (unconstrained in sign) and scalar-valued (has no domain) and which appears in an equation definition that equates it to the objective function.

2.8 Model and Solve Statements

The word model has a very precise meaning in GAMS. It is simply a collection of equations. Like other GAMS entities, it must be given a name in a declaration. The format of the declaration is the keyword model followed by the name of the model, followed by a list of equation names enclosed in slashes. If all previously defined equations are to be included, you can enter /all/ in place of the explicit list. In our example, there is one Model statement:

model transport /all/ ;

This statement may seem superfluous, but it is useful to advanced users who may create several models in one GAMS run. If we were to use the explicit list rather than the shortcut /all/, the statement would be written as

model transport / cost, supply, demand / ;

The domains are omitted from the list since they are not part of the equation name. The list option is used when only a subset of the existing equations comprises a specific model (or sub-model) being generated.

Once a model has been declared and assigned equations, we are ready to call the solver. This is done with a solve statement, which in our example is written as

solve transport using lp minimizing z ;

The format of the solve statement is as follows:

- 1. The key word solve
- 2. The name of the model to be solved
- 3. The key word using
- 4. An available solution procedure. The complete list is

lp	for linear programming
qcp	for quadratic constraint programming
nlp	for nonlinear programming
dnlp	for nonlinear programming with discontinuous derivatives
mip	for mixed integer programming
rmip	for relaxed mixed integer programming
miqcp	for mixed integer quadratic constraint programming

minlp	for mixed integer nonlinear programming
rmiqcp	for relaxed mixed integer quadratic constraint programming
rminlp	for relaxed mixed integer nonlinear programming
mcp	for mixed complementarity problems
mpec	for mathematical programs with equilibrium constraints
cns	for constrained nonlinear systems

- 5. The keyword 'minimizing' or 'maximizing'
- 6. The name of the variable to be optimized

2.9 Display Statements

The solve statement will cause several things to happen when executed. The specific instance of interest of the model will be generated, the appropriate data structures for inputting this problem to the solver will be created, the solver will be invoked, and the output from the solver will be printed to a file. To get the optimal values of the primal and/or dual variables, we can look at the solver output, or, if we wish, we can request a display of these results from GAMS. Our example contains the following statement:

display x.l, x.m ;

that calls for a printout of the final levels, x.1, and marginal (or reduced costs), x.m, of the shipment variables, x(i,j). GAMS will automatically format this printout in to dimensional tables with appropriate headings.

2.10 The '.lo, .l, .up, .m' Database

GAMS was designed with a small database system in which records are maintained for the variables and equations. The most important fields in each record are:

.lo lower bound
.l level or primal value
.up upper bound
.m marginal or dual value

The format for referencing these quantities is the variable or equation's name followed by the field's name, followed (if necessary) by the domain (or an element of the domain).

GAMS allows the user complete read-and write-access to the database. This may not seem remarkable to you now, but it can become a greatly appreciated feature in advanced use. Some examples of use of the database follow.

2.10.1 Assignment of Variable Bounds and/or Initial Values

The lower and upper bounds of a variable are set automatically according to the variable's type (free, positive, negative, binary, or integer), but these bounds can be overwritten by the GAMS user. Some examples follow.

```
x.up(i,j) = capacity(i,j) ;
x.lo(i,j) = 10.0 ;
x.up('seattle','new-york') = 1.2*capacity('seattle','new-york') ;
```

It is assumed in the first and third examples that capacity(i,j) is a parameter that was previously declared and assigned values. These statements must appear after the variable declaration and before the Solve statement. All the mathematical expressions available for direct assignments are usable on the right-hand side.

In nonlinear programming it is very important for the modeler to help the solver by specifying as narrow a range as possible between lower and upper bound. It is also very helpful to specify an initial solution from which the solver can start searching for the optimum. For example, in a constrained inventory model, the variables are quantity(i), and it is known that the optimal solution to the unconstrained version of the problem is a parameter called eoq(i). As a guess for the optimum of the constrained problem we enter

```
quantity.l(i) = 0.5*eoq(i) ;
```

(The default initial level is zero unless zero is not within the bounded range, in which case it is the bound closest to zero.)

It is important to understand that the .lo and .up fields are entirely under the control of the GAMS user. The .l and .m fields, in contrast, can be initialized by the user but are then controlled by the solver.

2.10.2 Transformation and Display of Optimal Values

(This section can be skipped on first reading if desired.)

After the optimizer is called via the solve statement, the values it computes for the primal and dual variables are placed in the database in the .1 and .m fields. We can then read these results and transform and display them with GAMS statements.

For example, in the transportation problem, suppose we wish to know the percentage of each market's demand that is filled by each plant. After the solve statement, we would enter

```
parameter pctx(i,j) perc of market j's demand filled by plant i;
pctx(i,j) = 100.0*x.l(i,j)/b(j) ;
display pctx ;
```

Appending these commands to the original transportation problem input results in the following output:

```
pctx percent of market j's demand filled by plant I
new-york chicago topeka
seattle 15.385 100.000
san-diego 84.615 100.000
```

For an example involving marginal, we briefly consider the *ratio constraints* that commonly appear in blending and refining problems. These linear programming models are concerned with determining the optimal amount of each of several available raw materials to put into each of several desired finished products. Let y(i,j) be the variable for the number of tons of raw material *i* put into finished product *j*. Suppose the *ratio constraint* is that no product can consist of more than 25 percent of one ingredient, that is,

y(i,j)/q(j) =1= .25;

for all i, j. To keep the model linear, the constraint is written as

ratio(i,j).. y(i,j) - .25*q(j) =1= 0.0 ;

rather than explicitly as a ratio.

The problem here is that ratio.m(i,j), the marginal value associated with the linear form of the constraint, has no intrinsic meaning. At optimality, it tells us by at most how much we can benefit from relaxing the linear constraint to

y(i,j) - .25*q(j) =l= 1.0;

Unfortunately, this relaxed constraint has no realistic significance. The constraint we are interested in relaxing (or tightening) is the nonlinear form of the ration constraint. For example, we would like to know the marginal benefit arising from changing the ratio constraint to

y(i,j)/q(j) =1= .26 ;

We can in fact obtain the desired marginals by entering the following transformation on the undesired marginals:

```
parameter amr(i,j) appropriate marginal for ratio constraint ;
amr(i,j) = ratio.m(i,j)*0.01*q.l(j) ;
display amr ;
```

Notice that the assignment statement for amr accesses both .m and .l records from the database. The idea behind the transformation is to notice that

y(i,j)/q(j) =1= .26 ;

is equivalent to

y(i,j) - .25*q(j) =l= 0.01*q(j) ;

2.11 GAMS Output

The default output of a GAMS run is extensive and informative. For a complete discussion, see Chapter 10, page 89. This tutorial discusses output partially as follows:

Echo Print	Reference Maps	Status Reports
Error Messages	Model Statistics	Solution Reports

A great deal of unnecessary anxiety has been caused by textbooks and users' manuals that give the reader the false impression that flawless use of advanced software should be easy for anyone with a positive pulse rate. GAMS is designed with the understanding that even the most experienced users will make errors. GAMS attempts to catch the errors as soon as possible and to minimize their consequences.

2.11.1 Echo Prints

Whether or not errors prevent your optimization problem from being solved, the first section of output from a GAMS run is an echo, or copy, of your input file. For the sake of future reference, GAMS puts line numbers on the left-hand side of the echo. For our transportation example, which luckily contained no errors, the echo print is as follows:

```
3
      Sets
 4
                                  / seattle, san-diego /
                canning plants
           i
 5
                                  / new-york, chicago, topeka / ;
           i
                markets
 6
 7
      Parameters
 8
 9
           a(i)
                  capacity of plant i in cases
10
                   seattle
                                350
                   san-diego
11
                                600
                                     1
12
                  demand at market j in cases
13
           b(j)
14
                   new-york
                                325
                                300
15
                   chicago
16
                                275
                                    /;
                   topeka
17
      Table d(i,j)
18
                     distance in thousands of miles
19
                         new-york
                                         chicago
                                                       topeka
20
          seattle
                            2.5
                                           1.7
                                                         1.8
```

```
21
          san-diego
                            2.5
                                          1.8
                                                       1.4 ;
22
23
      Scalar f freight in dollars per case per thousand miles /90/ ;
24
25
     Parameter c(i,j) transport cost in thousands of dollars per case;
26
27
                c(i,j) = f * d(i,j) / 1000;
28
29
      Variables
30
           x(i,j) shipment quantities in cases
31
           z
                  total transportation costs in thousands of dollars ;
32
33
      Positive Variable x ;
34
35
      Equations
36
                       define objective function
           cost
37
           supply(i)
                       observe supply limit at plant i
38
                       satisfy demand at market j ;
           demand(j)
39
                     z =e= sum((i,j), c(i,j)*x(i,j));
40
      cost ..
41
                     sum(j, x(i,j)) =l= a(i) ;
42
      supply(i) ..
43
44
      demand(j) .. sum(i, x(i,j)) =g= b(j);
45
46
      Model transport /all/ ;
47
48
      Solve transport using lp minimizing z ;
49
50
      Display x.l, x.m ;
51
```

The reason this echo print starts with line number 3 rather than line number 1 is because the input file contains two *dollar-print-control* statements. This type of instruction controls the output printing, but since it has nothing to do with defining the optimization model, it is omitted from the echo. The dollar print controls must start in column 1.

\$title a transportation model
\$offuppper

The **\$title** statement causes the subsequent text to be printed at the top of each page of output. The **\$offupper** statement is needed for the echo to contain mixed upper- and lowercase. Other available instructions are given in Appendix D, page 201.

2.11.2 Error Messages

When the GAMS compiler encounters an error in the input file, it inserts a coded error message inside the echo print on the line immediately following the scene of the offense. These messages always start with ******** and contain a '\$' directly below the point at which the compiler thinks the error occurred. The \$ is followed by a numerical error code, which is explained after the echo print. Several examples follow.

Example 1: Entering the statement

set q quarterly time periods / spring, sum, fall, wtr / ;

results in the echo

1 set q quarterly time periods / spring, sum, fall, wtr / ;
**** \$160

In this case, the GAMS compiler indicates that something is wrong with the set element sum. At the bottom of the echo print, we see the interpretation of error code 160:

Error Message 160 UNIQUE ELEMENT EXPECTED

The problem is that **sum** is a reserved word denoting summation, so our set element must have a unique name like 'summer'. This is a common beginner's error. The complete list of reserved words is shown in the next chapter.

Example 2: Another common error is the omission of a semicolon preceding a direct assignment or equation definition. In our transportation example, suppose we omit the semicolon prior to the assignment of c(i,j), as follows.

```
parameter c(i,j) transport cost in 1000s of dollars per case c(i,j) = f * d(i,j) / 1000 ;
```

Here is the resulting output.

```
16
       parameter c(i,j) transport cost in 1000s of dollars per case
                 c(i,j) = f*d(i,j)/1000
 17
****
                     $97
                            $195$96$194$1
Error Message
      REAL NUMBER EXPECTED
 1
 96
     BLANK NEEDED BETWEEN IDENTIFIER AND TEXT
      (-OR-ILLEGAL CHARACTER IN IDENTIFIER)
      (-OR-CHECK FOR MISSING ';' ON PREVIOUS LINE)
     EXPLANATORY TEXT CAN NOT START WITH '$', '=', or '...'
 97
      (-OR-CHECK FOR MISSING ';' ON PREVIOUS LINE)
194
      SYMBOL REDEFINED
      SYMBOL REDEFINED WITH A DIFFERENT TYPE
195
```

It is not uncommon for one little offense like our missing semicolon to generate five intimidating error messages. The lesson here is: concentrate on fixing the first error and ignore the other! The first error detected (in line 17), code 97, indicate that GAMS thinks the symbols in line 17 are a continuation of the documentary text at the end of line 16 rather than a direct assignment as we intended. The error message also appropriately advises us to check the preceding line for a missing semicolon.

Unfortunately, you cannot always expect error messages to be so accurate in their advice. The compiler cannot read your mind. It will at times fail to comprehend your intentions, so learn to detect the causes of errors by picking up the clues that abound in the GAMS output. For example, the missing semicolon could have been detected by looking up the c entry in the cross-reference list (to be explained in the next section) and noticing that it was never assigned.

SYMBOLTYPEREFERENCESCPARAMDECLARED15REF17

Example 3: Many errors are caused merely by spelling mistakes and are caught before they can be damaging. For example, with 'Seattle' spelled in the table differently from the way it was introduced in the set declaration, we get the following error message.

```
4
       sets
                  canning plants /seattle, san-diego /
   5
              i
   6
                  markets /new-york, chicago, topeka / ;
              i
   7
   8
       table d(i,j) distance in thousand of miles
   9
               new-york
                          chicago
                                   topeka
                                1.7
 10
          seatle
                       2.5
                                           1.8
****
             $170
          san-diego
                                                1.4
 11
                       2.5
                                1.8
                                                      ;
Error Message
170 DOMAIN VIOLATION FOR ELEMENT
```

Example 4: Similarly, if we mistakenly enter dem(j) instead of b(j) as the right-hand side of the demand constraint, the result is

45 demand(j) .. sum(i, x(i,j)) =g= dem(j) ;
**** \$
Error Message
140 UNKNOWN SYMBOL, ENTERED AS PARAMETER

Example 5: The next example is a mathematical error, which is sometimes committed by novice modelers and which GAMS is adept at catching. The following is mathematically inconsistent and, hence, is not an interpretable statement.

For all
$$i$$
, $\sum_{i} x_{ij} = 100$

There are two errors in this equation, both having to do with the control of indices. Index i is over-controlled and index j is under-controlled.

You should see that index i is getting conflicting orders. By appearing in the quantifier 'for all i', it is supposed to remain fixed for each instance of the equation. Yet, by appearing as an index of summation, it is supposed to vary. It can't do both. On the other hand, index j is not controlled in any way, so we have no way of knowing which of its possible values to use.

If we enter this meaningless equation into GAMS, both errors are correctly diagnosed.

meaninglss(i) .. sum(i, x(i,j)) =e= 100;
**** \$125 \$149
ERROR MESSAGES
125 SET IS UNDER CONTROL ALREADY [This refers to set i]
149 uncontrolled set entered as constant [This refers to set j]

A great deal more information about error reporting is given in Section 10.6, page 102. Comprehensive error detection and well-designed error messages are a big help in getting models implemented quickly and correctly.

2.11.3 Reference Maps

The next section of output, which is the last if errors have been detected, is a pair of *reference maps* that contain summaries and analyses of the input file for the purposes of debugging and documentation.

The first reference map is a *cross-reference map* such as one finds in most modern compilers. It is an alphabetical, cross-referenced list of all the entities (sets, parameters, variables, and equations) of the model. The list shows the type of each entity and a coded reference for each appearance of the entity in the input. The cross-reference map for our transportation example is as follows (we do not display all tables).

SYMBOL	TYPE	REFERENCES					
Α	PARAM	DECLARED	9	DEFINED	10	REF	42
В	PARAM	DECLARED	13	DEFINED	14	REF	44
С	PARAM	DECLARED	25	ASSIGNED	27	REF	40
COST	EQU	DECLARED	36	DEFINED	40	IMPL-ASN	48
		REF	46				
D	PARAM	DECLARED	18	DEFINED	18	REF	27
DEMAND	EQU	DECLARED	38	DEFINED	44	IMPL-ASN	48
		REF	46				
F	PARAM	DECLARED	23	DEFINED	23	REF	27
	SET	DECLARED	4	DEFINED	4	REF	9
		18	25	27	30	37	2*40
		2*42	44	CONTROL	27	40	42
		44					
J	SET	DECLARED	5	DEFINED	5	REF	13
		18	25	27	30	38	2*40
		42	2*44	CONTROL	27	40	42
		44					
SUPPLY	EQU	DECLARED	37	DEFINED	42	IMPL-ASN	48
		REF	46				
TRANSPORT	MODEL	DECLARED	46	DEFINED	46	IMPL-ASN	48
		REF	48				
Х	VAR	DECLARED	30	IMPL-ASN	48	REF	33
		40	42	44	2*50		

Z VAR DECLARED 31 IMPL-ASN 48 REF 40

For example, the cross-reference list tells us that the symbol A is a parameter that was declared in line 10, defined (assigned value) in line 11, and referenced in line 43. The symbol I has a more complicated entry in the cross-reference list. It is shown to be a set that was declared and defined in line 5. It is referenced once in lines 10, 19, 26, 28, 31, 38, 45 and referenced twice in lines 41 and 43. Set I is also used as a controlling index in a summation, equation definition or direct parameter assignment in lines 28, 41, 43 and 45.

For the GAMS novice, the detailed analysis of the cross-reference list may not be important. Perhaps the most likely benefit he or she will get from the reference maps will be the discovery of an unwanted entity that mistakenly entered the model owing to a punctuation or syntax error.

The second part of the reference map is a list of model entities grouped by type and listed with their associated documentary text. For example, this list is as follows.

```
sets
i
            canning plants
j
            markets
parameters
            capacity of plant i in cases
а
b
            demand at market j in cases
с
            transport cost in 1000s of dollars per case
d
            distance in thousands of miles
f
            freight in dollars per case per thousand miles
variables
            shipment quantities in cases
х
z
            total transportation costs in 1000s of dollars
equations
            define objective function
cost
            satisfy demand at market j
demand
            observe supply limit at plant i
supply
models
transport
```

2.11.4 Equation Listings

Once you succeed in building an input file devoid of compilation errors, GAMS is able to generate a model. The question remains, and only you can answer it, does GAMS generate the model you intended?

The equation listing is probably the best device for studying this extremely important question.

A product of the solve command, the equation listing shows the specific instance of the model that is created when the current values of the sets and parameters are plugged into the general algebraic form of the model. For example, the generic demand constraint given in the input file for the transportation model is

demand(j) .. sum(i, x(i,j)) = g = b(j);

while the equation listing of specific constraints is

```
------demand =g= satisfy demand at market j
demand(new-york).. x(seattle, new-york) +x(san-diego, new-york) =g= 325 ;
demand(chicago).. x(seattle, chicago) +x(san-diego, chicago) =g= 300 ;
demand(topeka).. x(seattle, topeka) +x(san-diego, topeka) =g= 275 ;
```

The default output is a maximum of three specific equations for each generic equation. To change the default, insert an input statement prior to the solve statement:

option limrow = r ;

where **r** is the desired number.

The default output also contains a section called the column listing, analogous to the equation listing, which shows the coefficients of three specific variables for each generic variable. This listing would be particularly useful for verifying a GAMS model that was previously implemented in MPS format. To change the default number of specific column printouts per generic variable, the above command can be extended:

```
option limrow = r, limcol = c ;
```

where c is the desired number of columns. (Setting limrow and limcol to 0 is a good way to save paper after your model has been debugged.)

In nonlinear models, the GAMS equation listing shows first-order Taylor approximations of the nonlinear equations. The approximations are taken at the starting values of the variables.

2.11.5 Model Statistics

MODEL STATISTICS

The last section of output that GAMS produces before invoking the solver is a group of statistics about the model's size, as shown below for the transportation example.

6 7

BLOCKS OF EQUATIONS	3	SINGLE EQUATIONS
BLOCKS OF VARIABLES	2	SINGLE VARIABLES
NON ZERO ELEMENTS	19	

The BLOCK counts refer to the number of generic equations and variables. The SINGLE counts refer to individual rows and columns in the specific model instance being generated. For nonlinear models, some other statistics are given to describe the degree of non-linearity in the problem.

2.11.6 Status Reports

After the solver executes, GAMS prints out a brief *solve summary* whose two most important entries are SOLVER STATUS and the MODEL STATUS. For our transportation problem the solve summary is as follows:

		S O	LV	E	S	U	ΜM	A	R	Y		
	MODEL TYPE SOLVER	TRANS LP BDMLP	PORT	•		0 D F)BJE)IRE 'ROM	CT CT L	IVE ION INE	2 J 2	Z MINIMIZ 49	Έ
**** ****	SOLVER S	STATUS FATUS		1 1	NORMAL OPTIMA	CO L	MPL	ET:	ION	1		
****	OBJECTI	VE VAL	UE				153	.6	750)		
RESOU	JRCE USA	GE, LI JNT, L	MIT IMIT		(!	0.1 5	.10		1 1	100	00.000 00	

The status reports are preceded by the same ******** string as an error message, so you should probably develop the habit of searching for all occurrences of this string whenever you look at an output file for the first time. The desired solver status is 1 NORMAL COMPLETION, but there are other possibilities, documented in Section 10.5, page 95, which relate to various types of errors and mishaps.

There are eleven possible model status's, including the usual linear programming termination states (1 OPTIMAL, 3 UNBOUNDED, 4 INFEASIBLE), and others relating to nonlinear and integer programming. In nonlinear programming, the status to look for is 2 LOCALLY OPTIMAL. The most the software can guarantee for nonlinear programming is a local optimum. The user is responsible for analyzing the convexity of the problem to determine whether local optimality is sufficient for global optimality.

In integer programming, the status to look for is 8 INTEGER SOLUTION. This means that a feasible integer solution has been found. More detail follows as to whether the solution meets the relative and absolute optimality tolerances that the user specifies.

2.11.7 Solution Reports

If the solver status and model status are acceptable, then you will be interested in examining the results of the optimization. The results are first presented in as standard mathematical programming output format, with the added feature that rows and columns are grouped and labeled according to names that are appropriate for the specific model just solved. In this format, there is a line of printout for each row and column giving the lower limit, level, upper limit, and marginal. Generic equation block and the column output group the row output by generic variable block. Set element names are embedded in the output for easy reading. In the transportation example, the solver outputs for supply(i), demand(j), and x(i,j) are as follows:

EQU \$	SUPPLY	observe s	upply limi	t at plant.	i
	LOWER	LEVEL	UPPER	MARGINAL	
seattle san-diego	-INF -INF	350.000 550.000	350.000 600.000	EPS	
EQU 1	DEMAND	satisfy d	emand at m	market j	
	LOWER	LEVEL	UPPER	MARGINAL	
new-york chicago topeka	325.000 300.000 275.000	325.000 300.000 275.000	+INF +INF +INF	0.225 0.153 0.126	
VAR 2	X	shipment	quantities	in cases	
		LOWER	LEVEL	UPPER	MARGINAL
seattle seattle	.new-york .chicago	•	50.000 300.000	+INF +INF	•
seattle	.topeka			+INF	0.036
san-diego	.new-york		275.000	+INF	
san-diego	.chicago			+INF	0.009
san-diego	.topeka		275.000	+INF	

The single dots '.' in the output represent zeroes. The entry EPS, which stands for *epsilon*, mean very small but nonzero. In this case, EPS indicates degeneracy. (The slack variable for the Seattle supply constraint is in the basis at zero level. The marginal is marked with EPS rather than zero to facilitate restarting the optimizer from the old basis.)

If the solvers results contain either infeasibilities or marginal costs of the wrong sign, then the offending entries are marked with INFES or NOPT, respectively. If the problem terminates unbounded, then the rows and columns corresponding to extreme rays are marked UNBND.

At the end of the solvers solution report is a very important *report summary*, which gives a tally of the total number of non-optimal, infeasible, and unbounded rows and columns. For our example, the report summary shows all zero tallies as desired.

```
**** REPORT SUMMARY : 0 NONOPT
0 INFEASIBLE
0 UNBOUNDED
```

After the solver's report is written, control is returned from the solver back to GAMS. All the levels and marginals obtained by the solver are entered into the GAMS database in the .1 and .m fields. These values can then be transformed and displayed in any desired report. As noted earlier, the user merely lists the quantities to be displayed, and GAMS automatically formats and labels an appropriate array. For example, the input statement.
display x.l, x.m ;

results in the following output.

50 VARIABLE X.L shipment quantities in cases new-york chicago topeka seattle 50.000 300.000 275.000 san-diego 275.000 50 VARIABLE X.M shipment quantities in cases chicago topeka 0.036 seattle 0.009 san-diego

As seen in reference maps, equation listings, solution reports, and optional displays, GAMS saves the documentary text and '*parrots*' it back throughout the output to help keep the model well documented.

2.12 Summary

This tutorial has demonstrated several of the design features of GAMS that enable you to build practical optimization models quickly and effectively. The following discussion summarizes the advantages of using an algebraic modeling language such as GAMS versus a matrix generator or conversational solver.

- ➤ By using an algebra-based notation, you can describe an optimization model to a computer nearly as easily as you can describe it to another mathematically trained person.
- ➤ Because an algebraic description of a problem has generality, most of the statements in a GAMS model are reusable when new instances of the same or related problems arise. This is especially important in environments where models are constantly changing.
- ➤ You save time and reduce generation errors by creating whole sets of closely related constraints in one statement.
- You can save time and reduce input errors by providing formulae for calculating the data rather than entering them explicitly.
- ➤ The model is self-documenting. Since the tasks of model development and model documentation can be done simultaneously, the modeler is much more likely to be conscientious about keeping the documentation accurate and up to date.
- ➤ The output of GAMS is easy to read and use. The solution report from the solver is automatically reformatted so that related equations and variables are grouped together and appropriately labeled. Also, the display command allows you to modify and tabulate results very easily.
- If you are teaching or learning modeling, you can benefit from the insistence of the GAMS compiler that every equation be mathematically consistent. Even if you are an experienced modeler, the hundreds of ways in which errors are detected should greatly reduce development time.
- By using the *dollar* operator and other advanced features not covered in this tutorial, one can efficiently implement large-scale models. Specific applications of the dollar operator include the following:
 - 1. It can enforce logical restrictions on the allowable combinations of indices for the variables and equations to be included in the model. You can thereby screen out unnecessary rows and columns and keep the size of the problem within the range of solvability.
 - 2. It can be used to build complex summations and products, which can then be used in equations or customized reports.
 - 3. It can be used for issuing warning messages or for terminating prematurely conditioned upon context-specific data edits.

3

GAMS Programs

3.1 Introduction

This chapter provides a look at the structure of the GAMS language and its components. It should be emphasized again that GAMS is a programming language, and that programs must be written in the language to use it. A GAMS program is contained in a disk file, which is normally constructed with a text editor of choice. When GAMS is 'run', the file containing the program (the input file) is submitted to be processed. After this processing has finished the results, which are in the output file(s), can be inspected with a text editor. On many machines a few terse lines appear on the screen while GAMS runs, keeping the user informed about progress and error detection. But it is the responsibility of the user to inspect the output file carefully to see the results and to diagnose any errors.

The first time or casual reader can skip this chapter: the discussion of specific parts of the language in the next Chapters does not assume an understanding of this chapter.

3.2 The Structure of GAMS Programs

GAMS programs consist of one or more statements (sentences) that define data structures, initial values, data modifications, and symbolic relationships (equations). While there is no fixed order in which statements have to be arranged, the order in which data modifications are carried out is important. Symbols must be declared as to type before they are used, and must have values assigned before they can be referenced in assignment statements. Each statement is followed by a semicolon except the last statement, where a semicolon is optional.

3.2.1 Format of GAMS Input

GAMS input is free format. A statement can be placed anywhere on a line, multiple statements can appear on a line, or a statement can be continued over any number of lines as follows:

statement; statement; statement; statement; statement; the words that you are now reading is an example of a very long statement which is stretched over two lines;

Blanks and end-of-lines can generally be used freely between individual symbols or words. GAMS is not case sensitive, meaning that lower and upper case letters can be mixed freely but are treated identically. Up to 255 characters can be placed on a line and completely blank lines can be inserted for easier reading.

Not all lines are a part of the GAMS language. Two special symbols, the asterisk '*' and the dollar symbol '\$' can be used in the first position on a line to indicate a non-language input line. An asterisk in column one means

that the line will not be processed, but treated as a comment. A dollar symbol in the same position indicates that compiler options are contained in the rest of the line.

Multiple files can be used as input through the use of the **\$include** facility described in Appendix D. In short, the statement,

\$include file1

inserts the contents of the specified file (file1 in this case) at the location of the call. A more complex versions of this is the **\$batinclude** which is described in Appendix D.

3.2.2 Classification of GAMS Statements

Each statement in GAMS is classified into one of two groups:

- \succ declaration and definition statements; or
- \succ execution statements

A declaration statement describes the class of symbol. Often initial values are provided in a declaration, and then it may be called a definition. The specification of symbolic relationships for an equation is a definition. The declaration and definition statements are:

acronym	parameter	equation declaration
set	scalar	equation definition
alias	table	model
	variable	

Execution statements are instructions to carry out actions such as data transformation, model solution, and report generation. The execution statements are:

option	display	solve
assignment	abort	loop
for	while	repeat
	execute	

Although there is great freedom about the order in which statements can be placed in a GAMS program, certain orders are commonly used. The two most common arrangements are discussed in the next sub-section.

3.2.3 Organization of GAMS Programs

The two most common ways of organizing GAMS programs are shown in table 3.1. The first style places the data first, followed by the model and then the solution statements. In this style of organization, the sets are placed first. Then the data are specified with parameter, scalar, and table statements. Next the model is defined with the variable, equation declaration, equation definition, and model statement. Finally the model is solved and the results are displayed.

A second style emphasizes the model by placing it before the data. This is a particularly useful order when the model may be solved repeatedly with different data sets. There is a separation between declaration and definition.

For example, sets and parameters may be declared first with the statements

```
set c "crops" ;
parameter yield "crop yield" ;
```

and then defined later with a statement:

Style 1	Style 2
Data:	Model:
Set declarations and definitions	Set declarations
Parameter declarations and definitions	Parameter declarations
Assignments	Variable declarations
Displays	Equation declaration
Model:	Equation definition
Variable declarations	Model definition
Equation declaration	Data:
Equation definition	Set definitions
Model definition	Parameter definitions
Solution:	Assignments
Solve	Displays
Displays	Solution:
	Solve
	Displays

Table 3.1: Organization of GAMS programs

The first statement declares that the identifier c is a set and the second defines the elements in the set

Sets and parameters used in the equations must be declared before the equations are specified; they can be defined, however, after the equation specifications but before a specific equation is used in a solve statement. This gives GAMS programs substantial organizational flexibility.

3.3 Data Types and Definitions

There are five basic GAMS data types and each symbol or identifier must be declared to belong to one of the following groups:

acronyms	models	sets
equations	parameters	variables

Scalars and tables are not separate data types but are a shorthand way to declare a symbol to be a parameter, and to use a particular format for initializing the numeric data.

Definitions have common characteristics, for example:

parameter a (i,j) input-output matrix data-type-keyword identifier domain list text

The domain list and the text are always optional characteristics. Other examples are:

set time time periods; model turkey turkish fertilizer model; variables x,y,z ;

In the last example a number of identifiers (separated by commas) are declared in one statement.

3.4 Language Items

Before proceeding with more language details, a few basic symbols need to be defined and the rules for recognizing and writing them in GAMS established. These basic symbols are often called lexical elements and form the building blocks of the language. They are:

characters	delimiters	labels	reserved words and tokens
comments	text	numbers	identifiers (indents)

Each of these items are discussed in detail in the following sub-sections.

As noted previously, we can use any mix of lower and upper case. GAMS makes no distinction between upper and lower case.

3.4.1 Characters

A few characters are not allowed in a GAMS program because they are illegal or ambiguous on some machines. Generally all unprintable and control characters are illegal. The only place where any character is legal is in an '\$ontext-\$offtext' block as illustrated in the section on comments below. For completeness the full set of legal characters are listed in table 3.2. Most of the uncommon punctuation characters are not part of the language, but can be used freely in text or comments.

A to Z	alphabet	a to z	alphabet	0 to 9	numerals
&	ampersand	"	double quote	#	pound sign
*	asterisk	=	equals	?	question mark
Q	at	>	greater than	;	semicolon
\	back slash	<	less than	,	single quote
:	colon	—	minus	/	slash
,	comma	()	parenthesis		space
\$	dollar	[]	square brackets	-	underscore
	dot	{ }	braces	!	exclamation mark
+	plus	%	percent	^	circumflex

Table 3.2: Legal characters

3.4.2 Reserved Words

GAMS, like computer languages such as C and Pascal, uses reserved words (often also called keywords) that have predefined meanings. It is not permitted to use any of these for one's own definitions, either as identifiers or labels. The complete list of reserved words are listed in table 3.3. In addition, a small number of symbols constructed from non-alphanumeric characters have a meaning in GAMS.

abort	eps	integer	not	sameas	sum
acronym	eq	le	option	scalar	system
alias all	equation equations	lt maximizing	or ord	scalars semicont semiint	then until
and	files	minimizing	parameter	set	using
assign binary	for	model models	parameters positive	sets smax	variable variables
card	ge	na	prod	smin	while
diag display	gt if	ne negative	putpage	sos1	yes
else	inf	no	repeat	sos2	
	The	e reserved non-alp	hanumeric symbo	ols are:	
••	=1= =g=	=e= =n=	=x= =		++ **

Table 3.3: Reserved words and symbols

3.4.3 Identifiers

Identifiers are the names given to sets, parameters, variables, models, etc. GAMS requires an identifier to start with a letter followed by more letters or digits. The length of an identifier is currently limited to 63 characters.

Identifiers can only contain alphanumeric characters (letters or numbers). Examples of legal identifiers are:

```
a a15 revenue x0051
```

whereas the following identifiers are incorrect:

```
15 $casg milk&meat
```

A name used for one data type cannot be reused for another.

3.4.4 Labels

Labels are set elements. They may be up to 63 characters long and can be used in quoted or unquoted form.

The unquoted form is simpler to use but places restrictions on characters used, in that any unquoted label must start with a letter or digit and can only be followed by letters, digits, or the sign characters + and -. Examples of unquoted labels are:

Phos-Acid 1986 1952-53 A September H2S04 Line-1

In quoted labels, quotes are used to delimit the label, which may begin with and/or include any legal character. Either single or double quotes can be used but the closing quote has to match the opening one. A label quoted with double quotes can contain a single quote (and vice versa). Most experienced users avoid quoted labels because they can be tedious to enter and confusing to read. There are a couple of special circumstances. If one wants to make a label stand out, then one can, for instance, put asterisks in it and indent it. A more subtle example is that GAMS keywords can be used as labels if they are quoted. If one needs to use labels like no, ne or sum then they will have to be quoted. Examples of quoted labels are:

'*TOTAL*' 'MATCH' '10%INCR' '12"/FOOT' 'LINE 1'

Labels do not have a value. The label '1986' does not have the numerical value 1986 and the label '01' is different from the label '1'.

The rules for constructing identifiers and labels are shown in table 3.4.

	Identifiers	Unquoted Labels	Quoted Labels
Number of Characters	63	63	63
Must Begin With	A letter	A letter or a number	Any character
Permitted Special Characters	None	+ or $- $ characters	Any but the starting quote

Table 3.4: Rules for constructing identifiers and labels

3.4.5 Text

Identifiers and simple labels can also be associated with a line of descriptive text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the identifier.

Text can be quoted or unquoted. Quoted text can contain any character except the quote character used. Single or double quotes can be used but must match. Text has to fit on one line and cannot exceed 80 characters in length. Text used in unquoted form must follow a number of mild restrictions. Unquoted text cannot start with a reserved word, '..' or '=' and must not include semicolon ';', commas ',', or slashes '/'. End of lines terminate a text. These restrictions are a direct consequence of the GAMS syntax and are usually followed naturally by the user. Some examples are:

```
this is text
final product shipment (tpy)
"quoted text containing otherwise illegal characters ; /,"
'use single quotes to put a "double" quote in text'
```

3.4.6 Numbers

Numeric values are entered in a style similar to that used in other computer languages

Blanks can not be used in a number: GAMS treats a blank as a separator.

The common distinction between real and integer data types does not exist in GAMS. If a number is used without a decimal point it is still stored as a real number.

In addition, GAMS uses an extended range arithmetic that contains special symbols for infinity (INF), negative infinity (-INF), undefined (UNDF), epsilon (EPS), and not available (NA). One cannot enter UNDF; it is only produced by an operation that does not have a proper result, such as division by zero. All the other special symbols can be entered and used as if they were ordinary numbers.

The following example shows various legal ways of entering numbers:

0	156.70	-135	.095	1.
2e10	2e+10	15.e+10	.314e5	+1.7
0.0	.0	0.	INF	-INF
EPS	NA			

The letter \mathbf{e} denotes the well-known scientific notation allowing convenient representation of very large or small numbers.

For example:

 $1e-5 = 1 * 10^{-5} = 0.00001$ $3.56e6 = 3.56 * 10^{6} = 3,560,000$

- GAMS uses a smaller range of numbers than many computers are able to handle. This has been done to ensure that GAMS programs will behave in the same way on a wide variety of machines, including personal computers. A good general rule is to avoid using or creating numbers with absolute values greater than 1.0e+20.
- A number can be entered with up to ten significant digits on all machines, and more on some.

3.4.7 Delimiters

As mentioned before, statements are separated by a semicolon ';'. However, if the next statement begins with a reserved word (often called keyword in succeeding chapters), then GAMS does not require that the semicolon be used.

The characters comma ',' and slash '/' are used as delimiters in data lists, to be introduced later. The comma terminates a data element (as does an end-of-line) and the slash terminates a data list.

3.4.8 Comments

A comment is an explanatory text that is not processed or retained by the computer. There are three ways to include comments in a GAMS program.

The first, already mentioned above, is to start a line with an asterisk '*' in the first character position. The remaining characters on the line are ignored but printed on the output file.

The second is to use special 'block' delimiters that cause GAMS to ignore an entire section of the program. The **\$** symbol must be in the first character position. The choice between the two ways is a matter of individual taste or utility. The example below illustrates the use of the block comment.

```
$ontext
Following a $ontext directive in column 1 all lines are ignored by GAMS but
printed on the output file until the matching $offtext is encountered, also
in column 1. This facility is often used to logically remove parts of programs
that are not used every time, such as statements producing voluminous reports.
Every $ontext must have a matching $offtext in the same file
$offtext
```

The third style of comment allows embedding a comment within a line. It must be enabled with the compiler option **\$inlinecom** or **\$eolcom** as in the following example.

```
$eolcom #
$inlinecom {}
x = 1 ; # this is a comment
y = 2 ; { this is also a comment } z = 3 ;
```

3.5 Summary

This completes the discussion of the components of the GAMS language. Many unfamiliar terms used in this chapter have been further explained in the Glossary.

4

Set Definitions

4.1 Introduction

Sets are fundamental building blocks in any GAMS model. They allow the model to be succinctly stated and easily read. In this chapter we will discuss how sets are declared and initialized. There are some more advanced set concepts, such as assignments to sets as well as lag and lead operations, but these are not introduced until much later in the book. However the topics covered in this chapter will be enough to provide a good start on most models.

4.2 Simple Sets

A set S that contains the elements a, b and c is written, using normal mathematical notation, as:

 $S = \{a, b, c\}$

In GAMS notation, because of character set limitations, the same set must be written as

set S /a, b, c/

The set statement begins with the keyword set (or sets). S is the name of the set, and its members are a, b, and c. They are labels, but are often referred to as elements or members.

4.2.1 The Syntax

In general, the syntax in GAMS for simple sets is as follows:

```
set set_name ["text"] [/element ["text"] {,element ["text"]} /]
{,set_name ["text"] [/element ["text"] {,element ["text"]} /] };
```

set_name is the internal name of the set (also called an identifier) in GAMS. The accompanying text is used to describe the set or element immediately preceding it.

4.2.2 Set Names

The name of the set is an identifier. An identifier has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. This is enough to construct meaningful names, and explanatory text can be used to provide more details.

Examples of legal identifiers are

i i15 countries s0051

whereas the following identifiers are incorrect:

25 \$currency food&drink

4.2.3 Set Elements

The name of each set element can be up to 63 characters long, and can be used in quoted or unquoted form. The unquoted form is simpler to use but places restrictions on characters used, in that any unquoted label must start with a letter or digit and can only be followed by letters, digits, or the sign characters + and -. Examples of legal unquoted labels are:

Phos-Acid	1986	1952-53	A
September	H2S04	Line-1	

In quoted labels, quotes are used to delimit the label, which may begin with and/or include any legal character. Either single or double quotes can be used but the closing quote has to match the opening one. A label quoted with double quotes can contain a single quote (and vice versa). Most experienced users avoid quoted labels because they can be tedious to enter and confusing to read. There are a couple of special circumstances. If one wants to make a label stand out, then to put asterisks in it and indent it, as below, is common. A more subtle example is that it is possible to use GAMS keywords as labels if they are quoted. If one need to use labels like no, ne or sum then they will have to be quoted.

Examples of quoted labels are:

'*TOTAL*' 'Match' '10%incr' '12"/foot' 'Line 1'

Labels do not have a value. The label '1986' does not have the numerical value 1986 and the label '01' is different from the label '1'.

Each element in a set must be separated from other elements by a comma or by an end-of-line. In contrast, each element is separated from any associated text by a blank.

Consider the following example from the Egyptian fertilizer model [FERTS], where the set of fertilizer nutrients could be written as

```
set cq "nutrients" / N, P205 / ;
or as
set cq "nutrients" / N
P205 / ;
```

The order in which the set members are listed is normally not important. However, if the members represent, for example, time periods, then it may be useful to refer to *next* or *previous* member. There are special operations to do this, and they will be discussed in Chapter 13. For now, it is enough to remember that the order in which set elements are specified is not relevant, unless and until some operation implying order is used. At that time, the rules change, and the set becomes what we will later call an ordered set.

4.2.4 Associated Text

It is also possible to associate text with each set member or element. Explanatory text must not exceed 254 characters and must all be contained on the same line as the identifier or label it describes.

For example, label text for the set of final products in [SHALE] contains details of the units of measurement.

```
Set f "final products"
/yncrude "refined crude (million barrels)"
lpg "liquified petroleum gas(million barrels)"
ammonia "ammonia (million tons)"
coke "coke (million tons)"
sulfur "sulfur (million tons)"
/;
```

Notice that text may have embedded blanks, and may include special characters such as parentheses. There are, however, restrictions on special characters in text. Include slashes, commas or semicolons only if the text is enclosed in quotes. A set definition like

```
set prices prices of commodities in dollars/ounce
    / gold-price, sil-price / ;
```

will cause errors since the slash between dollars and ounce will signal the beginning of the set declaration, and the GAMS compiler will treat ounce as the name of the first element. Further, the slash before gold-price will be treated as the end of the set definition, and gold-price will be treated as a new set. However, by enclosing the explanatory text in quotes, this problem is avoided. The following text is valid:

set prices "prices of commodities in dollars/ounce"

4.2.5 Sequences as Set Elements

The asterisk '*' plays a special role in set definitions. It is used to relieve the tedium of typing a sequence of elements for a set, and to make intent clearer. For example in a simulation model there might be ten annual time periods from 1991 to 2000. Instead of typing ten years, the elements of this set can be written as

set t "time" /1991 * 2000 /;

which means that the set includes the ten elements 1991, 1992,...,2000. GAMS builds up these label lists by looking at the differences between the two labels. If the only characters that differ are digits, and if the number (say L) formed by these digits in the left one is less than that from the right one (R), then a label is constructed for every integer in the sequence L to R. Any non-numeric differences or other inconsistencies cause errors.

The following example illustrates the most general form of the 'asterisked' definition:

set g / a1bc * a20bc /;

Note that this is not the same as

set g / a01bc * a20bc /;

although the sets, which have 20 members each, have 11 members in common. As a last example, the following are all illegal because they are not consistent with the rule given above for making lists:

```
set illegal1 / a20bc * a10bc /
    illegal2 / a1x1 * a9x9 /
    illegal3 / a1 * b9 /;
```

Note one last time that set elements (often referred to as labels) can contain the sign characters '-' and '+' as well as letters and numbers.

4.2.6 Declarations for Multiple Sets

The keyword **set** (if you prefer, say **sets** instead: the two are equivalent) does not need to be used for each set, rather only at the beginning of a group of sets. It is often convenient to put a group of set declarations together

at the beginning of the program. When this is done the set keyword need only be used once. If you prefer to intermingle set declarations with other statements, you will have to use a new **set** statement for each additional group of sets.

The following example below shows how two sets can be declared together. Note that the semicolon is used only after the last set is declared.

```
sets
s "Sector" / manuf
agri
services
government /
r "regions" / north
eastcoast
midwest
sunbelt /;
```

4.3 The Alias Statement: Multiple Names for a Set

It is sometimes necessary to have more than one name for the same set. In input-output models, for example, each commodity may be used in the production of all other commodities and it is necessary to have two names for the set of commodities to specify the problem without ambiguity. In the general equilibrium model [ORANI], the set of commodities is written

set c "commodities" / food, clothing / ;

and a second name for the set c is established with either of the following statements

alias (c, cp) ;
alias (cp, c) ;

where cp is the new set that can be used instead of the original set c.

The newly introduced set can be used as an alternative name for the original set, and will always contain only the same elements as the original set.

The alias statement can be used to introduce more than one new name for the original set.

alias (c,cp, cpp, cppp);

where the new sets cp, cpp, cppp are all new names for the original set c.

The order of the sets in the alias statement does not matter. The only restriction set by GAMS is that exactly one of the sets in the statement be defined earlier. All the other sets are introduced by the alias statement.

We will not demonstrate the use of set aliases until later. Just remember they are used for cases when a set has to be referred to by more than one name.

4.4 Subsets and Domain Checking

It is often necessary to define sets whose members must all be members of some larger set. The syntax is:

```
set set_ident1 (set_ident2) ;
```

where set_ident1 is a subset of the larger set set_ident2.

For instance, we may wish to define the sectors in an economic model following the style in [CHENERY].

```
set
i "all sectors" / light-ind, food+agr, heavy-ind, services /
t(i) "traded sectors" / light-ind, food+agr, heavy-ind /
nt "non-traded sectors" / services / ;
```

Some types of economic activity, for example exporting and importing, may be logically restricted to a subset of all sectors. In order to model the trade balance, for example, we need to know which sectors are traded, and one obvious way is to list them explicitly, as in the definition of the set t above. The specification t(i) means that each member of the set t must also be a member of the set i. GAMS will enforce this relationship, which is called domain checking. Obviously the order of declaration is important: the membership of i must be known before t is declared for checking to be done. There will be much more on this topic in succeeding chapters. For now it is important to note that domain checking will find any spelling errors that might be made in establishing the members of the set t. These would cause errors in the model if they went undetected.

It is legal but unwise to define a subset without reference to the larger set, as is done above for the set nt. If services were misspelled no error would be marked, but the model would give incorrect results. So we urge you to use domain checking whenever possible. It catches errors and allows you to write models that are conceptually cleaner because logical relationships are made explicit.

This completes the discussion of sets in which the elements are simple. This is sufficient for most GAMS applications; however, there are a variety of problems for which it is useful to have sets that are defined in terms of two or more other sets.

4.5 Multi-dimensional Sets

It is often necessary to provide mappings between elements of different sets. For this purpose, GAMS allows the use of multi-dimensional sets.

 \square GAMS allows sets with up to 20 dimensions.

The next two sub-sections explain how to express one-to-one and many-to-many mappings between sets.

4.5.1 One-to-one Mapping

Consider a set whose elements are pairs: $A = \{(b, d), (a, c), (c, e)\}$.

In this set there are three elements and each element consists of a pair of letters. This kind of set is useful in many types of modeling. As an illustrative example, consider the world aluminum model [ALUM], where it is necessary to associate, with each bauxite-supplying country, a port that is near to the bauxite mines. The set of countries is

and the set of ports is

```
set p "ports"
/ kingston
s-domingo
georgetown
belem / ;
```

Then a set can be created to associate each port with its country, viz.,

set ptoc(p, c) "port to country relationship"

/ kingston .jamaica
s-domingo .haiti
georgetown .guyana
belem .brazil /;

The dot between kingston and jamaica is used to create one such pair. Blanks may be used freely around the dot for readability. The set ptoc has four elements, and each element consists of a port-country pair. The notation (p,c) after the set name ptoc indicates that the first member of each pair must be a member of the set p of ports, and that the second must be in the set c of countries. This is a second example of domain checking. GAMS will check the set elements to ensure that all members belong to the appropriate sets.

4.5.2 Many-to-many Mapping

A many-to-many mapping is needed in certain cases. Consider the following set

```
set i / a, b /
    j / c, d, e /
    ij1(i,j) /a.c, a.d/
    ij2(i,j) /a.c, b.c/
    ij3(i,j) /a.c, b.c, a.d, b.d/;

ij1 represents a one-to-many mapping where one element of i maps onto many elements of j.
ij2 represents a many-to-one mapping where many elements of i map onto one element of j.
ij3 is the most general case where many elements of i map on to many elements of j.
```

These sets can be written compactly as

```
set i / a, b /
    j / c, d, e /
    ij1(i,j) /a.(c,d)/
    ij2(i,j) /(a,b).c/
    ij3(I,j) /(a,b).(c,d)/;
```

The parenthesis provides a list of elements that can be expanded when creating pairs.

When complex sets like this are created, it is important to check that the desired set has been obtained. The checking can be done by using a display statement.

The concepts may be generalized to set with more than two labels per set element. Mathematically these are called 3-tuples, 4-tuples, or more generally, *n*-tuples.

This section ends with some examples to illustrate definitions of multi-label set elements. Some examples of the compact representation of sets of n-tuples using combinations of dots, parentheses and commas are shown in table 4.1.

Construct	Result
(a,b).c.d	a.c.d, b.c.d
(a,b).(c,d) .e	a.c.e, b.c.e, a.d.e, b.d.e
(a.1*3).c	(a.1, a.2, a.3).c or a.1.c, a.2.c, a.3.c
1*3. 1*3. 1*3	1.1.1, 1.1.2, 1.1.3,, 3.3.3

Table 4.1: Examples of the compact representation of sets

Note that the asterisk can also be used in conjunction with the dot. Recall that the elements of the list 1*4 are $\{1, 2, 3, 4\}$ when examining the examples in table 4.1.

4.6 Summary

In GAMS, a simple set consists of a set name and the elements of the set. Both the name and the elements may have associated text that explains the name or the elements in more detail. More complex sets have elements that are pairs or even n-tuples. These sets with pairs and n-tuples are ideal for establishing relationships between the elements in different sets. GAMS also uses a domain checking capability to help catch labeling inconsistencies and typographical errors made during the definition of related sets.

The discussion here has been limited to sets whose members are all specified as the set is being declared. For many models this is all you need to know about sets. Later we will discuss more complicated concepts, such as sets whose membership changes in different parts of the model (assignment to sets) and other set operations such as unions, complements and intersections. $\mathbf{5}$

Data Entry: Parameters, Scalars & Tables

5.1 Introduction

One of the basic design paradigms of the GAMS language has been to use data in its most basic form, which may be scalar, list oriented, or tables of two or more dimensions. Based on this criterion, three data types are introduced in this chapter.

ScalarSingle (scalar) data entry.ParameterList oriented data.TableTable oriented data. Must involve two or more dimensions.

Each of these data types will be explained in detail in the following sections.

Initialization of data can only be done once for parameters; thereafter data must be modified with assignment statements.

5.2 Scalars

The scalar statement is used to declare and (optionally) initialize a GAMS parameter of dimensionality zero. That means there are no associated sets, and that there is therefore exactly one number associated with the parameter.

5.2.1 The Syntax

In general, the syntax in GAMS for a scalar declaration is:

```
scalar[s] scalar_name [text] [/signed_num/]
{ scalar_name [text] [/signed_num/]};
```

Scalar_name is the internal name of the scalar (also called an identifier) in GAMS. The accompanying text is used to describe the element immediately preceding it. Signed_num is a signed number and is assigned to be the value of scalar_name.

As with all identifiers, scalar_name has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. Explanatory text must not exceed 254 characters and must all be contained on the same line as the identifier or label it describes.

5.2.2 An Illustrative Example

An example of a scalar definition in GAMS is shown below.

```
Scalars rho "discount rate" / .15 /
    irr "internal rate of return"
    life "financial lifetime of productive units" /20/;
```

The statement above initializes **rho** and **life**, but not **irr**. Later on another **scalar** statement can be used to initialize **irr**, or, (looking ahead to a notion that will be developed later), an assignment statement could be used to provide the value:

irr = 0.07;

5.3 Parameters

While parameter is a data type that encompasses scalars and tables, the discussion in this chapter will focus on the use of parameters in data entry. List oriented data can be read into GAMS using the parameter statement.

5.3.1 The Syntax

In general, the syntax in GAMS for a parameter declaration is:

Param_name is the internal name of the parameter (also called an identifier) in GAMS. The accompanying text is used to describe the parameter immediately preceding it. **Signed_num** is a signed number and is declared to be the value of the entry associated with the corresponding element.

As with all identifiers, **param_name** has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 long. Explanatory text must not exceed 254 characters and must all be contained on the same line as the identifier or label it describes.

A parameter may be indexed over one or more sets (the maximum number being 20). The elements in the data should belong to the set that the parameter is indexed over.

The default value of a parameter is 0.

Parameter initialization requires a list of data elements, each consisting of a label and a value. Slashes must be used at the beginning and end of the list, and commas must separate data elements entered more than one to a line. An equals sign or a blank may be used to separate the label-tuple from its associated value. A parameter can be defined in a similar syntax to that used for a set.

5.3.2 An Illustrative Examples

The fragment below is adapted from [MEXSS]. We also show the set definitions because they make the example clearer.

```
Set i "steel plants" / hylsa "monterrey"
hylsap "puebla" /
j "markets" / mexico-df, monterrey, guadalaja /;
parameter dd(j) distribution of demand
/ mexico-df 55,
guadalaja 15 /;
```

The domain checking specification for dd means that there will be a vector of data associated with it, one number corresponding to every member of the set j listed. The numbers are specified along with the declaration in a format very reminiscent of the way we specified sets: in this simple case a label followed by a blank separator and then a value. Any of the legal number entry formats are allowable for the value. The default data value is zero. Since monterrey has been left out of the data list, then the value associated with dd('monterrey'), the market share in monterrey, would be zero.

We can also put several data elements on a line, separated by commas:

As with sets, commas are optional at end-of-line.

5.3.3 Parameter Data for Higher Dimensions

A parameter can have up to 20 dimensions. The list oriented data initialization through the parameter statement can be easily extended to data of higher dimensionality. The label that appears on each line in the one-dimensional case is replaced by a label-tuple for higher dimensions. The elements in the *n*-tuple are separated by dots (.) just like in the case of multi-dimensional sets.

The following example illustrates the use of parameter data for higher dimensions:

```
parameter salaries(employee,manager,department)
/anderson .murphy .toy = 6000
hendry .smith .toy = 9000
hoffman .morgan .cosmetics = 8000 / ;
```

All the mechanisms using asterisks and parenthesized lists that we introduced in our discussion of sets are available here as well. Below is an artificial example, in which a very small fraction of the total data points are initialized. GAMS will mark an error if the same label combination (or label-tuple) appears more than once in a data list.

In this example, the twelve elements row1.col2 to row1.col7 and row4.col2 to row4.col7 are all initialized at 12, the single element row10.col10 at 17, and the seven elements rows1.col10 to row7.col10 at 33. The other 80 elements (out of a total of 100) remain at their default value, which is 0. This example shows the ability of GAMS to provide a concise initialization, or definition, for a sparse data structure.

5.4 Tables

Tabular data can be declared and initialized in GAMS using a table statement. For 2- and higher-dimensional parameters this provides a more concise and easier method of data entry than the list based approach, since each label appears only once (at least in small tables).

5.4.1 The Syntax

In general, the syntax in GAMS for a table declaration is:

table table_name [text] EOL
 element { element }
 element signed_num { signed_num} EOL
 {element signed_num { signed_num} EOL ;

Table_name is the internal name of the table (also called an identifier) in GAMS. The accompanying text is used to describe the parameter immediately preceding it. Signed_num is a signed number and is declared to be the value of the entry associated with the corresponding element.

I[™] The table statement is the only statement in the GAMS language that is not free format.

The following rules apply:

- The relative positions of all entries in a table are significant. This is the only statement where end of line (EOL) has meaning. The character positions of the numeric table entries must overlap the character positions of the column headings.
- \succ The column section has to fit on one line.
- \succ The sequence of signed numbers forming a row must be on the same line.
- \succ The element definition of a row can span more than one line.
- \succ A specific column can appear only once in the entire table.

The rules for forming simple tables are straightforward. The components of the header line are the by now familiar keyword-identifier-domain_list-text sequence, the domain-list and text being optional. Labels are used on the top and the left to map out a rectangular grid that contains the data values. The order of labels is unimportant, but if domain checking has been specified each label must match one in the associated set. Labels must not be repeated, but can be left out if the corresponding numbers are all zero or not needed. At least one blank must separate all labels and data entries. Blank entries imply that the default value (zero) will be associated with that label combination.

Notice also that, in contrast to the set, scalar, and parameter statements, only one identifier can be declared and initialized in a table statement.

5.4.2 An Illustrative Example

The example below, adapted from [KORPET], is preceded by the appropriate set definitions,

```
sets
     i
          "plants"
          inchon,ulsan,yosu /
       1
          "productive units"
      m
                                   "atmospheric distillation unit"
              atmos-dist
                                  "steam cracker"
              steam-cr
                                   "aromatics unit"
              aromatics
                                  "hydrodealkylator" / ;
              hydrodeal
table ka(m,i) "initial cap. of productive units (100 tons per yr)"
                 inchon
                              ulsan
                                          vosu
                   3702
                                          9875
atmos-dist
                              12910
                                          1207
steam-cr
                                517
aromatics
                                181
                                           148
                                180
hvdrodeal
                                              ;
```

In the example above, the row labels are drawn from the set m, and those on the column from the set i. Note that the data for each row is aligned under the corresponding column headings.

If there is any uncertainty about which data column a number goes with, GAMS will protest with an error message and mark the ambiguous entry.

5.4.3 Continued Tables

If a table has too many columns to fit nicely on a single line, then the columns that don't fit can be continued on additional lines. We use the same example to illustrate:

table ka(m,i)	initial	cap. of	productive	units	(100	tons	\mathtt{per}	yr)
	inchon	ulsan						
atmos-dist	3702	12910						
steam-cr		517						
aromatics		181						
hydrodeal		180						
+	yosu							
atmos-dist	9875							
steam-cr	1207							
aromatics	148 ;							

The crucial item is the plus '+' sign above the row labels and to the left of the column labels in the continued part of the table. The row labels have been duplicated, except that hydroreal has been left out, not having associated data. Tables can be continued as many times as necessary.

5.4.4 Tables with more than Two Dimensions

A table can have up to 20 dimensions. Dots are again used to separate adjacent labels, and can be used in the row or column position. The label on the left of the row corresponds to the first set in the domain list, and that on the right of each column header to the last. Obviously there must be the same number of labels associated with each number in the table, as there are sets in the domain list.

The actual layout chosen will depend on the size of the controlling sets and the amount of data, and the ideal choice should be the one that provides the most intuitively satisfactory way of organizing and inspecting the data. Most people can more easily look down a column of numbers than across a row, but to put extra labels on the row leads to a greater density of information.

The following example, adapted from [MARCO], illustrates the use of tables with more than two dimensions.

```
Sets ci
           "commodities :
                            intermediate"
       / naphtha
                    "naphtha"
                    "distillate"
         dist
         gas-oil
                    "gas-oil"
           "commodities : crude oils"
      cr
                     "mid-continent"
       / mid-c
         w-tex
                     "west-texas"

          "attributes of intermediate products"
      q
         density, sulfur /
                            ;
table attrib(ci, cr, q)
                           blending attributes
                      density
                                  sulfur
naphtha. mid-c
                       272
                                   .283
naphtha. w-tex
                       272
                                  1.48
dist
                       292
                                   .526
      . mid-c
                                    2.83
dist
      . w-tex
                       297
                                   .98
gas-oil. mid-c
                       295
gas-oil. w-tex
                       303
                                  5.05
                                         :
```

The table attrib could also be laid out as shown below:

table	attrib (ci,cı	r,q) blending a	attributes	
	w-tex.densi	ity mid-c.densi	ty w-tex.sulfu	r mid-c.sulfur
naphth	a 272	272	1.48	.283
dist	297	297	2.83	.526
gas-oi	1 303	303	5.05	.98 ;

5.4.5 Condensing Tables

All the mechanisms using asterisks and parenthesized lists that were introduced in the discussion of sets are available here as well. The following example shows how repeated columns or rows can be condensed with asterisks and lists in parentheses follows. The set membership is not shown, but can easily be inferred.

```
table upgrade(strat,size,tech)
            small.tech1 small.tech2 medium.tech1 medium.tech2
strategy-1
                  .05
                               .05
                                              .05
                                                             .05
strategy-2
                  .2
                               .2
                                               .2
                                                             .2
                  .2
                               .2
                                               .2
                                                            .2
strategy-3
                                               .2
                                                             .2
strategy-4
table upgradex(strat, size, tech) alternative way of writing table
                                      tech1*tech2
strategy-1.(small,medium)
                                            .05
                                            .2
strategy-2*strategy-3.(small,medium)
trategy-4.medium
                                            .2;
display attrib, attribx;
```

Here we encounter the display statement again. It causes the data associated with upgrade and upgradex to be listed on the output file.

5.4.6 Handling Long Row Labels

It is possible to continue the row labels in a table on a second, or even third, line in order to accommodate a reasonable number of columns. The break must come after a dot, and the rest of each line containing an incomplete row label-tuple must be blank.

The following example, adapted from [INDUS], is used to illustrate. As written, this table actually has nine columns and many rows: we have just reproduced a small part to show continued row label-tuples.

```
table yield (c,t,s,w,z)
                              crop yield (metric tons per acre)
                                                       nwfp
                                                                pmw
wheat.(bullock, semi-mech).la-plant.
                                  (heavy, january)
                                                      .385
                                                               .338
wheat.(bullock, semi-mech).la-plant.light
                                                      .506
                                                               .446
wheat.(bullock, semi-mech).la-plant. standard
                                                      .592
                                                               .524
wheat.(bullock, semi-mech).(qk-harv, standard).
                                  (heavy, january) .439
                                                               .387
```

5.5 Acronyms

An acronym is a special data type that allows the use of strings as values.

5.5.1 The Syntax

The declaration for an acronymis similar to a set or parameter declaration in that several of them can be declared in one statement.

```
Acronym[s] acronym_name {,acronym_name}
```

Acronym_name is an identifier and follows the same naming convention as other identifiers like names of sets, parameters, or tables.

5.5.2 Illustrative Example

Consider the following example,

```
set machines /m-1*m-5/ ; acronyms monday, tuesday, wednesday, thursday, friday ;
parameter shutdown(machines) /
    m-1 tuesday
```

m-2 wednesday
m-3 friday
m-4 monday
m-5 thursday
/;

In the example above, data entries are in the form of strings like 'monday' and 'tuesday'. By declaring each of those character strings as acronyms, this kind of data entry can be used by GAMS. Sections 6.2.7 and 11.2.5 will explain the further use of acronyms once entered in this form.

5.6 Summary

In this chapter, the declaration and initialization of parameters with the **parameter**, **scalar** and the **table** statement have been discussed. The next chapter will describe how this data can be changed with assignment statements.

6

Data Manipulations with Parameters

6.1 Introduction

Data once initialized may require manipulation in order to bring it to the form required in the model. The first part of this chapter will deal explicitly with parameter manipulation. The rest of the chapter will be devoted to explaining the ramifications: indexed assignment functions, index operations.

6.2 The Assignment Statement

The assignment statement is the fundamental data manipulation statement in GAMS. It may be used to define or alter values associated with any sets, parameters, variables or equations.

A simple assignment is written in the style associated with many other computer languages. GAMS uses the traditional symbols for addition (+), subtraction (-), multiplication (*) and division (/). We will use them in the examples that follow, and give more details in Section 6.3, page 53.

6.2.1 Scalar Assignments

Consider the following artificial sequence:

```
scalar x / 1.5/;
x = 1.2;
x = x + 2;
```

The scalar \mathbf{x} is initialized to be 1.5. The second statement changes the value to 1.2, and the third changes it to 3.2. The second and third statement assignments have the effect of replacing the previous value of \mathbf{x} , if any, with a new one.

Note that the same symbol can be used on the left and right of the = sign. The new value is not available until the calculation is complete, and the operation gives the expected result.

An assignment cannot start with a reserved word. A semicolon is therefore required as a delimiter before all assignments.

6.2.2 Indexed Assignments

The syntax in GAMS for performing indexed assignments is extremely powerful. This operation offers what may be thought of as simultaneous or parallel assignment and it provides a concise way of specifying large amounts of data. Consider the mathematical statement, $DJ_d = 2.75DA_d$ for all d.

This means that for every member of the set d, a value is assigned to DJ. This can be written in GAMS as follows,

dj(d) = 2.75*da(d);

This assignment is known technically as an indexed assignment and set d will be referred to as the controlling index or controlling set.

The index sets on the left hand side of the assignment are together called the controlling domain of the assignment

The extension to two or more controlling indices should be obvious. There will be an assignment made for each label combination that can be constructed using the indices inside the parenthesis. Consider the following example of an assignment to all 100 data elements of **a**.

The calculation in the last statement is carried out for each of the 100 unique two-label combinations that can be formed from the elements of row and col. The first of these is, explicitly,

a('r-1','c-1') = 13.2 + r('r-1')*c('c-1').

6.2.3 Using Labels Explicitly in Assignments

It is often necessary to use labels explicitly in assignments. This can be done as discussed earlier with parameters - by using quotes around the label. Consider the following assignment,

a('r-7', 'c-4') = -2.36;

This statement assigns a constant value to one element of **a**. All other elements of **a** remain unchanged. Either single or double quotes can be used around the labels.

6.2.4 Assignments Over Subsets

In general, wherever a set name can occur in an indexed assignment, a subset (or even a label) can be used instead if you need to make the assignment over a subset instead of the whole domain.

Consider the following example,

a(sro,'col-10') = 2.44 -33*r(sro) ;

where sro has already been established to be a proper subset of row.

6.2.5 Issues with Controlling Indices

The number of controlling indices on the left of the = sign should be at least as many as the number of indices on the right. There should be no index on the right hand side of the assignment that is not present on the left unless it is operated on by an indexed operator

Consider the following statement,

a(row,'col-2') = 22 - c(col) 1;

GAMS will flag this statement as an error since col is an index on the right hand side of the equation but not on the left.

Each set is counted only once to determine the number of controlling indices. If the same set appears more than once within the controlling domain, the second and higher occurrences of the set should be **aliases** of the original set in order for the number of controlling indices to be equal to the number of indices.

Consider the following statement as an illustration,

b(row, row) = 7.7 - r(row);

This statement has only one controlling index (row). If one steps through the elements of row one at a time assignments will be made only to the diagonal entries in b. This will assign exactly 10 values! None of the off-diagonal elements of b will be filled.

If an additional name is provided for **row** and used in the second index position, then there will be two controlling indices and GAMS will make assignments over the full Cartesian product, all 100 values. Consider the following example,

alias(row,rowp) ; b(row,rowp) = 7.7 - r(row) + r(rowp) ;

6.2.6 Extended Range Identifiers in Assignments

The GAMS *extended range* identifiers can also be used in assignment statements, as in

a(row,'col-10') = inf ; a(row,'col-1') = -inf ;

Extended range arithmetic will be discussed later in this Section. The values most often used are NA in incomplete tables, and INF for variable bounds.

6.2.7 Acronyms in Assignments

Acronyms can also be used in assignment statements, as in

```
acronym monday, tuesday, wednesday, thursday, friday ;
parameter dayofweek ;
dayofweek = wednesday ;
```

Acronyms contain no numeric value, and are treated as character strings only.

6.3 Expressions

An expression is an arbitrarily complicated specification for a calculation, with parentheses nested as needed for clarity and intent. In this section, the discussion of parameter assignments will continue by showing in more detail the expressions that can be used on the right of the = sign. All numerical facilities available in both standard and *extended* arithmetic will be covered.

6.3.1 Standard Arithmetic Operations

The standard arithmetic symbols and operations are

****** exponentiation

- *,/ multiplication and division
- +,- addition and subtraction (unary and binary)

They are listed above in precedence order, which determines the order of evaluation in an expression without parentheses.

Consider, for example:

x = 5 + 4*3**2:

For clarity, this could have been written:

x = 5 + (4*(3**2));

In both cases the result is 41.

- It is better to use parentheses than to rely on the precedence of operators, since it prevents errors and clarifies intentions.
- Expressions may be freely continued over many lines: an end-of-line is permissible at any point where a blank may be used. Blanks may be used for readability around identifiers, parentheses and operator symbols. Blanks are not allowed within identifiers or numbers, and are significant inside the quote marks used to delimit labels.
- x**n is calculated inside GAMS as exp[n*log(x)]. This operation is not defined if x has a negative value, and an error will result. If the possibility of negative values for x is to be admitted and the exponent is known to be an integer, then a function call, power(x,n), is available.

Three additional capabilities are available to add power and flexibility of expression calculations. They are indexed operations, functions and extended range arithmetic.

6.3.2 Indexed Operations

In addition to the simple operations explained before, GAMS also provides the following four indexed operations.

sum	Summation over controlling index
prod	Product over controlling index

· 11•

· 1

- smin Minimum value over controlling index
- smax Maximum value over controlling index

These four operations are performed over one or more controlling indices. The syntax in GAMS for these operations is,

indexed_op((controlling_indices), expression)

If there is only one controlling index, the parentheses around it can be removed. The most common of these is sum, which is used to calculate totals over the domain of a set. Consider the following simple example adapted from [ANDEAN] for illustration.

```
sets i plants / cartagena, callao, moron /
    m product / nitr-acid, sulf-acid, amm-sulf /;
parameter capacity(i,m) capacity in tons per day
    totcap(m) total capacity by process ;
totcap(m) = sum(i, capacity(i,m));
```

This would be written, using normal mathematical representation, as $totC_m = \sum_i C_{im}$.

The index over which the summation is done, i, is separated from the reserved word sum by a left parenthesis and from the data term capacity(i,m) by a comma. i is again called the controlling index for this operation. The scope of the control is the pair of parentheses () that starts immediately after the sum. It is not likely to be useful to have two independent index operations controlled by the same index.

It is also possible to sum simultaneously over the domain of two or more sets, in which case more parentheses are needed. Also, of course, an arithmetic expression may be used instead of an identifier;

```
count = sum((i,j), a(i,j)) ;
emp = sum(t, l(t)*m(t)) ;
```

The equivalent mathematical forms are:

$$\texttt{count} = \sum_{i} \sum_{j} A_{ij} \text{ and } \texttt{emp} = \sum_{t} L_t M_t$$

The smin and smax operations are used to find the largest and smallest values over the domain of the index set or sets. The index for the smin and smax operators is specified in the same manner as in the index for the sum operator. Consider the following example to find the largest capacity,

```
lrgunit = smax((i,m),capacity(i,m));
```

6.3.3 Functions

Functions play an important part in the GAMS language, especially for non-linear models. Similar to other programming languages, GAMS provides a number of built-in (intrinsic) functions. However, GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS Function Library Facility (6.3.3) provides the means for managing that trade-off.

Intrinsic Functions

GAMS provides commonly used standard functions such as exponentiation, and logarithmic, and trigonometric functions. The complete list of available functions is given in table 6.1. There are cautions to be taken when functions appear in equations; these are dealt with in Section 8.4, page 74.

In table 6.1, the Endogenous Classification (second column) specifies in which models the function can legally appear with endogenous (non-constant) arguments. In order of least to most restrictive, the choices are any, NLP, DNLP or none (see Section 9.2.2 for details).

The following conventions are used for the function arguments. Lower case indicates that an endogenous variable is allowed. Upper case indicates that a constant argument is required. The arguments in square brackets can be omitted and default values will be used. Those default values are specified in the function description provided in the last column.

Function	Endogenous Classifica- tion	Description
Mathematical functions		
abs(x)	DNLP	returns the absolute value of an expression or term x
arccos(x)	NLP	returns the inverse cosine of the argument ${\bf x}$ where ${\bf x}$ is a real number between -1 and 1 and the output is in radians, see MathWorld

arcsin(x)	NLP	returns the inverse sine of the argument x where x is a real number between -1 and 1 and the output is in radians, see MathWorld
arctan(x)	NLP	returns the inverse tangent of the argument x where x is a real number and the output is in radians, see MathWorld
arctan2(y,x)	NLP	four-quadrant arctan function yielding $\operatorname{arctangent}(y/x)$ which is the angle the vector (x,y) makes with $(1,0)$ in radians
Beta(x,y)	DNLP	beta function: $B(x,y) = \frac{\gamma(x)\gamma(y)}{\gamma(x+y)}$, see MathWorld
betaReg(x,y,z)	NLP	regularized beta function, see MathWorld
binomial(n,k)	NLP	returns the (generalized) binomial coefficient for $n,k\geq 0$
ceil(x)	DNLP	returns the smallest integer number greater than or equal to x
centropy(x,y[,Z])	NLP	Centropy: $x \cdot \ln(\frac{x+Z}{y+Z})$, default setting: $Z = 0$
cos(x)	NLP	returns the cosine of the argument x where x must be in radi- ans, see MathWorld
cosh(x)	NLP	returns the hyperbolic cosine of x where x must be in radians, see MathWorld
cvPower(X,y)	NLP	returns X^y for $X \ge 0$, another possible command is 'X**y'
div(dividend,divisor)	NLP	returns $\frac{dividend}{divisor}$, undefined for $divisor = 0$
div0(dividend,divisor)	NLP	returns $\frac{dividend}{divisor}$, returns 10^{299} for $divisor = 0$
eDist(x1[,x2,x3,x4,x5,x6])	NLP	Euclidean or L-2 Norm: $\sqrt{x_1^2 + x_2^2 +}$, default setting: $x_2, x_3, x_4, x_5, x_6 = 0$
entropy(x)	NLP	Entropy: $-x \cdot \ln(x)$
errorf(x)	NLP	calculates the integral of the standard normal distribution from
		negative infinity to x, $errorf(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt$
execSeed	none	reads or writes the seed for the random number generator
exp(x)	NLP	returns the exponential function e^x of an expression or term x, see MathWorld
fact(X)	any	returns the factorial of X where X is an integer
floor(x)	DNLP	returns the greatest integer number less than or equal to x
frac(x)	DNLP	returns the fractional part of x
gamma(x)	DNLP	gamma function: $\gamma(x) = \int_{0}^{\infty} t^{x-1} e^{-t} dt$, see MathWorld
gammaReg(x,a)	NLP	regularized gamma function, see MathWorld
log(x)	NLP	returns the natural logarithm, logarithm base e, see Math-World
logBeta(x,y)	NLP	log beta function: $\log(B(x, y))$
logGamma(x)	NLP	log gamma function as discussed in MathWorld
log10(x)	NLP	returns the common logarithm, logarithm base 10, see Math-World
log2(x)	NLP	returns the binary logarithm, logarithm base 2, see MathWorld

mapVal(x)	none	 Function that returns an integer value associated with a numerical result that can contain special values. Possible values are: 0 for all regular numbers 4 for UNDF which means undefined 5 for NA which means not available 6 for INF which means plus infinity 7 for -INF which means minus infinity 8 for EPS which means very close to zero but different from zero
max(x1,x2,x3,)	DNLP	returns the maximum of a set of expressions or terms, the number of arguments is not limited
$\min(x1, x2, x3,)$	DNLP	returns the minimum of a set of expressions or terms, the number of arguments is not limited
mod(x,y)	DNLP	returns the remainder of x divided by y
ncpCM(x,y,Z)	NLP	function that computes a Chen-Mangasarian smoothing equaling: $x - Z \cdot \ln(1 + e^{\frac{x-y}{Z}})$
<pre>ncpF(x,y[,Z])</pre>	NLP	function that computes a Fisher smoothing equaling: $\sqrt{(x^2 + y^2 + 2 \cdot Z)} - x - y$, $Z \ge 0$, default setting: $Z = 0$
<pre>ncpVUpow(r,s[,MU])</pre>	NLP	NCP Veelken-Ulbrich: smoothed min ncpVUpow = $\begin{cases} \frac{(r+s- t)}{2} & \text{if } t \ge \mu \\ \frac{(r+s-\frac{\mu}{8}\cdot(-(\frac{t}{\mu})^4+6(\frac{t}{\mu})^2+3))}{2} & \text{otherwise} \end{cases}$ where $t = r - s$, default setting: $MU = 0$
<pre>ncpVUsin(r,s[,MU])</pre>	NLP	NCP Veelken-Ulbrich: smoothed min $ncpVUsin = \begin{cases} \frac{(r+s- t)}{2} & \text{if } t \ge \mu \\ \frac{(r+s-(\frac{2\mu}{\pi}\sin(\frac{\pi}{2\mu}+\frac{3\pi}{2})+\mu))}{2} & \text{otherwise} \end{cases}$ where $t = r - s$, default setting: $MU = 0$
normal(MEAN,STDDEV)	none	generates a random number with normal distribution with mean MEAN and standard deviation STDDEV, see Math-World
pi	any	value of $\pi = 3.141593$
poly(x,A0,A1,A2[,A3,A4])	NLP	computes a polynomial over scalar x where result = $A_0 + A_1x + A_2x^2 + A_3x^3 + \dots$ - this has a maximum of 6 arguments, default setting: $A_3, A_4 = 0$
power(x,Y)	NLP	returns x^Y where Y must be an integer, another possible command is 'x**Y'
randBinomial(N,P)	none	generates a random number with binomial distribution where n is the number of trials and p the probability of success for each trial, see MathWorld
randLinear(LOW,SLOPE,HIGH)	none	generates a random number between LOW and HIGH with linear distribution, SLOPE must be greater than $\frac{2}{\text{HIGH}-\text{LOW}}$
randTriangle(LOW,MID,HIGH)	none	generates a random number between LOW and HIGH with triangular distribution, MID is the most probable number, see MathWorld
round(x[,DECPL])	DNLP	rounding x, DECPL declares the number of decimal places, default setting: $DECPL = 0$
rPower(x,y)	NLP	returns x^y for $x, y \ge 0$, another possible command is 'x**y'
sigmoid(x)	NLP	Sigmoid calculation: $\frac{1}{1+e^{-x}}$, see MathWorld

sign(x)	DNLP	sign of x, returns 1 if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$
signPower(x,Y)	NLP	signed power, another possible command is $\operatorname{sign}(x)^* \operatorname{abs}(x)^{**} Y'$, where Y must be greater than 0
sin(x)	NLP	returns the sine of the argument x where x must be in radians, see MathWorld
sinh(x)	NLP	returns the hyperbolic sine of x where x must be in radians, see MathWorld
<pre>slexp(x[,SP])</pre>	NLP	smooth (linear) exponential function, SP means smoothing parameter, default setting: $SP=150$
<pre>sllog10(x[,SP])</pre>	NLP	smooth (linear) logarithm base 10, SP means smoothing parameter, default setting: $SP=10^{-150}$
<pre>slrec(x[,SP])</pre>	NLP	smooth (linear) reciprocal, SP means smoothing parameter, default setting: $SP = 10^{-10}$
<pre>sqexp(x[,SP])</pre>	NLP	smooth (quadratic) exponential function, SP means smoothing parameter, default setting: $SP=150$
<pre>sqlog10(x[,SP])</pre>	NLP	smooth (quadratic) logarithm base 10, SP means smoothing parameter, default setting: $SP = 10^{-150}$
sqr(x)	NLP	returns the square of an expression or term x
<pre>sqrec(x[,SP])</pre>	NLP	smooth (quadratic) reciprocal, SP means smoothing parameter, default setting: $SP = 10^{-10}$
sqrt(x)	NLP	returns the squareroot of x, see MathWorld
tan(x)	NLP	returns the tangent of the argument x where x must be in radians, see MathWorld
tanh(x)	NLP	returns the hyperbolic tangent of x where x must be in radians, see MathWorld
trunc(x)	DNLP	truncation, removes decimals from x
uniform(LOW,HIGH)	none	generates a random number between LOW and HIGH with uniform distribution, see $\underline{MathWorld}$
uniformInt(LOW,HIGH)	none	generates an integer random number between LOW and HIGH with uniform distribution, see $MathWorld$
vcPower(x,Y)	NLP	returns x^Y for $x \ge 0$, another possible command is 'x**Y'

Logical functions

0		
<pre>bool_and(x,y)</pre>	DNLP	boolean and: returns 0 if $x = 0 \lor y = 0$, else returns 1, another possible command is 'x and y'
<pre>bool_eqv(x,y)</pre>	DNLP	boolean equivalence: returns 0 if exactly one argument is 0, else returns 1, another possible command is 'x $eqv y$ '
<pre>bool_imp(x,y)</pre>	DNLP	boolean implication: returns 1 if $x = 0 \lor y \neq 0$, else returns 0, another possible command is 'x imp y'
bool_not(x)	DNLP	boolean not: returns 1 if $x = 0$, else returns 0, another possible command is 'not x'
<pre>bool_or(x,y)</pre>	DNLP	boolean or: returns 0 if $x = y = 0$, else returns 1, another possible command is 'x or y'
<pre>bool_xor(x,y)</pre>	DNLP	boolean xor: returns 1 if exactly one argument is 0, else returns 0, another possible command is 'x xor y'
ifThen(cond,iftrue,else)	DNLP	first argument contains a condition (e.g. $x > y$). If the condi- tion is true, the function returns iftrue else it returns else .

rel_eq(x,y)	DNLP	relation 'equal': returns 1 if $x = y$, else returns 0, another possible command is 'x eq y'
rel_ge(x,y)	DNLP	relation 'greater equal': returns 1 if $x \geq y,$ else returns 0, another possible command is 'x ge y'
rel_gt(x,y)	DNLP	relation 'greater than': returns 1 if $x > y$, else returns 0, another possible command is 'x gt y'
rel_le(x,y)	DNLP	relation 'less equal': returns 1 if $x \leq y,$ else returns 0, another possible command is 'x le y'
rel_lt(x,y)	DNLP	relation 'less than': returns 1 if $x < y$, else returns 0, another possible command is 'x lt y'
rel_ne(x,y)	DNLP	relation 'not equal': returns 1 if $x \neq y$, else returns 0, another possible command is 'x ne y'

Time and Calendar functions		
gday(SDAY)	any	returns Gregorian day from a serial day number date. time, where Jan 1, 1900 is day 1
gdow(SDAY)	any	returns Gregorian day of week from a serial day number date.time, where Jan 1, 1900 is day 1 $$
ghour(SDAY)	any	returns Gregorian hour of day from a serial day number date.time, where Jan 1, 1900 is day 1 $$
gleap(SDAY)	any	returns 1 if the year that corresponds to a serial day number date. time, where Jan 1, 1900 is day 1, is a leap year, else returns 0
gmillisec(SDAY)	any	returns Gregorian milli second from a serial day number date.time, where Jan 1, 1900 is day 1 $$
gminute(SDAY)	any	returns Gregorian minute of hour from a serial day number date. time, where Jan 1, 1900 is day 1
gmonth(SDAY)	any	returns Gregorian month from a serial day number date. time, where Jan 1, 1900 is day 1
gsecond(SDAY)	any	returns Gregorian second of minute from a serial day number date. time, where Jan 1, 1900 is day 1
gyear(SDAY)	any	returns Gregorian year from a serial day number date. time, where Jan 1, 1900 is day 1
jdate(YEAR,MONTH,DAY)	any	returns a serial day number, starting with Jan 1, 1900 as day 1
jnow	none	returns the current time as a serial day number, starting with Jan 1, 1900 as day 1
jstart	none	returns the time of the start of the GAMS job as a serial day number, starting with Jan 1, 1900 as day 1
jtime(HOUR,MIN,SEC)	any	returns fraction of a day that corresponds to hour, minute and second

GAMS utility and performance functions		
errorLevel	none	error code of the most recently used command
execError	none	number of execution errors, may either be read or assigned to
gamsRelease	none	returns the version number of the current GAMS release, for example 23.8
gamsVersion	none	returns the current gams version, for example 238

handleCollect(HANDLE)	none	 tests if the solve of the problem identified by the calling argument HANDLE is done and if so loads the solution into GAMS. In particular it returns: 0: if the model associated with HANDLE had not yet finished solution or could not be loaded 1: if the solution has been loaded
handleDelete(HANDLE)	none	 deletes the grid computing problem identified by the HANDLE calling argument and returns a numerical indicator of the status of the deletion as follows: 0: if the the model instance has been removed 1: if the argument HANDLE is not a legal handle 2: if the model instance is not known to the system 3: if the deletion of the model instance encountered errors A nonzero return indicates a failure in the deletion and causes an execution error.
handleStatus(HANDLE)	none	 tests if the solve of the problem identified by the calling argument HANDLE is done and if so loads the solution into a GDX file. A numerical indication of the result is returned as follows: 0: if a model associated with HANDLE is not known to the system 1: the model associated with HANDLE exists but the solution process is incomplete 2: the solution process has terminated and the solution is ready for retrieval 3: the solution process signaled completion but the solution cannot be retrieved An execution error is triggered if GAMS cannot retrieve the status of the handle.
handleSubmit(HANDLE)	none	 resubmits a previously created instance of the model identified by the HANDLE for solution. A numerical indication of the result is returned as follows: 0: the model instance has been resubmitted for solution 1: if the argument HANDLE is not a legal handle 2: if a model associated with the HANDLE is not known to the system 3: the completion signal could not be removed 4: the resubmit procedure could not be found 5: the resubmit process could not be started
heapFree	none	allocated memory which is no more in use but not freed yet
heapLimit	none	interrogates the current heap limit (maximum allowable mem- ory use) in Mb and allows it to be reset
heapSize	none	returns the current heap size in Mb
jobHandle	none	returns the Process ID (PID) of the last job started
jobKill(PID)	none	sends a kill signal to the running job with Process ID PID, the return value is one if this was succesful, otherwise it is zero
jobStatus(PID)	none	 checks for the status of the job with the Process ID PID, possible return values are: 0: error (input is not a valid PID or access is denied) 1: process is still running 2: process is finished with return code which could be accessed by errorlevel 3: process not running anymore or was never running, no return code available
-----------------------	--------	---
jobTerminate(PID)	none	sends an interrupt signal to the running job with Process ID PID, the return value is one if this was succesful, otherwise it is zero
licenseLevel	any	 returns an indicator of type of license: 0: demo license, limited to small models 1: full unlimited developer license 2: run time license, no new variables or equations can be introduced besides those inherited in a work file 3: application license, only works with a specific work file which is locked to the license file
licenseStatus	any	returns a non zero when a license error is incurred
maxExecError	none	maximum number of execution errors, may either be read or assigned to
<pre>sleep(SEC)</pre>	none	execution pauses for SEC seconds
timeClose	none	returns the model closing time
timeComp	none	returns the compilation time in seconds
timeElapsed	none	returns the elapsed time in seconds since the start of a GAMS run
timeExec	none	returns the execution time in seconds
timeStart	none	returns the model start time since last restart
	TT 1 1	

Table 6.1: GAMS functions

Consider the following example of a function used as an expression in an assignment statement,

x(j) = log(y(j));

which replaces the current value of \mathbf{x} with the natural logarithm of \mathbf{y} over the domain of the index set j.

Extrinsic Functions

Using the GAMS Function Library Facility, functions can be imported from an external library into a GAMS model. Apart from the import syntax, the imported functions can be used in the same way as intrinsic functions. In particular, they can be used in equation definitions. Some function libraries are included with the standard GAMS software distribution but GAMS users can also create their own libraries using an open programming interface. Simple examples in the programming languages C, Delphi and Fortran come with every GAMS system. Contact support@gams.com for detailed instructions.

Using Function Libraries

Function libraries are made available to a model using a compiler directive:

\$FuncLibIn <InternalLibName> <ExternalLibName>

Note that the Function Library Facility gives you complete control over naming so that potential name conflicts between libraries can be avoided. The <InternalLibName> will be used to refer to the library inside your model source code. The <ExternalLibName> is the one given the library when it was created. To access libraries included with your GAMS distribution you use the library's name with no path. GAMS will look for the library in a standard place within the GAMS installation. To access a library that is not part of the standard GAMS distribution the external name must include the absolute path of the library's location. When processing the \$FuncLibIn directive, GAMS will validate the library, make the included functions available for use, and add a table of the included functions to the listing file.

Before using individual functions you must declare them:

```
Function <InternalFuncName> /<InternalLibName>.<FuncName>/;
```

Note that the syntax is similar to that used for declaring Sets, Parameters, Variables and so forth and that the control over potential naming conflicts extends to the names of the individual functions. The <InternalFuncName> is the one that you will use in the rest of your model code. The <InternalLibName> is the one that you created with the \$FuncLibIn directive and <FuncName> is the name given the function when the library was created. Once functions have been declared with the Function statement they may be used exactly like intrinsic functions in the remainder of your model code.

Example

Here is an example that adds some concrete detail.

```
$eolcom //
$set SLASH \
$if %system.filesys% == UNIX $set SLASH /
$FuncLibIn trilib testlib_ml%SLASH%tridclib // Make the library available.
                                             // trilib is the internal name being created now.
                                             // tridclib is the external name.
                                             // With no path, GAMS will look for tridclib in
                                             // the GAMS system directory.
* Declare each of the functions that will be used.
* myCos, mySin and MyPi are names being declared now for use in this model.
* Cosine, Sine and Pi are the function names from the library.
* Note the use of the internal library name.
Function myCos /trilib.Cosine/
         mySin /trilib.Sine/
         myPi /trilib.Pi/;
scalar i, grad, rad, intrinsic;
for (i=1 to 360.
    intrinsic = cos(i/180*pi);
    grad = mycos(i,1);
    abort$round(abs(intrinsic-grad),4) 'cos', i, intrinsic, grad;
   rad = mvcos(i/180*pi):
    abort$round(abs(intrinsic-rad) ,4) 'cos', i, intrinsic, rad;);
variable x:
equation e;
e.. sqr(mysin(x)) + sqr(mycos(x)) =e= 1;
model m /e/;
x.lo = 0; x.l=3*mypi
solve m min x using nlp;
```

The following lines from the listing file describe the library loaded.

```
FUNCLIBIN trilib tridclib
Function Library trilib
Mod. Function Description
Type
NLP Cosine(x[,MODE]) Cosine: mode=0 (default) -> rad, mode=1 -> grad
NLP Sine(x[,MODE]) Sine : mode=0 (default) -> rad, mode=1 -> grad
any Pi Pi
```

A description of the libraries included in the GAMS sytem can be found in Appendix J.

Stateful Libraries

While GAMS initrinsic function are stateless, a user can implement stateful extrinsic functions, meaning that the extrinisc libraries can have some memory. This can be done in two ways:

- 1. Library initialization (see Piecewise Polynomial Library, appendix J.3): At initialization time, the function library reads some data to provide the necessary functions
- 2. Previous function calls (see Trigonometric Library, function setMode, appendix J.6): Function calls that alter the execution of successive function calls

The latter type of memory is problematic, since different parts of the GAMS system potentially use different instances of the function library. For example, if one sets SetMode(1) before the solve statement and one uses GAMS option solvelink<>5 (see solvelink), the solver runs in a separate process with a new instance of the the function library and therefore uses the default mode, which is 0. Even worse, if solvelink=0 is set, the GAMS process terminates in order to execute the solve statement and restarts a new GAMS process after the solve which again starts up with a fresh function library instance, so the function library's memory is lost also in this case. The GAMS Test Library model trilib04 demonstrates this problem.

6.3.4 Extended Range Arithmetic and Error Handling

GAMS uses an *extended range* arithmetic to handle missing data, the results of undefined operations, and the representation of bounds that solver systems regard as *infinite*. The special symbols are listed in table 6.2, with the brief explanation of the meaning of each.

Special symbol	mapval	Description
inf	6	Plus infinity. A very large positive number
-inf	7	Minus infinity. A very large negative number
na	5	Not available. Used for missing data. Any Operation that uses the value NA
undf	4	will produce the result NA Undefined. The result of an undefined or illegal operation. A user cannot directly set a value to INDE
eps	8	Very close to zero, but different from zero.

Table 6.2: Special symbols for extended arithmetic

GAMS has defined the results of all arithmetic operations and all function values using these special values.

The results can be inspected by running the model library problem [CRAZY]. As one would expect, 1+INF evaluates to INF, and 1-EPS to 1.

The mapval function should be used in comparisons involving extended range arithmetic. Only the extended range arithmetic shown in the table above give non-zero values for mapval. For example, mapval(a) takes a value of 6 if a is inf. All regular numbers result in a mapval of 0.

The following table shows a selection of results for exponentiation and division for a variety of input parameters.

Value Operations				
a	b	a**b	power(a,b)	a/b
2	2	4	4	1
-2	2	undf	4	-1
2	2.1	4.28	undf	.952
na	2.5	na	na	na
3	0	1	1	undf
inf	2	inf	inf	inf
2	inf	undf	undf	0

Table 6.3: Exponentiation and Division

Image of the should avoid creating or using numbers with absolute values larger than 1.0E20. If a number is too large, it may be treated by GAMS as undefined (UNDF), and all values derived from it in a model may be unusable. Always use INF (or −INF) explicitly for arbitrarily large numbers

When an attempted arithmetic operation is illegal or has undefined results because of the value of arguments (division by zero is the normal example), an error is reported and the result is set to undefined (UNDF).

From there on, UNDF is treated as a proper data value and does not trigger additional error messages.

GAMS will not solve a model if an error has been detected, but will terminate with an error condition.

It is thus always necessary to anticipate conditions that will cause errors, such as divide by zero. This is most easily done with the dollar control, and will be discussed in the next section.

6.4 Summary

GAMS provides powerful facilities for data manipulation with parallel assignment statements, built-in functions and extended range arithmetic.

7

Variables

7.1 Introduction

This chapter covers the declaration and manipulation of GAMS variables. Many of the concepts covered in the previous Chapters are directly applicable here.

A variable is the GAMS name for what are called *endogenous variables* by economists, *columns* or *activities* by linear programming experts, and *decision variables* by industrial Operations Research practitioners. They are the entities whose values are generally unknown until after a model has been solved. A crucial difference between GAMS variables and columns in traditional mathematical programming terminology is that one GAMS variable is likely to be associated with many columns in the traditional formulation.

7.2 Variable Declarations

A GAMS variable, like all other identifiers, must be declared before it is referenced.

7.2.1 The Syntax

The declaration of a variable is similar to a set or parameter declaration, in that domain lists and explanatory text are allowed and recommended, and several variables can be declared in one statement.

[var-type] variable[s] var_name [text] {, var_name [text]}

Var_type is the optional variable type that is explained in detail later. Var_name is the internal name of the variable (also called an identifier) in GAMS. An identifier has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. The accompanying text is used to describe the set or element immediately preceding it. This must not exceed 254 characters and must all be contained on the same line as the identifier it describes.

One important difference between variable and parameter declarations is that values *cannot* be initialized in a variable declaration.

A typical variable statement, adapted from [RAMSEY], is shown below for illustration:

The declaration of k above implies, as usual, that references to k are restricted to the domain of the set t. A model that includes k will probably have several corresponding variables in the associated mathematical programming

problem: most likely one for each member of t. In this way, very large models can be constructed using a small number of variables. (It is quite unusual for a model to have as many as 50 distinct variables.) It is still unclear from the declaration whether utility is not domain checked or whether it is a scalar variable, i.e., one without associated sets. Later references will be used to settle the issue.

It is important that variable declarations include explanatory text and that this be as descriptive as possible, since the text is used to annotate the solution output. Note the use of 'per' instead of '/' in the text above: slashes are illegal in all unquoted text.

7.2.2 Variable Types

There are five basic types of variables that may be used in variable statements. These are shown in table 7.1.

Keyword	Default	Default	Description
	Lower	Upper	
	Bound	Bound	
free (default)	-inf	+inf	No bounds on variable. Both bounds can be changed from the
			default values by the user
positive	0	+inf	No negative values are allowed for variable. The user can change
			the upper bound from the default value.
negative	-inf	0	No positive values are allowed for variables. The user can change
			the lower bound from the default value.
binary	0	1	Discrete variable that can only take values of 0 or 1
integer	0	100	Discrete variable that can only take integer values between the
			bounds. The user can change bounds from the default value.

Table 7.1: Variable types and default bounds

The default type is **free**, which means that if the type of the variable is not specified, it will not be bounded at all. The most frequently used types are **free** and **positive**, for descriptions of variables for which negative values are meaningless, such as capacities, quantities or prices.

Four additional, although more exotic, variable types - sos1, sos2, semicont and semiint are available in GAMS. These are explained in Section 17.2.1.

7.2.3 Styles for Variable Declaration

Two styles are commonly used to declare variable types. The first is to list all variables with domain specifications and explanatory text as a group, and later to group them separately as to type. The example shown below is adapted from [MEXSS]. The default type is **free**, so **phi**, **phipsi**, etc. will be **free** variables in the example below. Note the use of variable names derived from the original mathematical representation.

```
variables
           "purchase of domestic materials (mill units per yr)"
   u(c,i)
   v(c.j)
           "imports
                                (mill tpy)"
   e(c,i)
           "exports
                                (mill tpy)"
           "total cost
                                (mill us$)"
   phi
   phipsi
           "raw material cost
                               (mill us$)"
positive variables u, v, e ;
```

The commas in the list of positive variables are required separators.

It is possible to declare an identifier more than once, but that the second and any subsequent declarations should only add new information that does not contradict what has already been entered.

The second popular way of declaring variables is to list them in groups by type. We rewrite the example above using this second method:

```
free variables
  phi "total cost (mill us$)"
  phipsi "raw material cost (mill us$)"

positive variables
  u(c,i) "purchase of domestic materials (mill units per yr)"
  v(c,j) "imports (mill typ)"
  e(c,i) "exports (mill typ)";
```

The choice between the two approaches is best based on clarity.

7.3 Variable Attributes

Another important difference between parameters and variables is that an additional set of keywords can be used to specify various attributes of variables. A GAMS parameter has one number associated with each unique label combination. A variable, on the other hand, has seven. They represent:

- .10 The lower bound for the variable. Set by the user either explicitly or through default values.
- .up The upper bound for the variable. Set by the user either explicitly or through default values.
- .fx The fixed value for the variable.
- .1 The activity level for the variable. This is also equivalent to the current value of the variable. Receives new values when a model is solved.
- .m The marginal value (also called dual value) for the variable. Receives new values when a model is solved.
- .scale This is the scaling factor on the variable. This is normally an issue with nonlinear programming problems and is discussed in detail in Section 17.3.
- .prior This is the branching priority value of a variable. This parameter is used in mixed integer programming models only, and is discussed in detail later.

The user distinguishes between these suffix numbers when necessary by appending a suffix to the variable name.

7.3.1 Bounds on Variables

All default bounds set at declaration time can be changed using assignment statements.

For binary and integer variable types, the consequences of the type declaration cannot be completely undone.

Bounds on variables are the responsibility of the user. After variables have been declared, default bounds have already been assigned: for many purposes, especially in linear models, the default bounds are sufficient. In nonlinear models, on the other hand, bounds play a far more important role. It may be necessary to provide bounds to prevent undefined operations, such as division by zero.

It is also often necessary to define a '*reasonable*' solution space that will help to make the nonlinear programming problem be solved more efficiently.

The lower bound cannot be greater than the upper: if you happen to impose such a condition, GAMS will exit with an error condition.

7.3.2 Fixing Variables

GAMS allows the user to set variables through the .fx variable suffix. This is equivalent to the lower bound and upper bound being equal to the fixed value. Fixed variables can subsequently be freed by changing the lower and upper bounds.

7.3.3 Activity Levels of Variables

GAMS allows the user to fix the activity levels of variables through the .1 variable suffix. These activity levels of the variables prior to the solve statement serve as initial values for the solver. This is particularly important for nonlinear programming problems.

7.4 Variables in Display and Assignment Statements

GAMS allows the modeler to use the values associated with the various attributes of each variable in assignment and display statements. The next two sub-sections explain the use of variables in the left and right hand sides of assignment statements respectively. Later we will explain the use of variables in display statements.

7.4.1 Assigning Values to Variable Attributes

Assignment statements operate on one variable attribute at a time, and require the suffix to specify which attribute is being used. Any index list comes after the suffix.

The following example illustrates the use of assignment statements to set upper bounds for variables.

```
x.up(c,i,j) = 1000 ; phi.lo = inf ;
p.fx('pellets', 'ahmsa', 'mexico-df') = 200 ;
c.l(t) = 4*cinit(t) ;
```

Note that, in the first statement, the index set covering the domain of x appears after the suffix. The first assignment puts an upper bound on all variables associated with the identifier x. The statement on the second line bounds one particular entry. The statement on the last line sets the level values of the variables in c to four times the values in the parameter cinit.

Remember that the order is important in assignments, and notice that the two pairs of statements below produce very different results. In the first case, the lower bound for c('1985') will be 0.01, but in the second, the lower bound is 1.

c.fx('1985') = 1; c.lo(t) = 0.01; c.lo(t) = 0.01; c.fx('1985')=1;

Everything works as described in the previous chapter, including the various mechanisms described there of indexed operations, dollar operations, subset assignments and so on.

7.4.2 Variable Attributes in Assignments

Using variable attributes on the right hand side of assignment statements is important for a variety of reasons. Two common uses are for generating reports, and for generating initial values for some variables based on the values of other variables.

The following examples, adapted from [CHENERY], illustrate the use of variable attributes on the right hand side of assignment statements:

```
scalar  cva "total value added at current prices"
    rva "real value added"
    cli "cost of living index" ;
cva = sum (i, v.l(i)*x.l(i)) ;
cli = sum(i, p.l(i)*ynot(i))/sum(i, ynot(i)) ;
rva = cva/cli ;
display cli, cva, rva ;
```

As with parameters, a **variable** must have some non-default data values associated with it before one can use it in a display statement or on the right hand side of an assignment statement. After a solve statement (to be discussed later) has been processed or if non-default values have been set with an assignment statement, this condition is satisfied.

The .fx suffix is really just a shorthand for .lo and .up and can therefore only be used only on the left-hand side of an assignment statement.

7.4.3 Displaying Variable Attributes

When variables are used in display statements you must specify which of the six value fields should be displayed. Appending the appropriate suffix to the variable name does this. As before, no domain specification can appear. As an example we show how to display the level of phi and the level and the marginal values of v from [MEXSS]:

display phi.l, v.l, v.m;

The output looks similar, except that (of course) the listing shows which of the values is being displayed. Because zeroes, and especially all zero rows or columns, are suppressed, the patterns seen in the level and marginal displays will be quite different, since non-zero marginal values are often associated with activity levels of zero.

```
(MEXSS, SEQ=15)
Mexico Steel - Small Static
Execution
        203 VARIABLE PHI.L
                                                   538.811 total cost
                                                           (mill us$)
        203 VARIABLE V.L
                                     imports
                                     (mill tpy)
                    ( ALL
                                0.000)
        203 VARIABLE V.M
                                     imports
                                     (mill tpy)
        mexico-df
                    monterrey
                                guadalaja
            7.018
                       18.822
                                    6.606
steel
```

We should mention here a clarification of our previous discussion of displays. It is actually the default values that are suppressed on display output. For parameters and variable levels, the default is zero, and so zero entries are not shown. For bounds, however, the defaults can be non-zero. The default value for the upper bound of a positive variable is +INF, and if above you also would display v.up, for example, you will see:

 203 VARIABLE	V.UP	import	s
		(mill	tpy)
(ALL	+INF)	

If any of the bounds have been changed from the default value, then only the entries for the changed elements will be shown. This sounds confusing, but since few users display bounds it has not proved troublesome in practice.

7.5 Summary

Remember that wherever a parameter can appear in a display or an assignment statement, a variable can also appear - provided that it is qualified with one of the four suffixes. The only places where a variable name can appear without a suffix is in a variable declaration, as shown here, or in an equation definition, which is discussed in the next chapter. 8

Equations

8.1 Introduction

Equations are the GAMS names for the symbolic algebraic relationships that will be used to generate the constraints in the model. As with variables, one GAMS equation will map into arbitrarily many individual constraints, depending on the membership of the defining sets.

8.2 Equation Declarations

A GAMS equation, like all identifiers, must be declared before it can be used.

8.2.1 The Syntax

The declaration of an equation is similar to a set or parameter declaration, in that domain lists and explanatory text are allowed and recommended, and several equations can be declared in one statement.

```
Equation[s] eqn_name [text] {, eqn_name [text]} ;
```

Eqn_name is the internal name of the equation (an identifier) in GAMS. An identifier has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. The accompanying text is used to describe the set or element immediately preceding it. This must not exceed 254 characters and must all be contained on the same line as the identifier it describes.

There are no modifying keywords as there are with variables, and no initializing data list as there may be with parameters or sets.

8.2.2 An Illustrative Example

The example is adapted from [PRODSCH] , an inventory and production management problem. The relevant set definitions are also shown.

sets q 'quarters' / summer,fall,winter,spring /
s 'shifts' / first,second /;
equations
 cost 'total cost definition'
 invb(q) 'inventory balance'
 sbal(q,s) 'shift employment balance' ;

The declaration of the first equation follows the keyword equations. This declaration begins with the name of the equation, in this case cost, and is followed by the text, namely 'Total cost definition'. The equation cost above is a scalar equation, which will produce at most one equation in the associated optimization problem.

By contrast, the equation sbal is declared over the sets q (4 members) and s (2 members), and is thus likely to produce eight individual equations, one for each unique combination of labels. The circumstances under which less than eight equations might be produced will be discussed in later chapters. It is certainly true, however, that no more than eight equations will be produced.

8.3 Equation Definitions

The definitions are the mathematical specification of the equations in the GAMS language. The next sub-section explain the syntax for an equation definition and this is followed by an illustrative example. The rest of this section is devoted to discussions about some of the key components of equation definitions.

8.3.1 The Syntax

The syntax in GAMS for defining an equation is as follows,

```
eqn_name(domain_list).. expression eqn_type expression ;
```

Eqn_name is the name of the equation as in the equation declaration. The two dots '..' are always required between the equation name and start of the algebra. The expressions in the equation definition can be of the forms discussed in the Chapters before, but can involve variables as well. Eqn_type refers to the symbol between the two expressions that form the equation, and can be of the following types,

- =e= Equality: rhs must equal lhs
- =g= Greater than: lhs must be greater than or equal to rhs
- =1= Less than: lhs must be less than or equal to rhs
- =n= No relationships enforced between lhs and rhs. This equation type is rarely used.
- =x= External equation. Only supported by selected solvers.
- =c= Conic constraint. Only supported by selected solvers.
- As with the assignment statement, equation definitions can be carried over as many lines of input as needed. Blanks can be inserted to improve readability, and expressions can be arbitrarily complicated.
- An equation, once defined, can not be altered or re-defined. If one needs to change the logic, a new equation with a new name will have to be defined. It is possible, however, to change the meaning of an equation by changing the data it uses, or by using exception handling mechanisms (dollar operations) built into the definition

8.3.2 An Illustrative Example

Consider the following example, adapted from [MEXSS]. The associated declarations are also included.

```
Variables phi, phipsi, philam, phipi, phieps ;
equations obj ;
obj..phi =e= phipsi + philam + phipi - phieps ;
```

Obj is the name of the equation being defined. The =e= symbol means that this is an equality. Any of the following forms of the equation are mathematically equivalent,

```
obj.. phipsi + philam + phipi - phieps =e= phi ;
obj.. phieps - phipsi =e= philam - phi + phipi ;
obj.. phi - phieps - phipsi - philam - phipi =e= 0 ;
obj.. 0 =e= phi - phieps - phipsi - philam - phipi ;
```

The arrangement of the terms in the equation is a matter of choice, but often a particular one is chosen because it makes the model easier to understand.

8.3.3 Scalar Equations

A scalar equation will produce at most one equation in the associated optimization problem. The equation defined in the last Section is an example of a scalar equation, which contains only scalar variables. Note that in general, scalar equations may contain indexed variables operated on by index operators. Consider the following example from [CHENERY]

dty.. td =e= sum(i, y(i)) ;

8.3.4 Indexed Equations

All the set references in scalar equations are within the scope of index operations - many references can therefore be included in one equation. However, GAMS allows for equations to be defined over a domain, thereby developing a compact representation for constraints. The index sets to the left of the '...' are called the *domain of definition* of the equation.

Domain checking ensures that the domain over which an equation is defined must be the set or a subset of the set over which the equation is declared.

Consider the following example of a singly indexed equation, meaning one that produces a separate constraint for each member of the driving (or controlling) set.

dg(t).. g(t) = e = mew(t) + xsi(t)*m(t);

As t has three members, three constraints will be generated, each one specifying separately for each member of t, the dependence of g on m. Mew and xsi are parameters: the data associated with them are used in building up the individual constraints. These data do not have to be known when the equation is defined, but do have to be when a model containing the equation is solved.

The extension to two or more index positions on the left of the '..' should be obvious. There will be one constraint generated for each label combination that can constructed using the indices inside the parenthesis. Here are two examples from [AIRCRAFT], a scheduling model.

bd(j,h).. b(j,h) =e= dd(j,h) - y(j,h) ;
yd(j,h).. y(j,h) =l= sum(i, p(i,j)*x(i,j)) ;

The domain of definition of both equations is the Cartesian product of j and h: constraints will be generated for every label pair that can be constructed from the membership of the two sets.

8.3.5 Using Labels Explicitly in Equations

It is often necessary to use labels explicitly in equations. This can be done as with parameters - by using quotes around the label. Consider the following example,

dz.. tz =e= y('jan') + y('feb') + y('mar') + y('apr') ;

8.4 Expressions in Equation Definitions

The arithmetic operators and functions that were described in Section 6.3, page 53, can be used inside equations as well.

8.4.1 Arithmetic Operators in Equation Definitions

All the mechanisms that may be used to evaluate expressions in assignments are also available in equations.

Consider the following example adapted from [CHENERY] showing parentheses and exponentiation,

```
dem(i) .. y(i) =e= ynot(i)*(pd*p(i))**thet(i) ;
```

8.4.2 Functions in Equation Definitions

Function references in equation definitions can be classified into two types based on the type of the arguments, *Exogenous arguments:* The arguments(s) are known. Parameters and variable attributes (for example, .1 and .m attributes) are used as arguments. The expression is evaluated once when the model is being set up, and all functions except the random distribution functions uniform and normal are allowed.

Endogenous arguments: The arguments are variables and therefore unknown. The function will be evaluated many times at intermediate points while the model is being solved.

I The occurrence of any function with endogenous arguments implies that the model is not linear.

It is forbidden to use the uniform and normal functions in an equation definition.

Functions with endogenous arguments can be further classified into types listed in table 8.1.

Type	Function	Derivative	Examples
Smooth	Continuous	Continuous	exp, sin, log
Non-Smooth	Continuous	Discontinuous	max, min, abs
Discontinuous	Discontinuous	Discontinuous	ceil, sign

Table 8.1: Classification of functions with endogenous arguments

Smooth functions can be used routinely in nonlinear models, but non-smooth ones may cause numerical problems and should be used only if unavoidable, and only in a special model type called dnlp. However, the use of the dnlp model type is strongly discouraged and the use of binary variables is recommended to model non-smooth functions. Discontinuous functions are not allowed at all with variable arguments.

A fuller discussion is given in Chapter 9, page 77. For convenience, all the available functions are classified in table 6.1.

8.4.3 Preventing Undefined Operations in Equations

Certain operations can be undefined at particular values for the arguments. For example, the log-function is undefined when the argument is 0. Division by 0 is another example. While this can easily be determined for exogenous functions and expressions, it is a lot more difficult when the operands are variables. The expression may be evaluated many times when the problem is being solved. One way of preventing an expression from becoming undefined at all intermediate points is by adding bounds to the variable concerned. Consider the following function reference from [RAMSEY], preceded by the bounding of the variables: c.lo(t) = 0.01; util .. utility =e= sum(t, beta(t)*log(c(t)));

The bounding on c(t) away from 0 prevents the log function from being undefined.

8.5 Data Handling Aspects of Equations

The previous section dealt with the algebraic nature of equations. This section deals with the other aspect of an equation - it also serves as data. As with variables, four data values are associated with each unique *label-tuple* (unique label combination) of every equation. In practice these are used mainly for reporting purposes after a solve, and so the discussion will be brief. The suffixes associated with the four values are .1, .m, .lo and .up, as with variables. They may be assigned values in assignments (this is rare), or referenced in expressions or displayed, which is more common, especially for the marginal, .m. The meanings of the attributes .lo, .l and .up will be described with respect to an individual constraint rather than the symbolic equation.

After a solution has been obtained, there is a value associated with the unknown terms on the left, and this is by definition .1. The meaning of .1o and .up are shown in table 8.2 in terms of the constant right-hand-side (rhs) and the variable left-hand-side (.1) for each of the equation types. The relationship between rhs and .1 is satisfied only if the constraint is feasible at the solution point.

Type	.lo	. up	. l
=e=	\mathbf{rhs}	\mathbf{rhs}	\mathbf{rhs}
=1=	-inf	\mathbf{rhs}	\mathbf{rhs}
=g=	\mathbf{rhs}	inf	\mathbf{rhs}
=n=	-inf	inf	any

Table 8.2: Subfield definitions for equations

The meaning of the marginal value (.m) in terms of the objective value is discussed in detail in most texts on mathematical programming. The crude but useful definition is that it is the amount by which the objective function would change if the equation level were moved one unit.

9

Model and Solve Statements

9.1 Introduction

This chapter brings together all the concepts discussed in previous chapters by explaining how to specify a model and solve it.

9.2 The Model Statement

The model statement is used to collect equations into groups and to label them so that they can be solved. The simplest form of the model statement uses the keyword all: the model consists of all equations declared before the model statement is entered. For most simple applications this is all you need to know about the model statement.

9.2.1 The Syntax

In general, the syntax in GAMS for a model declaration is:

Model_name is the internal name of the model (also called an identifier) in GAMS. The accompanying text is used to describe the set or element immediately preceding it. Eqn_name is the name of an equation that has been declared prior to the model statement.

As with all identifiers, model_name has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. Explanatory text must not exceed 80 characters and must all be contained on the same line as the identifier or label it describes.

An example of a model definition in GAMS is shown below.

Model transport "a transportation model" / all / ;

The model is called transport and the keyword all is a shorthand for all known (declared) equations.

Several models can be declared (and defined) in one model statement. This is useful when experimenting with different ways of writing a model, or if one has different models that draw on the same data. Consider the following example, adapted from [PROLOG], in which different groups of the equations are used in alternative versions of the problem. Three versions are solved – the linear, nonlinear, and 'expenditure' versions. The model statement to define all three is

```
"linear version"
                                       / cb,rc,dfl,bc,obj /
model nortonl
                                       / cb,rc,dfn,bc,obj /
      nortonn
                 "nolinear version"
       nortone
                 "expenditure version / cb,rc,dfe,bc,obj /
                                                             ;
```

where cb, rc, etc. are the names of the equations. We will describe below how to obtain the solution to each of the three models.

9.2.2 **Classification of Models**

Various types of problems can be solved with GAMS. The type of the model must be known before it is solved. The model types are briefly discussed in this section. GAMS checks that the model is in fact the type the user thinks it is, and issues explanatory error messages if it discovers a mismatch - for instance, that a supposedly linear model contains nonlinear terms. This is because some problems can be solved in more than one way, and the user has to choose which way to go. For instance, if there are binary or integer variables in the model, it can be solved either as a MIP or as a RMIP.

The problem types and their identifiers, which are needed in the a solve statement, are listed below.

LP	Linear programming. There are no nonlinear terms or discrete (binary or integer) variables in your model.
QCP	Quadratic constraint programming. There are linear and quadratic terms but no general nonlinear term or discrete (binary or integer) variables in your model.
NLP	Nonlinear programming. There are general nonlinear terms involving only $smooth$ functions in the model, but no discrete variables. The functions were classified as to smoothness in the previous chapter.
DNLP	Nonlinear programming with discontinuous derivatives. This is the same as NLP, except that <i>non-smooth</i> functions can appear as well. These are more difficult to solve than normal NLP problems. The user is strongly recommended not to use this model type.
RMIP	Relaxed mixed integer programming. The model can contain discrete variables but the discrete requirements are relaxed, meaning that the integer and binary variables can assume any values between their bounds.
MIP	Mixed integer programming. Like RMIP but the discrete requirements are enforced: the discrete variables must assume integer values between their bounds.
RMIQCP	Relaxed mixed integer quadratic constraint programming. The model can contain both discrete variables and quadratic terms. The discrete requirements are relaxed. This class of problem is the same as QCP in terms of difficulty of solution.
RMINLP	Relaxed mixed integer nonlinear programming. The model can contain both discrete variables and general nonlinear terms. The discrete requirements are relaxed. This class of problem is the same as NLP in terms of difficulty of solution.
MIQCP	Mixed integer quadratic constraint programming. Characteristics are the same as for RMIQCP, but the discrete requirements are enforced.
MINLP	Mixed integer nonlinear programming. Characteristics are the same as for RMINLP, but the discrete requirements are enforced.
RMPEC	Relaxed Mathematical Programs with Equilibrium Constraints.
MPEC	Mathematical Programs with Equilibrium Constraints.
MCP	Mixed Complementarity Problem.
CNS	Constrained Nonlinear System.
EMP	Extended Mathematical Program.

Each of these model types will be discussed in detail in later chapters.

9.2.3 Model Attributes

Model attributes can be accessed through

model_name.attribute

Some of the attributes are mainly used before the solve statement to provide information to GAMS or the solver link. Others are set by GAMS or the solver link and hence are mainly used after a solve statement. Moreover, some of the input attributes can also be set globally via an option statement or the command line, e.g.

option reslim = 10 gamsmodel.gms reslim = 10

Note that a model specific option takes presedence over the global setting and that a setting via an option statement takes presedence over via the command line parameter.

The complete list of model attributes is shown below. The third and fourth column indicate whether there is also a global option and/or a command line parameter.

Attribute	Description	option	command
			line
bRatio	Basis acceptance test. A bratio of 0 accepts any basis, and a bratio of 1 always rejects the basis.	x	x
cheat	Cheat value. Requires a new integer solution to be a given amount better than the current best integer solution. Default value is 0.		
cutOff	Cutoff value. Occasionally used attribute that causes MIP solvers to delete parts of the branch and bound tree with an objective worse than the numerical value of the cutoff attribute.		
dictFile	Force writing of a dictionary file if $dictfile > 0$.		
domLim	Maximum number of domain errors. This attribute allows errors to occur up to a given number during solution. Default value is 0.	x	х
holdFixed	 This attribute can reduce the problem size by treating fixed variables as constants. Allowable values are: 0: fixed variables are not treated as constants (default) 1: fixed variables are treated as constants 		х
integer1	Integer communication cell that can contain any integer number.	x	x
integer2	Integer communication cell that can contain any integer number.	x	x
integer3	Integer communication cell that can contain any integer number.	x	x
integer4	Integer communication cell that can contain any integer number.	x	x
integer5	Integer communication cell that can contain any integer number.	x	x
iterLim	Iteration limit. The solvers will interrupt the solution process when the iteration count reaches that limit. The default value is 2 billion.	x	х
limCol	Maximum number of cases written to the LST file for each named variable in a model. The default value is 3.	х	х
limRow	Maximum number of cases written to the LST file for each named equation in a model. The default value is 3.	x	х
nodLim	Node limit. This attribute specifies the maximum number of nodes to process in the branch and bound tree for a MIP problem. The default value is 0 and is interpreted as 'not set'.	x	X

Model Attributes mainly used before solve

optCA	Absolute optimality criterion. This attribute specifies an absolute termi- nation tolerance for use in solving all mixed-integer models. The default value is 0.	x	х
optCR	Relative optimality criterion. This attribute specifies a relative termina- tion tolerance for use in solving all mixed-integer models. The default value is 0.1.	х	Х
optFile	 Look for a solver options file if optFile > 0. The value of optfile determines which option file is used: If Optfile = 1 the option file solvername.opt will be used. If Optfile = 2 the option file solvername.op2 will be used. If Optfile = 3 the option file solvername.op3 will be used. If Optfile = 15 the option file solvername.o15 will be used. If Optfile = 222 the option file solvername.222 will be used. If Optfile = 1234 the option file solvername.1234 will be used. If Optfile = 0 no option file will be used. (default) 		х
priorOpt	Priority option. Variables in mixed integer programs can have a priority attribute. One can use this parameter to specify an order for picking variables to branch on during a branch and bound search for MIP model solutions. The default value is 0 in which case priorities will not be used.		
real1	Real communication cell that can contain any real number.	x	
real2	Real communication cell that can contain any real number.	x	
real3	Real communication cell that can contain any real number.	x	
real4	Real communication cell that can contain any real number.	x	
real5	Real communication cell that can contain any real number.	x	
reform	Reformulation level.	x	
resLim	Maximum time available to solve in seconds. The default value is 1000.	x	х
savePoint	 This parameter tells GAMS to save a point format GDX file that contains the information on the current solution point. One can save the solution information from the last solve or from every solve. Numeric input with the following values is expected: 0: no point gdx file is to be saved 1: a point gdx file called model_name_p.gdx is to be saved from the last solve in the GAMS model 2: a point gdx file called model_name_pnn.gdx is to be saved from every solve in the GAMS model, where nn is the solve number as determined internally by GAMS 	x	x
scaleOpt	This attribute tells GAMS whether to employ user specified variable and equation scaling factors. It must be set to a nonzero value if the scaling factors are to be used.	х	х
solPrint	 This attribute controls the printing of the model solution to the LST file. Note that the corresponding option expects a text, while the use of model_name.solPrint and the command line expect a numeric value. Allowed are: 0/Off: remove solution listings following solves 1/On: include solution listings following solves 2/Silent: suppress all solution information The default value is 1 respectively 'On'. 	x	x

solveLink	 This attribute controls GAMS function when linking to solve. Allowable values are: 0: GAMS operates as always (default) 1: the solver is called from a shell and GAMS remains open 2: the solver is called with a spawn (if possible as determined by GAMS) or a shell (if the spawn is not possible) and GAMS remains open 3: GAMS starts the solution and continues in a Grid computing environment 4: GAMS starts the solution and waits (same submission process as 3) in a Grid computing environment 5: the problem is passed to the solver in core without use of temporary files. 	x	x
solveOpt	 This attribute tells GAMS how to manage the model solution when only part of the variables and equations are in the particular problem being solved. Note that the corresponding option expects a text, while the use of model_name.solveOpt and the command line expect a numeric value. Allowed are: 0/replace: the solution information for all equations appearing in the model is completely replaced by the new model results; variables are only replaced if they appear in the final model 1/merge: the solution information for all equations and variables is merged into the existing solution information; (default) 2/clear: the solution information for all equations appearing in the model is completely replaced; in addition, variables appearing in the symbolic equations but removed by conditionals will be removed 	x	
sysOut	This attribute controls the incorporation of additional solver generated output (that in the solver status file) to the LST file. Note that the cor- responding option expects a text, while the use of model_name.solPrint and the command line expect a numeric value. Allowed are: • 0/Off: suppress additional solver generated output (default) • 1/On: include additional solver generated output	x	х
threads	 This attribute controls the number of threads or CPU cores to be used by a solver. Allowable values are: -n: number of cores to leave free for other tasks 0: use all available cores n: use n cores (will be reduced to the available number of cores if n is too large) 	x	x
tolInfeas	Infeasibility tolerance for an 'empty' row of the form $a 0*x = e= 0.0001$;. By default this tolerance is set 'close' to the machine precision.		
tolInfRep	This attribute sets the tolerance for infeasible solutions. The default value is 1.0e-6.		
tolProj	This attribute controls the tolerance for setting a variable to its bound if the difference between its level and the bound is less than tolProj. If a variable is set to its lower bound the difference to the upper bound is not checked any more. The default setting is 1e-8.		

tryInt	Signals the solver to make use of a partial or near-integer-feasible solution stored in current variable values to get a quick integer-feasible point. If or how tryint is used is solver-dependent.	
tryLinear	Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used. To activate use model_name.trylinear=1. Default value is 0. The procedure also checks to see if QCP, and DNLP models can be reduced to an LP; MIQCP and MINLP can be solved as an MIP; RMIQCP and RMINLP can be solved as an RMIP.	
workFactor	This attribute tells the solver how much workspace to allocate for problem solution relative to the GAMS estimate.	х
workSpace	This attribute tells the solver how much workspace in Megabytes to allo- cate for problem solution.	х

Table 9.1: Model Attributes mainly used before solve

Model Attributes mainly used after solve

Attribute	Description
domUsd	Number of domain violations.
etAlg	This attribute returns the elapsed time it took to execute the solve algorithm. The time does not include the time to generate the model, reading and writing of files etc. The time is expressed in seconds of wall-clock time.
etSolve	This attribute returns the elapsed time it took to execute a solve statement in total. This time includes the model generation time, time to read and write files, time to create the solution report and the time taken by the actual solve. The time is expressed in seconds of wall-clock time.
etSolver	This attribute returns the elapsed time taken by the solver only. This does not include the GAMS model generation time and time taken to report and load the solution back into the GAMS database. The time is expressed in seconds of wall-clock time.
handle	Every solve gets unique handle number that may be used by handlecollect, handlestatus or handledelete. See Appendix I.
iterUsd	Number of iterations used.
line	Line number of last solve of the corresponding model
linkUsed	Integer number that indicates the value of solveLink used for the last solve
modelStat	Model status. Range from 1 to 19. For details check page 98.
nodUsd	The number of nodes used by the MIP solver.
number	Model instance serial number. The first model solved gets number 1, the second number 2 etc. The user can also set a value and the next model solved will get value+1 as number.
numDepnd	Number of dependencies in a CNS model.
numDVar	Number of discrete variables.
numEqu	Number of equations.
numInfes	Number of infeasibilities.
numNLIns	Number of nonlinear instructions.
numNLNZ	Number of nonlinear nonzeros.
numNOpt	Number of nonoptimalities.
numNZ	Number of nonzero entries in the model coefficient matrix.

numRedef	Number of MCP redefinitions.
numVar	Number of variables.
numVarProj	Number of bound projections during model generation.
objEst	The estimate of the best possible solution for a mixed-integer model.
objVal	The objective function value.
procUsed	Integer number that indicates the used model type. Possible values are: 1: LP 2: MIP 3: RMIP 4: NLP 5: MCP 6: MPEC 7: RMPEC 8: CNS 9: DNLP 10: RMINLP 11: MINLP 12: QCP 13: MIQCP 14: RMIQCP 15: EMP
resGen	Time GAMS took to generate the model in CPU seconds.
resUsd	Time the solver used to solve the model in CPU seconds.
rObj	The objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish.
solveStat	This model attribute indicates the solver termination condition. Range from 1 to 13. For details check page 99.
sumInfes	Sum of infeasibilities.

Table 9.2: Model Attributes mainly used after solve

9.3 The Solve Statement

Once the model has been put together through the model statement, one can now attempt to solve it using the solve statement. On seeing this statement, GAMS calls one of the available solvers for the particular model type.

It is important to remember that GAMS itself does not solve your problem, but passes the problem definition to one of a number of separate solver programs.

The next few sub-sections discuss the solve statement in detail.

9.3.1 The Syntax

In general, the syntax in GAMS for a model declaration is:

solve model_name using model_type maximizing|minimizing var_name|; solve model_name maximizing|minimizing var_name using model_type; Model_name is the name of the model as defined by a model statement. Var_name is the name of the objective variable that is being optimized. Model_type is one of the model types described before. An example of a solve statement in GAMS is shown below.

Solve transport using lp minimizing cost ;

Solve and using are reserved words. Transport is the name of the model, lp is the model type, minimizing is the direction of optimization, and cost is the objective variable. The opposite of minimizing is maximizing, both reserved words. Note that an objective variable is used instead of an objective row or function

The objective variable must be scalar and type free, and must appear in the least one of the equations in the model.

The next two sub-sections will describe briefly below what happens when a solve statement is processed, and more details on how the resulting output is to be interpreted will be given in the next chapter. After that sequences of solve statements will be discussed. The final section will describe options that are important in controlling solve statements.

9.3.2 Requirements for a Valid Solve Statement

When GAMS encounters a solve statement, the following are verified:

- 1. All symbolic equations have been defined and the objective variable is used in at least one of the equations
- 2. The objective variable is scalar and of type free
- 3. Each equation fits into the specified problem class (linearity for lp, continuous derivatives for nlp, as we outlined above)
- 4. All sets and parameters in the equations have values assigned.

9.3.3 Actions Triggered by the Solve Statement

The solve statement triggers a sequence of steps that are as follows:

- 1. The model is translated into the representation required by the solution system to be used.
- 2. Debugging and comprehension aids are produced and written to the output file (EQUATION LISTING, etc).
- 3. GAMS verifies that there are no inconsistent bounds or unacceptable values (for example NA or UNDF) in the problem.
- 4. Any errors detected at this stage cause termination with as much explanation as possible, using the GAMS names for the identifiers causing the trouble.
- 5. GAMS passes control to the solution subsystem and waits while the problem is solved.
- 6. GAMS reports on the status of the solution process and loads solution values back into the GAMS database. This causes new values to be assigned to the .1 and .m fields for all individual equations and variables in the model. A row by row and column by column listing of the solution is provided by default. Any apparent difficulty with the solution process will cause explanatory messages to be displayed. Errors caused by forbidden nonlinear operations are reported at this stage.

The outputs from these steps, including any possible error messages, are discussed in detail in the next chapter.

9.4 Programs with Several Solve Statements

Several solve statements can be processed in the same program. If you have to solve sequences of expensive or difficult models, you should read the chapter on workfiles to find out how to interrupt and continue program execution. The next few sub-sections discuss various instances where several solve statements may be needed in the same file.

9.4.1 Several Models

If there are different models then the solves may be sequential, as below. Each of the models in [PROLOG] consists of a different set of equations, but the data are identical, so the three solves appear in sequence with no intervening assignments:

solve nortonl using nlp maximizing z; solve nortonn using nlp maximizing z; solve nortone using nlp maximizing z;

When there is more than one **solve** statement in your program, GAMS uses as much information as possible form the previous solution to provide a starting point in the search for the next solution.

9.4.2 Sensitivity or Scenario Analysis

Multiple solve statements can be used not only to solve different models, but also to conduct sensitivity tests, or to perform case (or scenario) analysis of models by changing data or bounds and then solving the same model again. While some commercial LP systems allow access to sensitivity analysis through GAMS, it is possible to be far more general and not restrict the analysis to either solver or model type. This facility is even more useful for studying many scenarios since no commercial solver will provide this information.

An example of sensitivity testing is in the simple oil-refining model [MARCO]. Because of pollution control, one of the key parameters in oil refinery models is an upper bound on the sulfur content of the fuel oil produced by the refinery. In this example, the upper bound on the sulfur content of the fuel oil produced in the refinery. In this example, the upper bound on the sulfur content of fuel oil was set at 3.5 percent in the original data for the problem. First the model is solved with this value. Next a slightly lower value of 3.4 percent is used and the model is solved again. Finally, the considerably higher value of 5 percent is used and the model is solved for the last time. After each solve, key solution values (the activity levels are associated with z, the process levels by process p and by crude oil type cr) are saved for later reporting. This is necessary because a following solve replaces any existing values. The complete sequence is :

```
parameter report(*,*,*) "process level report";
qs('upper','fuel-oil','sulfur') = 3.5 ;
solve oil using lp maximizing phi;
report(cr,p,'base') = z.l(cr,p) ;
report('sulfur','limit','base') = qs('upper','fuel-oil','sulfur');
qs ('upper','fuel-oil','sulfur') = 3.4 ;
solve oil using lp maximizing phi ;
report(cr,p,'one') = z.l(cr,p) ;
report('sulfur','limit','one') = qs ('upper','fuel-oil','sulfur');
qs('upper','fuel-oil','sulfur') = 5.0 ;
solve oil using lp maximizing phi ;
report(cr,p,'two') = z.l(cr,p) ;
report('sulfur','limit','two') = qs('upper','fuel-oil','sulfur');
display report ;
```

This example shows not only how simply sensitivity analysis can be done, but also how the associated multi-case reporting can be handled. The parameter qs is used to set the upper bound on the sulfur content in the fuel oil, and the value is retrieved for the report.

The output from the display is shown below. Notice that there is no production at all if the permissible sulfur content is lowered. The *case attributes* have been listed in the row SULFUR.LIMIT. The *wild card* domain is useful when generating reports: otherwise it would be necessary to provide special sets containing the labels used in the report. Any mistakes made in spelling labels used only in the report should be immediately apparent, and their effects should be limited to the report. Section 14.5.1, page 134, contains more detail on how to arrange reports in a variety of ways.

	205	PARAMETER	REPORT	PROCESS	LEVEL	REPORT
			BASE	ONE	T	<i>N</i> 0
MID-C	.A-DI	ST	89.718		3	5.139
MID-C	.N-RE	FORM	20.000		(5.772
MID-C	.CC-D	IST	7.805		:	3.057
W-TEX	.CC-G	AS-OIL			Į	5.902
W-TEX	.A-DI	ST			64	4.861
W-TEX	.N-RE	FORM			12	2.713
W-TEX	.CC-D	IST			4	4.735
W-TEX	.HYDR	0			28	3.733
SULFUR.I	LIMIT		3.500	3.400	Į	5.000

9.4.3 Iterative Implementation of Non-Standard Algorithms

Another use of multiple solve statements is to permit iterative solution of different blocks of equations, solution values from the first are used as data in the next. These decomposition methods are useful for certain classes of problems because the sub-problems being solved are smaller, and therefore more tractable. One of the most common examples of such a method is the Generalized Bender's Decomposition method.

An example of a problem that is solved in this way is an input-output system with endogenous prices, described in Henaff (1980)¹. The model consists of two groups of equations. The first group uses a given final demand vector to determine the output level in each sector. The second group uses some exogenous process and inputoutput data to compute sectoral price levels. Then the resulting prices are used to compute a new vector of final demands, and the two block of equations are solved again. This iterative procedure is repeated until satisfactory convergence is obtained. Henaff has used GAMS statements to perform this kind of calculation. The statements that solve the system for the first time and the next iteration are shown below:

```
model usaio / mb, output /;
model dualmodel / dual, totp /;
solve usaio using lp maximizing total ;
solve dualmodel using lp maximizing totprice;
pbar(ta) = (sum(ipd.l(i,ta))/4.);
d(i,t) = (db(i)*g(t))/(pd.l(i,t)/pbar(t)) ;
solve usaio using lp maximizing total;
solve dualmodel using lp maximizing totprice;
```

Mb is a set of material balance (input-output) equations, and output is a total output equation. Dual is a group of price equations, and totp is an equation that sums all the sectoral prices. The domestic prices pd used in the calculation of the average price pbar are divided by four because there are four sectors in this example. Also the .1 is appended to pd to indicate that this is the level of the variable in the solution of the model namely in dualmodel. Thus the iterative procedure uses solution values from one iteration to obtain parameter values for the next one. In particular, both pbar and pd are used to compute the demand d for the i-th product in time period t, d(i,t). Also, the base year demand db and the growth factor g are used in that calculation. Then when the new final demand vector d is calculated, the two blocks of equations are solved again.

¹Henaff, Patrick (1980). An Input-Output Model of the French Economy, Master's Thesis, Department of Economics, University of Maryland.

9.5 Making New Solvers Available with GAMS

This short section is to encourage those of you who have a favorite solver not available through GAMS. Linking a solver program with GAMS is a straightforward task, and we can provide documents that describe what is necessary and provide the source code that has been used for existing links. The benefits of a link with GAMS to the developer of a solver are several. They include:

- \succ Immediate access to a wide variety of test problems.
- > An easy way of making performance comparisons between solvers.
- > The guarantee that a user has not somehow provided an illegal input specification.
- > Elaborate documentation, particularly of input formats, is not needed.
- > Access to the existing community of GAMS users, for marketing or testing.

This completes the discussion of the model and solve statements. In the next chapter the various components of GAMS output are described in some detail.

10

GAMS Output

10.1 Introduction

The output from GAMS contains many aids for checking and comprehending a model. In this chapter the contents of the output file are discussed. Ways by which the amount of diagnostic output produced can be controlled will also be discussed, although complete lists of all these controls are not given until later. A small nonlinear model, [ALAN] by Alan S. Manne, is used to illustrate the output file, and list it piece by piece as we discuss the various components. The possibilities for extension to large models with voluminous output (which is when the diagnostics are really useful) should be apparent.

The output from a GAMS run is produced on one file, which can be read using any text editor. The default name of this output file depends on the operating system, but Appendix C describes how this default can be changed. The display statement, described in detail in Chapter 14, can be used to export information from the GAMS program to the listing file.

10.2 The Illustrative Model

[ALAN] is a portfolio selection model whose object is to choose a portfolio of investments whose expected return meets a target while minimizing the variance. We will discuss a simplified version of this model. The input file is listed for reference.

```
$Title A Quadratic Programming Model for Portfolio Analysis ALAN, SEQ=124a)
$onsymlist onsymxref onuellist onuelxref
$Ontext
This is a mini mean-variance portfolio selection problem described in
'GAMS/MINOS: Three examples' by Alan S. Manne, Department of Operations
Research, Stanford University, May 1986.
$Offtext
* This model has been modified for use in the documentation
Set i securities
                   /hardware, software, show-biz, t-bills/; alias (i,j);
Scalar target
                  target mean annual return on portfolio % /10/,
       lowyield
                 yield of lowest yielding security,
       highrisk variance of highest security risk
 Parameters mean(i) mean annual returns on individual securities (%)
     / hardware 8
       software
                 9
       show-biz 12
                 7 /
       t-bills
```

Table v(i,j) variance-covariance array (%-squared annual return)

```
hardware software show-biz t-bills
     hardware
                    4
                              3
                                        -1
                                                  0
      software
                    3
                              6
                                         1
                                                  0
      show-biz
                   -1
                              1
                                        10
                                                  0
      t-bills
                    0
                              0
                                         0
                                                  0;
lowyield = smin(i, mean(i)) ;
highrisk = smax(i, v(i,i)) ;
display lowyield, highrisk ;
Variables x(i)
                      fraction of portfolio invested in asset i
                     variance of portfolio
           variance
Positive Variable x;
                   fractions must add to 1.0
Equations fsum
           dmean
                   definition of mean return on portfolio
           dvar
                   definition of variance;
fsum...
           sum(i, x(i))
                                            =e=
                                                 1.0 ;
dmean..
           sum(i, mean(i)*x(i))
                                            =e=
                                                 target:
dvar..
           sum(i, x(i)*sum(j,v(i,j)*x(j))) =e=
                                                 variance;
Model portfolio / fsum, dmean, dvar / ;
Solve portfolio using nlp minimizing variance;
```

10.3 Compilation Output

This is the output produced during the initial check of the program, often referred to as compilation. It contains two or three parts: the echo print of the program, an explanation of any errors detected, and the maps. The next four sub-sections will discuss each of these in detail.

10.3.1 Echo Print of the Input File

The *Echo Print* of the program is always the first part of the output file. It is just a listing of the input with the lines numbers added. The **\$offlisting** directive would turn off the listing of the input file.

A Quadratic Programming Model for Portfolio Analysis (ALAN, SEQ=124a)

This is a mini mean-variance portfolio selection problem described in 'GAMS/MINOS: Three examples' by Alan S. Manne, Department of Operations Research, Stanford University, May 1986.

9 * This model has been modified for use in the documentation

Note that the first line number shown is 9. If the lines on the input are counted, it can be seen that this comment line shown above appears after 8 lines of dollar directives and comments.

The line starting **\$title** has caused text of the users choice to be put on the page header, replacing the default tile, which just announces GAMS. The following **\$-** directives are used to display more information in the output file and we be discussed. The text within the **\$ontext-\$offtext** pair is listed without line numbers, whereas comments starting with asterisks have line numbers shown. Line numbers always refer to the physical line number in your input file.

Sollar control directives are only listed if a directive to list them is enabled, or if they contain errors.

Here is the rest of the echo print:

```
10
11 Set i securities /hardware,software,show-biz,t-bills/; alias (i,j);
```

```
12
13
     Scalar target
                       target mean annual return on portfolio % /10/,
14
            lowyield
                       yield of lowest yielding security,
15
            highrisk
                       variance of highest security risk ;
16
17 Parameters mean(i) mean annual returns on individual securities (%)
18
19
          / hardware
                       8
20
            software
                       9
21
            show-biz
                      12
22
            t-bills
                       7 /
23
24
     Table v(i,j) variance-covariance array (%-squared annual return)
25
26
                      hardware software show-biz t-bills
27
28
           hardware
                          4
                                    3
                                              -1
                                                         0
                         3
                                                         0
29
                                    6
           software
                                               1
30
           show-biz
                         -1
                                    1
                                              10
                                                         0
                                                         0;
31
           t-bills
                         0
                                    0
                                               0
32
33
     lowyield = smin(i, mean(i)) ;
34
     highrisk = smax(i, v(i,i)) ;
35
     display lowyield, highrisk ;
36
37
     Variables x(i)
                            fraction of portfolio invested in asset i
38
                           variance of portfolio
                variance
39
40
     Positive Variable x;
41
42
     Equations fsum
                         fractions must add to 1.0
43
                dmean
                         definition of mean return on portfolio
44
                dvar
                         definition of variance:
45
46
                sum(i, x(i))
     fsum..
                                                  =e= 1.0 ;
47
     dmean..
                sum(i, mean(i)*x(i))
                                                  =e=
                                                      target;
48
                sum(i, x(i)*sum(j,v(i,j)*x(j))) =e= variance;
     dvar..
49
50
     Model portfolio / fsum, dmean, dvar / ;
51
     Solve portfolio using nlp minimizing variance;
52
```

That is the end of the echo of the input file. If errors had been detected, the explanatory messages would be found in this section of the listing file. All discussion of error messages have been grouped in the section 10.6, page 102.

10.3.2 The Symbol Reference Map

Symbol Listing

The maps are extremely useful if one is looking into a model written by someone else, or if trying to make some changes in one's own model after spending time away from it.

The first map is the *Symbol Cross Reference*, which lists the identifiers (symbols) from the model in alphabetical order, identifies them as to type, shows the line numbers where the symbols appear, and classifies each appearance. The symbol reference map can be turned on by entering a line containing **\$onsymxref** at the beginning of the program. The map that resulted from [ALAN] is shown.

SYMBOL TYPE REFERENCES DMEAN EQU DECLARED 43 DEFINED 47 IMPL-ASN 52 REF 50 DVAR. EQU DECLARED 44 DEFINED 48 IMPL-ASN 52 50 REF DECLARED FSUM EQU 42 DEFINED 46 TMPL-ASN 52

		REF	50				
HIGHRISK	PARAM	DECLARED	15	ASSIGNED	34	REF	35
I	SET	DECLARED	11	DEFINED	11	REF	11
		17	24	33	2*34	37	46
		2*47	2*48	CONTROL	33	34	46
		47	48				
J	SET	DECLARED	11	REF	24	2*48	
		CONTROL	48				
LOWYIELD	PARAM	DECLARED	14	ASSIGNED	33	REF	35
MEAN	PARAM	DECLARED	17	DEFINED	19	REF	33
		47					
PORTFOLIO	MODEL	DECLARED	50	DEFINED	50	IMPL-ASN	52
		REF	52				
TARGET	PARAM	DECLARED	13	DEFINED	13	REF	47
V	PARAM	DECLARED	24	DEFINED	24	REF	34
		48					
VARIANCE	VAR	DECLARED	38	IMPL-ASN	52	REF	48
		52					
Х	VAR	DECLARED	37	IMPL-ASN	52	REF	40
		46	47	2*48			

For each symbol, the name and type of the symbol are first provided. For example, the last symbol listed is X which is defined to be of type VAR. The complete list of data types are given in table 10.1.

Entry in symbol reference table	GAMS Data Type
EQU	equation
MODEL	model
PARAM	parameter
SET	set
VAR	variable

Table 10.1: List of GAMS data types

Then comes a list of references to the symbol, grouped by reference type and identified by the line number in the output file. The actual reference can then be found by referring to the echo print of the program, which has line numbers on it. In the case of the symbol X in the example above, the list of references as shown in the symbol reference map are as follows,

DECLARED	37			
IMPL-ASN	52			
REF	40	46	47	2*48

This means that X is declared on line 37, implicitly assigned through a solve statement on line 52, and referenced on lines 40, 46, and 47. The entry 2*48 means that there are two references to X on line 48 of the input file.

The complete list of reference types is given below.

- DECLARED This is where the identifier is declared as to type. This must be the first appearance of the identifier.
- DEFINED This is the line number where an initialization (a table or a data list between slashes) or symbolic definition (equation) starts for the symbol.
- ASSIGNED This is when values are replaced because the identifier appears on the left of an assignment statement.
- IMPL-ASN This is an *implicit assignment*: an equation or variable will be updated as a result of being referred to implicitly in a solve statement.
- CONTROL This refers to the use of a set as the driving index in an assignment, equation, loop or other indexed operation (sum, prod, smin or smax).
- **REF** This is a reference: the symbol has been referenced on the right of an assignment, in a **display**, in an **equation**, or in a **model** or **solve** statement.

10.3.3 The Symbol Listing Map

The next map is called the *Symbol Listing*. All identifiers are grouped alphabetically by type and listed with their explanatory texts. This is another very useful aid to have handy when first looking into a large model prepared by someone else. The symbol listing map can be turned on by entering a line containing **\$onsymlist** at the beginning of the program.

```
Symbol Listing
SETS
Ι
            securities
J
            Aliased with I
PARAMETERS
            variance of highest security risk
HIGHRISK
LOWYIELD
            yield of lowest yielding security
MEAN
            mean annual returns on individual securities (%)
TARGET
            target mean annual return on portfolio %
            variance-covariance array (%-squared annual return)
V
VARIABLES
VARIANCE
            variance of portfolio
            fraction of portfolio invested in asset i
Х
EQUATIONS
DMEAN
            definition of mean return on portfolio
DVAR
            definition of variance
FSUM
            fractions must add to 1.0
MODELS
PORTFOLIO
```

10.3.4 The Unique Element Listing - Map

The following map is called the *Unique Element Listing*. All unique elements are first grouped in entry order and then in sorted order with their explanatory texts. The unique element listing map can be turned on by entering a line containing **\$onuelxref** at the beginning of the program.

```
Unique Element Listing
Unique Elements in Entry Order
    1 hardware
                   software
                                show-biz
                                             t-bills
Unique Elements in Sorted Order
                                             t-bills
    1 hardware
                   show-biz
                                software
ELEMENT
                    REFERENCES
hardware
             DECLARED
                              11
                                      REF
                                                 19
                                                          26
                                                                    28
show-biz
             DECLARED
                                      REF
                                                 21
                                                          26
                                                                    30
                              11
software
             DECLARED
                              11
                                      REF
                                                 20
                                                          26
                                                                    29
             DECLARED
                                      REF
                                                 22
                                                          26
t-bills
                              11
                                                                    31
```

10.3.5 Useful Dollar Control Directives

This sub-section reviews the most useful of the *Dollar Control Directives*. These must not be confused with the dollar exception-handling operators that will be introduced later: the similarity of terminology is unfortunate.

The dollar control directives are compiler directives that can be put in the input file to control the appearance and amount of detail in the output produced by the GAMS compiler.

\$offlisting, \$onlisting

This directive stops the echo print of the input file. **\$onlisting** restores the default.

\$offsymxref, \$offsymlist, \$onsymxref, \$onsymlist

These four directives are used to control the production of symbol maps. Maps are most often turned on or off at the beginning of the program and left as initially set, but it is possible to produce maps of part of the program by using a *on-map* directive followed later by an *off-map*. The symlist lists all the symbols in the model. The symref shows a complete cross-reference list of symbols by number. Both these maps are suppressed by default.

\$offuelxref, \$offuellist, \$onuelxref, \$onuellist

These four directives are used to control the production of Unique Element maps which show set membership labels. Maps are most often turned on or off at the beginning of the program and left as initially set, but it is possible to produce maps of part of the program by using a *on-map* directive followed later by an *off-map*. The uellist lists all labels in both GAMS entry and alphabetical order. The uelxref shows a complete cross-reference list by number. These label maps are suppressed by default.

\$offupper, \$onupper

This directive causes the echo print of the portion of the GAMS program following the directive to appear on the output file in the case that it has been entered in. This is the default on newer GAMS systems. It is necessary if case conventions have been used in the program, for example to distinguish between variables and equations. **\$onupper**, will cause all echo print to be in upper case.

\$ontext, \$offtext

\$ontext-\$offtext pairs are used to create *block comments* that are ignored by GAMS. Every **\$ontext** must have a matching **\$offtext** in the same file. The **\$offtext** must be on a line by itself.

\$title 'text'

The text can have up to 80 characters. This causes every page of the output to have the title specified.

- 🖙 In all dollar control directives, the \$ symbol must be in the first character position on the line.
- Dollar control directives, are dynamic: they affect only what happens after they are encountered, and they can be set and reset wherever appropriate.

They are remembered in *continued compilations* started from work files The directives that do not have following text can be entered many to a line, as shown below for the map controls.

10.4 Execution Output

The only output to the listing file while GAMS is executing (performing data manipulations) is from the display statement. All the user controls available to change the format will be discussed in detail later. The output from the display statement on line 41 of the example is shown below. Note the wrap of the explanatory text.

 32	PARAMETER	LOWYIELD	=	7.000	yield of	lowest
					yielding	security
	PARAMETER	HIGHRISK	=	10.000	variance	of highest
					security	risk

If errors are detected because of illegal data operations, a brief message indicating the cause and the line number of the offending statement will appear.

10.5 Output Produced by a Solve Statement

In this section, the content of the output produced when a solve statement is executed will be explained. In Chapter 9 all the actions that are triggered by a solve were listed. All output produced as a result of a solve is labeled with a subtitle identifying the model, its type, and the line number of the solve statement.

10.5.1 The Equation Listing

The first output is the *Equation Listing*, which is marked with that subtitle on the output file. By default, the first three equations in every block are listed. If there are three or fewer single equations in any equation block, then all the single equations are listed. The equation listing section from the example is listed below. This model has three equation blocks, each producing one single equation.

```
A Quadratic Programming Model for Portfolio Analysis (ALAN, SEQ=124a)
                    SOLVE PORTFOLIO USING NLP FROM LINE 48
Equation Listing
---- FSUM
                 =E= fractions must add to 1.0
FSUM...
       X(hardware) + X(software) + X(show-biz) + X(t-bills) =E= 1 ;
      (LHS = 0, INFES = 1 ***)
---- DMEAN
                 =E= definition of mean return on portfolio
DMEAN.. 8*X(hardware) + 9*X(software) + 12*X(show-biz) + 7*X(t-bills) = E= 10
      ; (LHS = 0, INFES = 10 ***)
---- DVAR
                 =E= definition of variance
DVAR.. (0)*X(hardware) + (0)*X(software) + (0)*X(show-biz) - VARIANCE = E = 0 ;
      (LHS = 0)
```

The equation listing is an extremely useful debugging aid. It shows the variables that appear in each constraint, and what the individual coefficients and right-hand-side value evaluate to after the data manipulations have been done.

Most of the listing is self-explanatory. The name, text, and type of constraints are shown. The four dashes are useful for mechanical searching.

All the terms that depend on variables are collected on the left, and all the constant terms are combined into one number on the right, any necessary sign changes being made.

Four places of decimals are shown if necessary, but trailing zeroes following the decimal point are suppressed. E-format is used to prevent small numbers being displayed as zero.

The nonlinear equations are treated differently. If the coefficient of a variable in the equation listing is enclosed in parentheses, then the corresponding constraint is nonlinear, and the value of the coefficient depends on the activity levels of one or more of the variables. The listing is not algebraic, but shows the partial derivative of each variable evaluated at their current level values.

Note that, in the equation listing from our example, the equation dvar is nonlinear. A simpler example will help to clarify the point. Consider the following equation and associated level values.

eq1.. 2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2; x.1 = 2; y.1 = 3 ;

then the equation listing will appear as

EQ1.. (221)*X + (216.1667)*Y =E= 2 ; (LHS = 225.5 ***)

The coefficient of x is determined by first differentiating the equation above with respect to x. This results in 2*(2*x.1)*power(y.1,3)+5, which evaluates to 221. Similarly the coefficient of y is obtained by differentiating the equation above with respect to y which results in 2*(sqr(x.1)*3*sqr(y.1) + 1.5/sqr(y.1)), giving 216.1667. Notice that the coefficient of y could not have been determined if its level had been left at zero. The attempted division by zero would have produced an error and premature termination.

The result of evaluating the left-hand-side of the equation at the initial point is shown at the end of each individual equation listing. In the example above it is 225.5, and the three asterisks (***) are a warning that the constraint is infeasible at the starting point.

The order in which the equations are listed depends on how the model was defined. If it was defined with a list of equation names, then the listing will be in the order in that list. If it was defined as /all/, then the list will be in the order of declaration of the equations. The order of the entries for the individual constraints is determined by the label entry order.

10.5.2 The Column Listing

The next section of the listing file is the *Column Listing*. This is a list of the individual coefficients sorted by column rather than by row. Once again the default is to show the first three entries for each variable, along with their bound and level values. The format for the coefficients is exactly as in the equation listing, with the nonlinear ones enclosed in parentheses and the trailing zeroes dropped. The column listing section from our example follows.

A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a) Column Listing SOLVE PORTFOLIO USING NLP FROM LINE 48 ---- X fraction of portfolio invested in asset I X(hardware) (.LO, .L, .UP = 0, 0, +INF) 1 FSUM 8 DMEAN (0) DVAR

(.L0, .L, .UP = 0, 0, +INF)1 FSUM DMEAN 9 (0) DVAR X(show-biz) (.LO, .L, .UP = 0, 0, +INF)FSUM 1 12 DMEAN (0) DVAR REMAINING ENTRY SKIPPED -- VARIANCE variance of portfolio VARIANCE

X(software)

(.LO, .L, .UP = -INF, 0, +INF) -1 DVAR

The order in which the variables appear is the order in which they were declared.

10.5.3 The Model Statistics

The final information generated while a model is being prepared for solution is the statistics block, shown below. Its most obvious use is to provide details on the size and nonlinearity of the model.
Model Statistics SOLVE PORTFOLIO USING NLP FROM LINE 48

BLOCKS OF FOUATIONS	3	SINGLE EQUATIO	INS	3
BLOCKS OF VARIABLES	2	SINGLE VARIABI	.ES	5
NON ZERO ELEMENTS	12	NON LINEAR N-2	1	3
DERIVATIVE POOL	10	CONSTANT POOL	1	0
CODE LENGTH	87			
GENERATION TIME	=	0.020 SECONDS	0.1 Mb	WAT-50-094

The BLOCK counts refer to GAMS equations and variables, the SINGLE counts to individual rows and columns in the problem generated. The NON ZERO ELEMENTS entry refers to the number of non-zero coefficients in the problem matrix.

There are four entries that provide additional information about nonlinear models. The NON LINEAR N-Z entry refers to the number of nonlinear matrix entries in the model.

All forms of nonlinearity do not have the same level of complexity. For example, x*y is a simpler form of nonlinearity than exp(x*y). So, even though both these terms count as 1 nonlinear entry in the matrix, additional information is required to provide the user with a feel for the complexity of the nonlinearity. GAMS provides the CODE LENGTH entry as a good yardstick for this purpose. There are two other entries - DERIVATIVE POOL and CONSTANT POOL that provide some more information about the nonlinearity. In general, the more nonlinear a problem is, the more difficult it is to solve.

The times that follow statistics are also useful. The GENERATION TIME is the time used since the syntax check finished. This includes the time spent in generating the model. The measurement units are given, and represent ordinary clock time on personal computers, or central processor usage (CPU) time on other machines.

10.5.4 The Solve Summary

This is the point (chronologically speaking) where the model is solved, and the next piece of output contains details about the solution process. It is divided into two parts, the first being common to all solvers, and the second being specific to a particular one.

The section of the solve summary that is common for all solvers is first discussed. The corresponding section for the example model is shown below.

S	0	I		V	Е			S	U	М	М	A	R	Y											
			м	∩т	ਮਿਸ		DI	יפר	r Fi	יזר	тп						ΩB	150	דדי	VF		v۸	рт	۸N	CE
			Т	YF	PE		N	LP.			10						DI	REC		ON		MI	NI	MI	ZE
			S	OI	LVEF	ł	M	INC)S	5							FR	OM	LI	NE		48			
			~	.																~ 11					
**	***	*	S	UL	LAFE	1 2	511	ATU	JS			1	NU	JRI	1AL	- (JOM.	PLE	STI	UN					
**	***	*	М	01	DEL	SI	'A'	ΓUS	3			2	LC	C	۱LI	LΥ	0P'	TIN	IAL						
**	***	*	0	B.	JECI	'IV	Έ	V	ALI	JE								2.	89	90					
_								_	_											_	_				
F	RES	SC	JU	RC	CΕU	JSI	G	Ε,	L.	EM.	IΤ					(0.0	20		1	10	00	.0	00	
]	T	EF	ł٨	T]	CON	CC	UU	NT	, 1	LII	MI	Г				5	5			10	0	00			
E	EV	AI	JU	A٦	LION	IE	ERI	ROI	RS							()					0			

The common part of the solve summary is shown above. It can be found mechanically by searching for four asterisks. The explanation for the information provided in this section follows.

MODEL PORTFOLIO

This provides the name of the model being solved.

TYPE NLP

This provides the model type of the model being solved.

SOLVER MINOS5

This provides the name of the solver used to solve the model.

OBJECTIVE VARIANCE

This provides the name of the objective variable being optimized

DIRECTION MINIMIZE

This provides the direction of optimization being performed.

**** SOLVER STATUS1 NORMAL COMPLETION**** MODEL STATUS2 LOCALLY OPTIMAL

These provide the solver status and model status for the problem, and are discussed in greater detail at the end of this subsection.

**** OBJECTIVE VALUE 2.8990

This provides the value of the objective function at the termination of the solve. If the Solver and Model have the right status, this value is the optimum value for the problem.

RESOURCE USAGE, LIMIT 0.109 1000.000

These two entries provide the amount of CPU time (in seconds) taken by the solver, as well as the upper limit allowed for the solver. The solver will stop as soon as the limit on time usage has been reached. The default limit on time usage is 1000 seconds. This limit can be changed by entering a line containing the statement option reslim = xx; in the program before the solve statement, where xx is the required limit on CPU time in seconds.

ITERATION COUNT, LIMIT 5 1000

These two entries provide the number of iterations used by the solver, as well as the upper limit allowed for the solver. The solver will stop as soon as this limit is reached. The default limit on iterations used is pratically infinity. This limit can be changed by entering a line containing the statement option iterlim = nn ; in the program before the solve statement, where nn is the required limit on the iterations used.

EVALUATION ERRORS 0 0

These two entries provide the number of numerical errors encountered by the solver, as well as the upper limit allowed for the solver. These errors result due to numerical problems like division by 0. This is suppressed for LP, RMIP, and MIP models since evaluation errors are not applicable for these model types. The default limit on evaluation errors used is 0. This limit can be changed by entering a line containing the statement option domlim = nn ; in the program before the solve statement, where nn is the required limit on the evaluation errors allowed.

The SOLVER STATUS and MODEL STATUS require special explanation. The status for the solver (the state of the program) and the model (what the solution looks like) are characterized, and a complete list of possible MODEL STATUS and SOLVER STATUS messages is given below.

Here is a list of possible MODEL STATUS messages:

1 OPTIMAL

This means that the solution is optimal. It only applies to linear problems or relaxed mixed integer problems (RMIP).

2 LOCALLY OPTIMAL

This message means that a local optimum has been found. This is the message to look for if the problem is nonlinear, since all we can guarantee for general nonlinear problems is a local optimum.

3 UNBOUNDED

This means that the solution is unbounded. This message is reliable if the problem is linear, but occasionally it appears for difficult nonlinear problems that are not truly unbounded, but that lack some strategically placed bounds to limit the variables to sensible values.

4 INFEASIBLE

This means that the linear problem is infeasible. Something is probably misspecified in the logic or the data.

5 LOCALLY INFEASIBLE

This message means that no feasible point could be found for the nonlinear problem from the given starting point. It does not necessarily mean that no feasible point exists.

6 INTERMEDIATE INFEASIBLE

This means that the current solution is not feasible, but that the solver program stopped, either because of a limit (iteration or resource), or because of some sort of difficulty. Check the solver status for more information.

7 INTERMEDIATE NONOPTIMAL

This is again an incomplete solution, but it appears to be feasible.

8 INTEGER SOLUTION

An integer solution has been found to a MIP (mixed integer problem). There is more detail following about whether this solution satisfies the termination criteria (set by options optcr or optca).

9 INTERMEDIATE NON-INTEGER

This is an incomplete solution to a MIP. An integer solution has not yet been found.

10 INTEGER INFEASIBLE

There is no integer solution to a MIP. This message should be reliable.

11 LIC PROBLEM - NO SOLUTION

The solver cannot find the appropriate license key needed to use a specific subsolver.

12 ERROR UNKNOWN

After a solver error, the model status is unknown.

13 ERROR NO SOLUTION

An error occurred and no solution has been returned. No solution will be returned to GAMS because of errors in the solution process.

14 NO SOLUTION RETURNED

A solution is not expected for this solve. For example, the convert solver only reformats the model but does not give a solution.

15 SOLVED UNIQUE

This indicates a unique solution to a CNS model.

16 SOLVED

A CNS model has been solved but multiple solutions may exist.

17 SOLVED SINGULAR

A CNS model has been solved but the point is singular.

18 UNBOUNDED - NO SOLUTION

The model is unbounded and no solution can be provided.

19 INFEASIBLE - NO SOLUTION

The model is infeasible and no solution can be provided.

This is the list of possible SOLVER STATUS messages:

1 NORMAL COMPLETION

This means that the solver terminated in a normal way: i.e., it was not interrupted by an iteration or resource limit or by internal difficulties. The model status describes the characteristics of the accompanying solution.

2 ITERATION INTERRUPT

This means that the solver was interrupted because it used too many iterations. Use option iterlim to increase the iteration limit if everything seems normal.

3 RESOURCE INTERRUPT

This means that the solver was interrupted because it used too much time. Use option **reslim** to increase the time limit if everything seems normal.

4 TERMINATED BY SOLVER

This means that the solver encountered difficulty and was unable to continue. More detail will appear following the message.

5 EVALUATION ERROR LIMIT

Too many evaluations of nonlinear terms at undefined values. You should use bounds to prevent forbidden operations, such as division by zero. The rows in which the errors occur are listed just before the solution.

6 CAPABILITY PROBLEMS

The solver does not have the capability required by the model, for example, BARON has a more limited set of functions than other solvers.

7 LICENSING PROBLEMS

The solver cannot find the appropriate license key needed to use a specific subsolver.

8 USER INTERRUPT

The user has sent a message to interrupt the solver via the interrupt button in the IDE or sending a control C from a command line.

9 ERROR SETUP FAILURE

The solver encountered a fatal failure during problem set-up time.

10 ERROR SOLVER FAILURE

The solver encountered a fatal error.

11 ERROR INTERNAL SOLVER FAILURE

The solver encountered an internal fatal error.

12 SOLVE PROCESSING SKIPPED

The entire solve step has been skipped. This happens if execution errors were encountered and the GAMS parameter ExeErr has been set to a nonzero value, or the property MaxExecError has a nonzero value.

13 ERROR SYSTEM FAILURE

This indicates a completely unknown or unexpected error condition.

10.5.5 Solver Report

The next section in the listing file is the part of the solve summary that is particular to the solver program that has been used. This section normally begins with a message identifying the solver and its authors: MINOS was used in the example here. There will also be diagnostic messages in plain language if anything unusual was detected, and specific performance details as well, some of them probably technical. The Solver Manual will help explain these. In case of serious trouble, the GAMS listing file will contain additional messages printed by the solver. This may help identify the cause of the difficulty. If the solver messages do not help, a perusal of the solver documentation or help from a more experienced user is recommended. The solver report from our example follows.

GAMS/MINOS
B. A. Murtagh, University of New South Wales and
P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright Systems Optimization Laboratory, Stanford University.
Work space allocated -- 0.04 Mb
EXIT -- OPTIMAL SOLUTION FOUND MAJOR ITNS, LIMIT 11 200

FUNOBJ, FUNCON CALLS	0	71
SUPERBASICS	4	
INTERPRETER USAGE	0.02	
NORM RG / NORM PI	1.801E-09	

The line 'work space allocated -- 0.04 MB' provides the amount of memory used by the solver for the problem. If the amount of memory the solver estimates that it needs is not available, GAMS will return a message saying that not enough memory was allocated. GAMS will also return the maximum amount of memory available on the machine. The user can direct the solver to use less memory by entering a line containing the statement mymodel.workspace = xx; were mymodel is the name of the model being solved as specified by the model statement, and xx is the amount of memory in Megabytes. Note that the solver will attempt to solve the problem with xx MB of memory, however it is not guaranteed to succeed since the problem may require more memory.

More information can be obtained for a successful run by entering a line containing the statement option sysout = on ; in the program above the solve statement.

10.5.6 The Solution Listing

The next section of the listing file is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed with four pieces of information.

This section of the listing file can be turned off by entering a line containing the statement option solprint = off ; in the program above the solve statement.

The solution listing section from our example is shown below.

		LOWER	LEVEL	UPPER	MARGINAL	
EQU EQU EQU	FSUM DMEAN DVAR	1.000 10.000	1.000 10.000	1.000 10.000	-13.529 1.933 -1.000	
FSUM DMEAN DVAR	fractic definit definit	ns must a ion of me ion of va	dd to 1.0 an return on riance	o portfolio	0	
VAR	X	fraction	of portfolic	invested	in asset	i
	LOWER	LEVEL	UPPER M	ARGINAL		
hardware	LOWER	LEVEL 0.303	UPPER M +INF	IARGINAL		
hardware software	LOWER	LEVEL 0.303 0.087	UPPER M +INF +INF	IARGINAL EPS		
hardware software show-biz	LOWER	LEVEL 0.303 0.087 0.505	UPPER M +INF +INF +INF	IARGINAL EPS		
hardware software show-biz t-bills	LOWER	LEVEL 0.303 0.087 0.505 0.106	UPPER M +INF +INF +INF +INF +INF	ARGINAL EPS EPS		
hardware software show-biz t-bills	LOWER	LEVEL 0.303 0.087 0.505 0.106 LOWER	UPPER M +INF +INF +INF +INF LEVEL	ARGINAL EPS EPS UPPER	MARGINAL	
hardware software show-biz t-bills	LOWER VARIANCE	LEVEL 0.303 0.087 0.505 0.106 LOWER -INF	UPPER M +INF +INF +INF +INF LEVEL 2.899	IARGINAL EPS EPS UPPER +INF	MARGINAL	

VARIANCE variance of portfolio

The order of the equations and variables are the same as in the symbol listing described before and will be described later

The four columns associated with each entry have the following meaning,

LOWER lower bound (.1o) LEVEL level value (.1) UPPER upper bound (.up) MARGINAL marginal (.m) For variables the values in the LOWER and UPPER columns refer to the lower and upper bounds. For equations they are obtained from the (constant) right-hand-side value and from the relational type of the equation. These relationships were described in Chapter 8.

The LEVEL and MARGINAL values have been determined by the solver, and the values shown are used to update the GAMS values. In the list they are shown with fixed precision, but the values are returned to GAMS with full machine accuracy. The single dots '.' on the list represent zero.

EPS is the GAMS extended value that means very close to but different from zero. It is common to see a marginal value given as EPS, since GAMS uses the convention that marginal are zero for basic variables, and not zero for others.

EPS is used with non-basic variables whose marginal values are very close to, or actually, zero, or in nonlinear problems with superbasic variables whose marginals are zero or very close to it. A superbasic variable is one between its bounds at the final point but not in the basis.

There are brief explanations of technical terms used in this section in the Glossary. For models that are not solved to optimality, some constraints may additionally be marked with certain flags. The list of these flags and their description is given below.

- **INFES** The row or column is infeasible. This mark is made for any entry whose level value is not between the upper and lower bounds.
- **NOPT** The row or column is non-optimal. This mark is made for any non-basic entries for which the marginal sign is incorrect, or superbasic ones for which the marginal value is too large.
- UNBND The row or column that appears to cause the problem to be unbounded.

10.5.7 Report Summary

The final section of the solution listing is the report summary, marked with four asterisks (as are all important components of the output). It shows the count of rows or columns that have been marked INFES, NOPT, or UNBND in the solution listing section. The sum of infeasibilities will be shown if it the reported solution is infeasible. The error count in is only shown if the problem is nonlinear.

****	REPORT	SUMMARY	:	0	NONOPT
				0	INFEASIBLE
				0	UNBOUNDED
				0	ERRORS

If our example had display output for reporting, it would come here.

10.5.8 File Summary

The last piece of the output file is important: it gives the names of the input and output disk files. If work files (save or restart) have been used, they will be named here as well.

**** FILE SUMMARY

INPUT C:\PROGRAM FILES\\gamsIDE\ALAN.GMS OUTPUT C:\PROGRAM FILES\\gamsIDE\ALAN.LST

10.6 Error Reporting

All the comments and description about errors have been collected into this section for easy reference when disaster strikes.

Effective error detection and recovery are important parts of any modeling system. GAMS is designed around the assumption that the *error State* is the normal state of modeling. Experience shows that most compilations during the early stages of development will produce errors. Not to Worry! The computer is much better at checking details that the human mind and should be able to provide positive feedback and suggestions about how to correct errors or avoid ambiguities. Developing a model is like writing a paper or an essay ; many drafts and rewrites are required until the arguments are presented in the most effective way for the reader and meet all the requirements of proper English. GAMS acts like a personal assistant with knowledge of mathematical modeling and of the syntactic and semantic details of the language.

Errors are detected at various stages in the modeling process. Most of them are caught at the compilation stage, which behaves like the proofreading stage of the modeling process. Once a problem has passed through the rigorous test of this stage, the error rate drops almost to zero. Most of the execution runs, which are much more expensive than compilation, proceed without difficulties because GAMS *knows* about modeling and has anticipated problems. Many of the typical errors made with conventional programming languages are associated with concepts that do not exist in GAMS. Those error sources – such as address calculations, storage assignment, subroutine linkages, input-output and flow control – create problems at execution time, are difficult to locate, often lead to long and frustrating searches, and leave the computer user intimidated. GAMS takes a radically different approach. Errors are spotted as early as possible, are reported in a way understandable to the user, including clear suggestions for how to correct the problem, and a presentation of the source of the error in terms of the user's problem.

All errors are marked with four asterisks '****' at the beginning of a line in the output listing.

As soon as an error is detected, processing will be stopped at the next convenient opportunity. A model will never be solved after an error has been detected. The only remedy is to fix the error and repeat the run.

Errors are grouped into the three phases of GAMS modeling! compilation, execution and model generation (which includes the solution that follows). The following three sub-sections discuss these types of errors.

10.6.1 Compilation Errors

Compilation errors were discussed in some detail in Chapter 2. There is some overlap between the material in those sections and this one. Several hundred different types of errors can be detected during compilation and can often be traced back to just one specific symbol in the GAMS input. Most of the errors will be caused by simple mistakes: forgetting to declare an identifier, putting indices in the wrong order, leaving out a necessary semicolon, or misspelling a label. For errors that are not caused by mistakes, the explanatory error message text will help you diagnose the problem and correct it.

When a compilation error is discovered, a *\$-symbol* and error number are printed below the offending symbol (usually to the right) on a separate line that begins with the four asterisks.

If more than one error is encountered on a line (possibly because the first error caused a series of other spurious errors) the \$-signs may be suppressed and error number squeezed. GAMS will not list more than 10 errors on any one line.

At the end of the echo print of the program, a list of all error numbers encountered, together with a description of the probable cause of each error, will be printed. The error messages are self-explanatory and will not be listed here.

It is worth noting that it is easy to produce a model that does not do what you want it to do, but does not contain errors in the sense that the term is being used in this section. The best precaution is to check your work carefully and build in as many automatic consistency checks as possible.

One mistake that may cause confusion is if a GAMS reserved word is used for a label or an identifier. In this case, it is impossible to provide helpful messages for technical reasons.

In some cases, an error may not be detected until the statement following its occurrence, where it may produce a number of error conditions whose explanations seem quite silly. Always check carefully for the cause of the first error is such a group, and look at the previous statement (and especially for missing semicolons) if nothing seems obvious.

The following example illustrates the general reporting format for compiler errors.

```
1 set c crops / wheat, corn, wheat, longaname /
**** $172
2 parameter price(c) / wheat 200, cotton 700 /
**** $170
3
Error Messages
170 Domain violation for element
172 Element is redefined
***** 2 ERROR(S) 0 WARNING(S)
...
***** USER ERROR(S) ENCOUNTERED
```

10.6.2 Compilation Time Errors

The reporting format for errors found while analyzing solve statements is more complicated than for normal compilation errors, mainly because many things must be checked. All identifiers referenced must be defined or assigned, the mathematics in the equations must match the model class, and so on. More elaborate reporting is required to accurately describe any problems found. The **solve** statement is only checked if the model has been found to be error free up to this point. This is not only because the check is comparatively expensive, but also because many erroneous and confusing messages can be produced while checking a **solve** in a program containing other errors.

Solve error messages are reported in two places and in two formats. First, they are shown immediately below the solve statement with a short text including the name of any offending identifier and the type of model involved. This will be sufficient in most cases. Second, a longer message with some hints appears with the rest of the error messages at the end of the compilation.

The example below illustrates how the general reporting format for compiler errors associated with a solve statement.

```
1 variables x,y, z ;
   2
     equations eq1 , eq2;
   3
   4
     eq1.. x**2 - y =e= z ;
  5
      eq2.. min(x,y) =1= 20 ;
   6
   7
     model silly / all / ;
   8
     solve silly using lp maximizing z ;
****
                                      $54,51,256
**** THE FOLLOWING LP ERRORS WERE DETECTED IN MODEL SILLY:
                              .. ENDOG OPERANDS FOR **
**** 54 IN EQUATION EQ1
**** 51 IN EQUATION EQ2
                               .. ENDOG ARGUMENT(S) IN FUNCTION
   9
Error Messages
 51 Endogenous function argument(s) not allowed in linear models
    Endogenous operands for ** not allowed in linear models
 54
256
     Error(s) in analyzing solve statement. More detail appears
     Below the solve statement above
                 O WARNING(S)
**** 3 ERROR(S)
**** USER ERROR(S) ENCOUNTERED
```

10.6.3 Execution Errors

Execution time errors are usually caused by illegal arithmetic operations such as division by zero or taking the log of a negative number. GAMS prints a message on the output file with the line number of the offending statement and continues execution. A GAMS program should never abort with an unintelligible message from the computer's operating system if an invalid operation is attempted. GAMS has rigorously defined an extended algebra that contains all operations including illegal ones. The model library problem [CRAZY] contains all non-standard operations and should be executed to study its exceptions.

Recall that GAMS arithmetic is defined over the closed interval [-INF,+INF] and contains values EPS (small but not zero), NA (not available), and UNDF (the result of an illegal operation). The results of illegal operations are propagated through the entire system and can be displayed with standard display statements. However remember that one cannot solve a model or save a work file if errors have been detected previously.

10.6.4 Solve Errors

The execution of a **solve** statement can trigger additional errors called **MATRIX ERRORS**, which report on problems encountered during transformation of the model into a format required by the solver. Problems are most often caused by illegal or inconsistent bounds, or an extended range value being used as a matrix coefficient. The example below shows the general format of these errors:

```
variable x;
   1
     equation eq1;
   2
   3
   4
      eq1.. x =l= 10 ;
   5 x.lo = 10;
   6 x.up = 5;
   7
     model wrong /eq1/;
   8
      solve wrong using lp maximizing x ;
   9
**** MATRIX ERROR - LOWER BOUNDS > UPPER BOUND
   (.L0, .L, .UP = 10, 0, 5)
Х
. . .
**** SOLVE from line 8 ABORTED, EXECERROR = 1
**** USER ERROR(S) ENCOUNTERED
```

Some solve statement require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under its direct control, errors associated with these calculations are reported in the solution report. Unless reset with the domlim option the subsystems will interrupt the solution process if arithmetic exceptions are encountered. They are then reported on the listing as shown in the following example:

```
1 variable x, y;
  2
     equation one;
  3
  4 one.. y = e = sqrt(10/x);
  5
    x.1 = 10;
  6 x.lo = 0;
  7
  8
     model divide / all / ;
     solve divide maximizing y using nlp;
  9
SOLVE
              SUMMARY
    MODEL
            DIVIDE
                                OBJECTIVE Y
    TYPE
            NLP
                                DIRECTION MAXIMIZE
                                FROM LINE 9
    SOLVER MINOS5
**** SOLVER STATUS
                      5 EVALUATION ERROR LIMIT
**** MODEL STATUS
                      7 INTERMEDIATE NONOPTIMAL
**** OBJECTIVE VALUE
                                   1.0000
```

```
1000.000
 RESOURCE USAGE, LIMIT
                                0.141
 ITERATION COUNT, LIMIT
                                         10000
                                0
 EVALUATION ERRORS
                                2
                                             0
 EXIT -- Termination requested by User in subroutine FUNOBJ after
                                                                   7 calls
**** ERRORS(S) IN EQUATION ONE
     2 INSTANCES OF - DIVISION BY ZERO (RESULT SET TO 0.1E+05)
                                   NONOPT ( NOPT)
**** REPORT SUMMARY :
                             1
                             O INFEASIBLE
                               UNBOUNDED
                             0
                             2
                                   ERRORS ( ****)
```

Note that the solver status returned with a value of 5, meaning that the solver has been interrupted because more than domlim evaluation errors have been encountered. The type of evaluation error and the equation causing the error are also reported.

If the solver returns an intermediate solution because of evaluation errors, the following solve will still be attempted. The only fatal GAMS error that can be caused by a solver program is the failure to return any solution at all. If this happens, as mentioned above, all possible information is listed on the GAMS output file and any solves following will not be attempted.

10.7 Summary

This is the end of the sequential discussion of the basic features of the GAMS language. All further chapters are geared towards more advanced use of GAMS.

11

Conditional Expressions, Assignments and Equations

11.1 Introduction

This chapter deals with the way in which conditional assignments, expressions and equations are made in GAMS. The index operations already described are very powerful, but it is necessary to allow for exceptions of one sort or another. For example, heavy trucks may not be able use a particular route because of a weak bridge, or some sectors in an economy may not produce exportable product. The use of a subset in an indexed expression has already been shown to provide some ability to handle exceptions.

11.2 Logical Conditions

Logical conditions are special expressions that evaluate to a value of True or False. Numerical Expressions can also serve as logical conditions. Additionally, GAMS provides for numerical relationship and logical operators that can be used to generate logical conditions. The next four sub-sections discuss these various building blocks that can be used to develop complex logical conditions.

11.2.1 Numerical Expressions as Logical Conditions

Numerical expressions can also serve as logical conditions - a result of zero is treated as a logical value of False, and a non-zero result is treated as a logical value of True.

The following numerical expression can be used to illustrate this point.

2*a - 4

This expression results in a logical value of False when **a** is 2 because the expression numerically evaluates to 0. For all other values of **a**, the expression results in a non-zero value, and therefore is equivalent to a logical value of True.

11.2.2 Numerical Relationship Operators

Numerical relationship operators compare two numerical expressions. For completeness all numerical relationship operators are listed below.

lt, < strictly less than

le, <= less than-or-equal to
eq, = equal to
ne, <> not equal to
ge, >= greater than or equal to
gt, > strictly greater than

The following example of a numerical relationship illustrates its use as a logical condition.

(sqr(a) > a)

This condition evaluates to False if $-1 \le a \le 1$. For all other values of a, this condition evaluates to True. Note that the same expression can also be written as (sqr(a) gt a).

11.2.3 Logical Operators

The logical operators available in GAMS are listed below.

notnotandandorinclusive orxorexclusive or

The truth table generated by these logical operators is given in table 11.1.

	and a		Door		
Oper	anas		Rest	uits	
a	ь	a and b	a or b	a xor b	not a
0	0	0	0	0	1
0	non-zero	0	1	1	1
non-zero	0	0	1	1	0
non-zero	non-zero	1	1	0	0

Table 11.1: Truth table of logical operators

11.2.4 Set Membership

Set membership can also be used as a logical condition. The label results in a logical value of True if it is a member of the set in question, and False if it is not. This is used with subsets and dynamic sets.

Consider the following example for illustration.

```
set i /1*10/
    subi(i) /1*3/;
```

The set subi(i) results in a logical value of True for all elements that belong to subi and False for all elements of i that do not belong to subi.

The use of set membership as a logical condition is an extremely powerful feature of GAMS and while its use will be illustrated later on in this chapter, its full power becomes clear when considered with the description of dynamic sets later.

11.2.5 Logical Conditions Involving Acronyms

Acronyms, which are character string values, can be used in logical conditions only with the = or <> operators only.

Consider the following example of logical conditions involving the use of acronyms,

dayofweek = wednesday
dayofweek <> thursday

where dayofweek is a parameter, and wednesday and thursday are acronyms.

11.2.6 Numerical Values of Logical Conditions

The previous four sub-sections have described the various features in GAMS that can be used as logical conditions. However, GAMS does not have a Boolean data type.

GAMS follows the convention that the result of a relational operation is zero if the assertion is False, and one if True.

Consider the following example for illustration,

x = (1 < 2) + (2 < 3)

The expression to the right of the assignment evaluates to 2 since both logical conditions within parenthesis are true and therefore assume a value of 1. Note that this is different from the assignment below,

x = (1 < 2) or (2 < 3)

which evaluates to 1 due to the or operator behaving as explained above.

11.2.7 Mixed Logical Conditions - Operator Precedence

The building blocks discussed in the first four subsections can be used to generate more complex logical conditions. The default precedence order in a logical condition used by GAMS in the absence of parenthesis is shown in table 11.2 in decreasing order.

Operation	Operator
Exponentiation	**
Numerical Operators	
- Multiplication, Division	*, /
- Unary operators - Plus, Minus	+, -
- Binary operators - addition, subtraction	+, -
Numerical Relationship operators	<, <=, =, <>, >=, >
Logical Operators	
- not	not
- and	and
- or, xor	or, xor

Table 11.2: Operator precedence

Note that in the case of operators with the same precedence, the order in which the operator appears in the expression is used as the precedence criterion, with the order reducing from left to right.

It is always advisable to use parentheses rather than relying on the precedence order of operators. It prevents errors and makes the intention clear.

Consider the following example for illustration,

x - 5*y and z - 5

is treated equivalent to (x - (5*y)) and (z-5). However, note that the use of parenthesis does make the expression clearer to understand.

11.2.8 Mixed Logical Conditions - Examples

Some simple examples of logical conditions, containing the building blocks described in the previous sub-sections, are shown in table 11.3 to illustrate the generation and use of more complex logical conditions.

Logical Condition	Numerical Value	Logical Value
(1 < 2) + (3 < 4)	2	True
(2 < 1) and $(3 < 4)$	0	False
(4*5 - 3) + (10/8)	17.125	True
(4*5 - 3) or (10 - 8)	1	True
(4 and 5) + (2*3 <= 6)	2	True
(4 and 0) + (2*3 < 6)	0	False

Table 11.3: Examples of logical conditions

11.3 The Dollar Condition

This section introduces the dollar operator, which is one of the most powerful features of GAMS. The dollar operator operates with a logical condition. The term (condition) can be read as 'such that condition is valid' where condition is a logical condition.

The dollar logical conditions cannot contain variables. Variable attributes (like .1 and .m) are permitted however.

The dollar operator is used to model conditional assignments, expressions, and equations. The following subsection provides an example that will clarify its use. The next section will deal individually with the topic of using dollar conditions to model conditional assignments, expressions, and equations respectively.

11.3.1 An Example

Consider the following simple condition,

if (b > 1.5), then a = 2

This can be modeled in GAMS using the dollar condition as follows,

a(b > 1.5) = 2;

If the condition is not satisfied, no assignment is made. Note that one can *read* the **\$** as 'such that' to clarify the meaning: 'a, such that b is greater than 1.5, equals 2'.

11.3.2 Nested Dollar Conditions

Dollar conditions can be also nested. The term \$(condition1\$(condition2)) can be read as \$(condition1 and condition2).

■ For nested dollar conditions, all succeeding expressions after the dollar must be enclosed in parentheses.

Consider the following example,

u(k)\$(s(k)\$t(k)) = a(k) ;

K, s(k), t(k), and i are sets, while u(k) and a(i) are parameters. The assignment will be made only for those members of k that are also members of both s and t. Note the position of the parenthesis in the dollar condition. The statement above can be rewritten as

u(k)\$(s(k) and t(k)) = a(k) ;

To assist with the readability of statements, it is strongly recommended to use the logical and operator instead of nesting dollar operators.

11.4 Conditional Assignments

The statement comprising the example in the Section before was a conditional assignment. In this example, the dollar condition was on the left-hand-side of the assignment.

- The effect of the dollar condition is significantly different depending on which side of the assignment it is in.
- In many cases, it may be possible to use either of the two forms of the dollar condition to describe an assignment. In such a case, clarity of logic should be used as the criterion for choice.

The next two subsections describe the use of the dollar condition on each side of the assignment.

11.4.1 Dollar on the Left

The example illustrated in the section above uses the dollar condition on the left-hand side of the assignment statement.

- For an assignment statement with a dollar condition on the left-hand side, no assignment is made unless the logical condition is satisfied. This means that the previous contents of the parameter on the left will remain unchanged for labels that do not satisfy the condition.
- If the parameter on the left-hand side of the assignment has not previously been initialized or assigned any values, zeroes will be used for any label for which the assignment was suppressed.

Consider the following example adapted from [CHENERY],

rho(i)\$(sig(i) ne 0) = (1./sig(i)) - 1. ;

The parameter sig(i) has been previously defined in the model and the statement above is used to calculate rho(i). The dollar condition on the statement protects against dividing by zero. If any of the values associated with sig(i) turn out to be zero, no assignment is made and the previous values of rho(i) remain. As it happens, rho(i) was previously not initialized, and therefore all the labels for which sig(i) is 0 will result in a value of 0.

Now recall the convention, explained in Section 11.2.1 that non zero implies True and zero implies False. The assignment above could therefore be written as

rho(i)\$sig(i) = (1./sig(i)) - 1.;

11.4.2 Dollar on the Right

For an assignment statement with a dollar condition on the right hand side, an assignment is always made. If the logical condition is not satisfied, the corresponding term that the logical dollar condition is operating on evaluates to 0.

Consider the following example, which is a slight modification to the one described in Section 11.3.1,

x = 2\$(y > 1.5);

Expressed in words, this is equivalent to,

if (y > 1.5) then (x = 2), else (x = 0)

Therefore an if-then-else type of construct is implied, but the else operation is predefined and never made explicit. Notice that the statement in the illustrative example above can be re-written with an explicit if-then-else and equivalent meaning as

x = 2\$(y gt 1.5) + 0\$(y le 1.5);

This use of this feature is more apparent for instances when an else condition needs to be made explicit. Consider the next example adapted from [FERTD]. The set i is the set of plants, and are calculating mur(i), the cost of transporting imported raw materials. In some cases a barge trip must be followed by a road trip because the plant is not alongside the river and we must combine the separate costs. The assignment is:

mur(i) =(1.0 +. 0030*ied(i,'barge'))\$ied(i,'barge')
+(0.5 + .0144*ied(i,'road'))\$ied(i,'road');

This means that if the entry in the distance table is not zero, then the cost of shipping using that link, which has a fixed and a variable components, is added to the total cost,. If there is no distance entry, there is no contribution to the cost, presumably because that mode is not used.

11.4.3 Filtering Controlling Indices in Indexed Operations

The controlling indices can, in certain cases, be filtered through the conditional set without the use of the dollar operator. Consider the example described in that section. The total cost of shipment is obtained through the following equation,

```
variable shipped(i,j), totcost ;
equation costequ ;
cost.. totcost =e= sum((i,j)$ij(i,j), shipcost(i,j)*shipped(i,j));
```

where shipped is the amount of material shipped from i to j, and totcost is the total cost of all shipment. The equation above can be written as,

cost.. totcost =e= sum(ij, shipcost(ij)*shipped(ij));

However, if the original equation is expressed as,

cost.. totcost =e= sum((i,j)\$ij(i,j), factor*congestfac(j)*distance(i,j) *shipped(i,j));

Index j appears separately from i in congestfac(j). The equation then needs to be simplified as,

cost.. totcost =e= sum(ij(i,j), factor*congestfac(j)*distance(ij) *shipped(ij));

Note that the presence of j separately in the indexed expression necessitated the use of ij(i,j) rather than ij.

11.4.4 Filtering Sets in Assignments

Consider the following statement,

u(k)\$s(k) = a(k);

where k and s(k) are sets, while u and a are parameters. This can be rewritten as,

u(s) = a(s);

Note that the assignment has been filtered through the conditionality without the use of the dollar operator. This is a cleaner and more understable representation of the assignment. This feature gets more useful when dealing with tuples (sets with multiple indices).

Consider the following example for calculating the travel cost for a fictional parcel delivery service between collection sites (i) and regional transportation hubs (j),

```
set
        i /miami,boston,chicago,houston,sandiego,phoenix,baltimore/
    j /newyork,detroit,losangeles,atlanta/ ;
set ij(i,j) /
    boston.newyork
    baltimore.newyork
    miami.atlanta
    houston.atlanta
    chicago.detroit
    sandiego.losangeles
    phoenix.losangeles / ;
table distance(i,j) "distance in miles"
           newyork
                       detroit losangeles
                                                 atlanta
miami
              1327
                          1387
                                       2737
                                                     665
boston
               216
                            699
                                       3052
                                                    1068
                                       2095
                                                     695
chicago
               843
                            275
              1636
                           1337
                                       1553
houston
                                                     814
sandiego
               206
              2459
                           1977
                                        398
                                                    1810;
phoenix
parameter factor,shipcost(i,j) ; factor = 0.009 ;
```

The set ij denotes the regional transportation hub for each collection site. Factor is the cost estimate per unit mile. The cost of transporting parcels (shipcost) from a local collection site (i) to a regional hub(j) is then provided by the following assignment,

shipcost(i,j)\$ij(i,j) = factor*distance(i,j) ;

Note that i and j do not appear separately in the assignment for shipcost. The assignment can then be simply written as,

```
shipcost(ij) = factor*distance(ij) ;
```

If i or j appear separately in any assignment, the above simplification cannot be made. For example, consider that travelcost depended not only on factor and the distance between collection sites and regional hubs but also the load on the regional hub.

```
Parameter congestfac(j) /
    newyork 1.5
    detroit 0.7
    losangeles 1.2
    atlanta 0.9/;
```

Congestfac is a parameter used to model the congestion at each regional hub. The unit cost of shipment is then computed as follows:

```
shipcost(i,j)$ij(i,j) = factor*congestfac(j)*distance(i,j) ;
```

This cannot be re-written as

```
shipcost(ij) = factor*congestfac(j)*distance(ij) ;
```

The above representation has the index j on the right hand side, but not on the left hand side. As explained before, GAMS will flag this assignment as an error. However, the following representation will work:

```
shipcost(ij(i,j)) = factor*congestfac(j)*distance(ij) ;
```

In the above assignment ij is specifically denoted as a tuple of i and j which then appear on the left hand side.

11.5 Conditional Indexed Operations

Another important use of the dollar condition is to control the domain of operation of indexed operations. This is conceptually similar to the 'dollar on the left' described in Section 11.3.1.

Consider the following example adapted from [GTM].

```
tsubc = sum(i$(supc(i) ne inf), supc(i)) ;
```

This statement evaluates the sum of the finite values in supc.

```
A common use of dollar controlled index operations is where the control is itself a set. This importance of this concept will become apparent with the discussion of dynamic sets.
```

A set has been used to define the mapping between mines and ports in Chapter 4. Another typical example is a set-to-set mapping defining the relationship between states and regions, used for aggregating data obtained by state to the models requirements (by region).

```
sets r / west,east /
    s / florida,texas,vermont,maine /
    corr(r,s) / north.(vermont,maine)
        south.(florida,texas) /
parameter y(r)
    income (s) "income of each state"
        / florida 4.5, vermont 4.2
        texas 6.4, maine 4.1 / ;
```

The set corr provides a correspondence to show which states belong to which regions. The parameter income is the income of each state. Y(r) can be calculated with this assignment statement:

```
y(r) = sum(s$corr(r,s), income(s)) ;
```

For each region r, the summation over s is only over those pairs of (r,s) for which corr(r,s) exists. Conceptually, set existence is analogous to the Boolean value True or the arithmetic value 'not zero'. The effect is that only the contributions for 'vermont' and 'maine' are included in the total for 'north', and 'south' includes only 'texas' and 'florida'.

Note that the summation above can also be written as sum(s, income(s)\$corr(r,s)) but this form is not as easy to read as controlling the index of summation.

11.5.1 Filtering Controlling Indices in Indexed Operations

The controlling indices can, in certain cases, be filtered through the conditional set without the use of the dollar operator. Consider the example described in that section. The total cost of shipment is obtained through the following equation,

```
variable shipped(i,j), totcost ;
equation costequ ;
cost.. totcost =e= sum((i,j)$ij(i,j), shipcost(i,j)*shipped(i,j));
```

where shipped is the amount of material shipped from i to j, and totcost is the total cost of all shipment. The equation above can be written as,

```
cost.. totcost =e= sum(ij, shipcost(ij)*shipped(ij));
```

However, if the original equation is expressed as,

```
cost.. totcost =e= sum((i,j)$ij(i,j),
factor*congestfac(j)*distance(i,j) *shipped(i,j));
```

Index j appears separately from i in congestfac(j). The equation then needs to be simplified as,

```
cost.. totcost =e= sum(ij(i,j),
factor*congestfac(j)*distance(ij) *shipped(ij));
```

Note that the presence of j separately in the indexed expression necessitated the use of ij(i,j) rather than ij.

11.6 Conditional Equations

The dollar operator is also used for exception handling in equations. The next two subsections discuss the two main uses of dollar operators within equations - within the body of an equation, and over the domain of definition.

11.6.1 Dollar Operators within the Algebra

A dollar operator within an equation is analogous to the dollar control on the right of assignments as discussed in Section 11.4.2, and if one thinks of 'on the right' as meaning on the right of the '..' then the analogy is even closer. An if-else operation is implied as it was with assignments. It is used to exclude parts of the definition from some of the generated constraints.

Consider the following example adapted from [CHENERY],

mb(i).. x(i) =g= y(i) + (e(i) - m(i))\$t(i) ;

The term is added to the right hand side of the equation only for those elements of i that belong to t(i).

Controlling indexing operations using the dollar condition can also be done as with any assignment. Consider the following supply balance (sb) equation from [GTM],

```
sb(i).. sum(j$ij(i,j), x(i,j)) =l= s(i) ;
```

11.6.2 Dollar Control over the Domain of Definition

This is analogous to the dollar control on the left of assignments as discussed in Section 11.4.1, and if one thinks of 'on the left' as meaning on the left of the '...' then the analogy is even closer.

The purpose of the dollar control over the domain of definition of equations is to restrict the number of constraints generated to less than that implied by the domain of the defining sets.

Consider the following example adapted from [FERTS]:

```
cc(m,i)$mpos(m,i)..
sum(p$ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i);
```

Cc is a capacity constraint defined for all units (m) and locations (i).

Not all types of units exist at all locations, however, and the mapping set mpos(m,i) is used to restrict the number of constraints actually generated. The control of the summation over p with ppos(p,i) is an additional one, and is required because not all processes (p) are possible at all locations (i).

11.6.3 Filtering the Domain of Definition

The same rules that apply to filtering assignments and controlling indices in indexed operations applies to equation domains as well. Consider the following equation using the same set definitions as described before,

```
parameter bigM(i,j) ;
variable shipped(i,j) ;
binary variable bin(i,j) ;
equation logical(i,j) ;
logical(i,j)$ij(i,j).. shipped(i,j) =l= bigM(i,j)*bin(i,j) ;
```

The equation logical relates the continuous variable shipped(i,j) to the binary variable bin(i,j). This can be simplified as follows:

```
logical(ij).. shipped(ij) =l= bigM(ij)*bin(ij) ;
```

Note that if the right hand side of the equation contained any term that was indexed over i or j separately, then the equation logical(i,j)\$ij(i,j) would have to be simplified as logical(ij(i,j)).

12

Dynamic Sets

12.1 Introduction

All the sets that have been discussed so far had their membership declared as the **set** itself was declared, and the membership was never changed. In this chapter we will discuss changing the membership of **sets**. A set whose membership can change is called a dynamic set to contrast it with a static set whose membership will never change. The distinction is important and will be discussed in detail in this chapter. This is a topic that has been avoided until now because of a potential confusion for new users. Advanced Users will, however, find it useful.

12.2 Assigning Membership to Dynamic Sets

Sets can be assigned in a similar way to other data types. One difference is that arithmetic operations cannot be performed on sets in the same way that they can on *value typed* identifiers (parameters, or variables and equations subtypes). A dynamic set is most often used as a *controlling index* in an assignment or an equation definition, or as the controlling entity in a dollar-controlled indexed operation.

12.2.1 The Syntax

In general, the syntax in GAMS for assigning membership to dynamic sets is:

```
set_name(domain_name | domain_label) = yes | no ;
```

Set_name is the internal name of the set (also called an identifier) in GAMS. Yes and no are keywords used in GAMS to denote membership or absence respectively from the assigned set.

- The most important principle to follow is that a dynamic set should always be domain checked at declaration time to be a subset of a static set (or sets).
- It is of course possible to use dynamic sets that are not domain checked, and this provides additional power, flexibility, lack of intelligibility, and danger. Any label is legal as long as the dimensionally, once established, is preserved.

12.2.2 Illustrative Example

The following example, adapted from [ZLOOF], is used to illustrate the assignment of membership to dynamic sets.

```
/ dish,ink,lipstick,pen,pencil,perfume /
         all items
set item
subitem1(item)
                first subset of item / pen,pencil /
                second subset of item;
subitem2(item)
subitem1('ink') = yes ;
                          subitem1('lipstick')
                                              =
                                                  yes ;
subitem2(item) = yes ;
                          subitem2('perfume')
                                               =
                                                  no ;
display subitem1, subitem2;
```

Note that the sets subitem1 and subitem2 are declared like any other set. The two sets become dynamic because of assignments. They are also domain checked: the only members they will ever be able to have must also be members of item. And item is a static set and henceforth its membership is frozen. The first two assignments each add one new element to subitem1. The third is an example of the familiar indexed assignment: subitem2 is assigned all the members of item. The output caused by the display statement, that will reveal the membership of the sets, is shown below for verification.

	7	SET	SUBITEM1	first	subset (of item
INK	,	LIPSTICK	, PEN	, PEN	CIL	
	7	SET	SUBITEM2	second	subset	of item
DISH	,	INK	, LIPSTICK	, PEN	,	PENCIL

The elements are displayed in the order specified in the declaration of item.

12.2.3 Dynamic Sets with Multiple Indices

Dynamic sets, like static sets, can have up to 20 dimensions. The following example illustrates assignments for multi-dimensional sets.

```
Sets item items sold /pencil, pen/
    sup suppliers /bic, parker, waterman /
    supply(item,sup) ;
supply('pencil','bic') = yes ;
supply('pen',sup) = yes ;
```

All the mechanisms using asterisks and parenthesized lists that we introduced in the discussion on static sets in chapter 4 are available for dynamic sets as well.

12.2.4 Assignments over the Domain of Dynamic Sets

One can make an assignment over the domain of a dynamic set because dynamic sets are known to be proper subsets of static sets. This is not the same as doing domain checking using a dynamic set.

The following example, adapted from the Section 12.2.2 illustrates the use of dynamic sets as domains:

```
subitem1(item) = no
subitem1(subitem2) = yes;
```

The first assignment ensures that **subitem1** is empty. Note that this can also be done with parameters. For example,

```
parameter inventory(item) ;
inventory(subitem1) = 25 ;
```

12.2.5 Equations Defined over the Domain of Dynamic Sets

It is sometimes necessary to define an equation over a dynamic set.

The trick is to declare the equation over the entire domain but define it over the dynamic set.

The following example illustrates its use,

```
set allr all regions / n, s, w, e, n-e, s-w /
    r(alr) region subset for particular solution ;
scalar price /10/ ;
equations prodbal(allr) production balance ;
variables activity(allr) first activity
    revenue(allr) revenue ;
prodbal(r).. activity(r)*price =e= revenue(r) ;
```

To repeat the important point: the equation is *declared* over allr but referenced over r. Then arbitrary assignments can be made to r within the membership of allr.

12.3 Using Dollar Controls with Dynamic Sets

The rest of this chapter requires an understanding of the dollar condition. All the dollar control machinery is available for use with dynamic sets. In fact, the full power of dynamic sets can be exploited using these dollar controls.

Note that the dynamic set has values of yes and no only, and can therefore be treated as a logical statement. The only operations that can be performed on dynamic sets inside the dollar operator are therefore not, and, or, or xor, as well as the set operations described in Section 12.4, page 120.

The main uses of dynamic sets inside dollar conditions are in assignments, indexed operations and in equations. Each of these will be discussed in detail in the following subsections. Examples will be used to illustrate its use in each of the three cases.

12.3.1 Assignments

Dynamic sets can be used inside dollar conditions within assignments defining other dynamic sets or parameters.

As an illustration of its use in defining other dynamic sets, the two statements in the example from Section 12.2.4 can be written with equivalent effect as

```
subitem1(item) = yes$subitem2(item) ;
```

which is a terse form of the following statement

```
subitem1(item) = yes$subitem2(item) + no$(not subitem2(item)) ;
```

The value used in the implied "else" that goes with "dollar on the right" is no in a set assignment, rather than zero which is used with normal data.

The second example from Section 12.2.4 can be rewritten as follows to illustrate the use of dynamic sets in defining parameters,

```
inventory(item)$subitem1(item) = 25 ;
```

12.3.2 Indexed Operations

Another important use of dollar controls with dynamic sets is to control the domain while performing indexed operations like sum and prod. Consider the following adaptation of the second example from Section 12.3.1.

```
parameter totinv total inventory ;
totinv = sum(item$subitem1(item),inventory(item)) ;
```

This example has been shown only for illustration. Note that the second statement above can also be rewritten tersely as

```
totinv = sum(subitem1,inventory(subitem1)) ;
```

This is not always possible. Consider the following artificially created example,

```
sets item items sold /pencil, pen/
    sup suppliers /bic, parker, waterman /
    dep department /stationery, household/
    supply(item,sup) ;
supply('pencil', 'bic') = yes ; supply('pen',sup) = yes ;
parameter totsales(dep) ;
totsales(dep) = sum(item$supply(item,'bic'), sales(dep,item)) ;
```

The assignment above is used to find the total sales of all departments that sell items supplied by bic. Note that the dynamic set is used to limit the domain of summation to those for which supply(item,'bic') is true.

12.3.3 Equations

Dynamic sets can be used inside dollar conditions in equations both as part of the equation algebra, or while defining the domain of the equation. The first case is similar to the case of assignments discussed in Section 12.3.1. The latter case is used to restrict the equation over the domain of a dynamic set. The equation defined in the example from Section 12.2.5 can be rewritten with equivalent effect as follows,

```
prodbal(allr)$r(allr).. activity(allr)*price =e= revenue(allr) ;
```

The domain of definition of equation prodbal is restricted to those elements that belong to the dynamic set r.

12.3.4 Filtering through Dynamic Sets

The filtering process explained in previous sections is valid when the conditional set is a dynamic one. Consider the following two examples as described before,

```
inventory(item)$subitem1(item) = 25 ;
prodbal(allr)$r(allr).. activity(allr)*price =e= revenue(allr) ;
```

These statements can be rewritten as,

```
inventory(subitem1) = 25 ;
prodbal(r).. activity(r)*price =e= revenue(r) ;
```

12.4 Set Operations

This section describes how various symbolic set operations can be performed in GAMS using dynamic sets. The Union, Intersection, Complement, and Difference set operations are described individually in the following subsections. Once again the example from Section 12.2.2 is used to illustrate each operation.

12.4.1 Set Union

The symbol + performs the set union operation. Consider the following example,

subitem3(item) = subitem1(item) + subitem2(item) ;

The membership of subitem3 is set equal to all the elements of subitem1 and all the elements of subitem2. The operation above is equivalent to the following longer way of representation,

subitem3(item)=no; subitem3(subitem2)=yes; subitem3(subitem1)=yes;

12.4.2 Set Intersection

The symbol * performs the set intersection operation. Consider the following example,

subitem3(item) = subitem1(item) * subitem2(item) ;

The membership of subitem3 is set equal to only those present in both subitem1 and subitem2. The operation above is equivalent to the following longer way of representation,

subitem3(item)=yes\$(subitem1(item) and subitem2(item)) ;

12.4.3 Set Complement

The operator not performs the set complement operation. Consider the following example,

```
subitem3(item) = not subitem1(item) ;
```

The membership of subitem3 is set equal to all those in item but not in subitem1. The operation above is equivalent to the following longer way of representation,

```
subitem3(item)=yes; subitem3(subitem1)=no;
```

12.4.4 Set Difference

The operator - performs the set difference operation. Consider the following example,

```
subitem3(item) = subitem1(item) - subitem2(item) ;
```

The membership of subitem3 is set equal to all elements that are members of subitem1 but subitem2. The operation above is equivalent to the following longer way of representation,

```
subitem3(item)=yes$(subitem1(item)); subitem3(subitem2)=no;
```

12.5 Summary

The purpose of set assignments is to make calculations based on given data (the static sets) for use in exception handling. It is one more example of the principle of entering a small amount of data and building a model up from the most elemental information.

13

Sets as Sequences: Ordered Sets

13.1 Introduction

In our original discussion of sets in Chapter 4, we said that unless there is a special need to do things differently, a one-dimensional set should be regarded as an unordered collection of labels. In this chapter we will discuss special features that can be used when you need to be able to deal with a set as if it were a sequence.

For example, in economic models that explicitly represent conditions in different time periods, it is necessary to refer to the *next* or *previous* time period, because there must be links between the periods. As another example, stocks of capital are normally tracked through such models by equations of the form 'stocks at the end of period n are equal to stocks at the end of period n - 1 plus net gains during period n'. Location problems, where the formulation may require a representation of contiguous areas, as in a grid representation of a city, and scheduling problems are other classes of problems in which sets must also have the properties of sequences.

Models involving sequences of time periods are often called dynamic models, because they describe how conditions change over time. This use of the word dynamic unfortunately has a different meaning from that used in connection with sets, but this is unavoidable.

13.2 Ordered and Unordered Sets

As with sets used in domain checking, restrictions are imposed when the set needs to be referred as if it were a sequence. The notion of static sets was introduced already: the set must be initialized with a list of labels enclosed in slashes at the time the set is declared, and never changed afterwards.

- Solution of the sets must be static sets. In other words, no order is possible for dynamic sets.
- GAMS maintains one list of unique elements the labels that are used as elements in one or more sets. The order of the elements in any one set is the same as the order of those elements in that unique element list. This means that the order of a set may not be what it appears to be if some of the labels were used in an earlier definition.
- The map of your labels in the GAMS order can be seen by putting the compiler directive **\$onuellist** somewhere before the first set declaration.
- A good rule of thumb is that if the labels in a set one wants to be ordered have not been used already, then they will be ordered.

The map is shown with the other compiler maps after the listing of your program. In the example below we show ordered and unordered sets and the map showing the order. The input is:

The map below shows the entry order (the important one) and the sorted order, obtained by sorting the labels into dictionary order. The single digits on the left are the sequence numbers of the first label on that line.

```
General
              Algebraic Modeling
                                                  System
Unique Element Listing
Unique Elements in Entry Order
                  1988
                                      1990
                                                 1991
                                                          1983
   1
       1987
                            1989
   7
       1984
                  1985
                            1986
Unique Elements in Sorted Order
   1
       1983
                  1984
                            1985
                                      1986
                                                 1987
                                                          1988
   7
       1989
                  1990
                            1991
```

A set can always be made ordered by moving its declaration closer to the beginning of the program. With these restrictions in mind, we move on the operations that are used in dealing with sets as sequences.

13.3 Ord and Card

In Chapter 4, it was explained that labels do not have a numerical value. The examples used were that the label '1986' does not have a numerical value of 1986 and the label '01' is different from the label '1'. This section introduces two operators - ord and card that return integer values when applied to sets. While the integer values returned do not represent the numerical value of the label, they can be used for the same purpose.

The next two subsections describe each of these two functions in turn.

13.3.1 The Ord Operator

Ord returns the relative position of a member in a set. 🖙 Ord can be used only with a one-dimensional, static, ordered set.

Some examples show the usage.

```
set t time periods / 1985*1995 /
parameter val(t);
val(t) = ord(t);
```

As a result of the statements above, the value of val('1985') will be 1, val('1986') will be 2 and so on.

A common use of **ord** is in setting up vectors that represent quantities growing in some analytically specified way. For example, suppose a country has 56 million people in the base period and population is growing at the rate of 1.5 percent per year. Then the population in succeeding years can be calculated by using:

```
population(t) = 56*(1.015**(ord(t) - 1));
```

It is often useful to simulate general matrix operations in GAMS. The first index on a two dimensional parameter can conveniently represent the rows, and the second the columns, and order is necessary. The example below shows how to set the upper triangle of a matrix equal to the row index plus the column index, and the diagonal and lower triangle to zero.

```
set i row and column labels / x1*x10 /; alias (i,j);
parameter a(i,j) a general square matrix;
a(i,j)$(ord(i) lt ord(j)) = ord(i) + ord(j);
```

13.3.2 The Card Operator

Card returns the number of elements in a set. Card can be used with any set, even dynamic or unordered ones. The following example illustrates its use:

```
set t time periods / 1985*1995 /
parameter s ; s = card(t);
```

As a result of the statement above, \mathbf{s} will be assigned the value 11.

A common use of card is to specify some condition only for the final period, for example to fix a variable. An artificial example is:

c.fx(t) (ord(t) = card(t)) = demand(t)

which fixes the variable for the last member only: no assignment is made for other members of t. The advantage of this way of fixing c is that the membership of t can be changed safely and this statement will always fix c for the last one.

13.4 Lag and Lead Operators

The lag and lead operators are used to relate the *current* to the *next* or *previous* member of a set. In order to use these operators the set in question must, of course, be ordered. GAMS provides two forms of lag and lead operators

- \succ Linear Lag and Lead Operators (+, -)
- ➤ Circular Lag and Lead Operators (++, --)

The difference between these two types of operators involves the handling of endpoints in the sequence. GAMS provides some built in facilities to deal with this issue, but in any work involving sequences the user must think carefully about the treatment of endpoints, and all models will need special exception handling logic to deal with them.

In the linear case, the members of the set that are endpoints are left hanging. In other words, there are no members preceding the first member or following the last one. This may cause the use of non-existent elements. The next section will describe how this is handled in GAMS. This form of the lag and lead operators is useful for modeling time periods that do not repeat. A set of years (say 1990 to 1997) is an example. The operators are + and -.

GAMS is able to distinguish linear lag and lead operators (+,−) from arithmetic operators by context.

In the circular case, the first and last members of the set are assumed to be adjacent, so as to form a circular sequence of members. The notion is that 'first -1' is a reference to 'last' and 'last +2' is the same as 'first +1' and so on. All references and assignments are defined. This is useful for modeling time periods that repeat, such as months of the year or hours in the day. It is quite natural to think of January as the month following December. Agricultural farm budget models and workforce scheduling models are examples of applications where circular leads occur naturally. The operators are ++ and --.

The next two sections will describe the use of these lag and lead operators in assignment statements and in equations respectively.

13.5 Lags and Leads in Assignments

One use of the lag and lead operator is in assignment statements. The use of a lag and lead operator on the right-hand-side of an assignment is called a reference, while its use in the left-hand-side is called an assignment and

involves the definition of a domain of the assignment. The concepts behind reference and assignment are equally valid for the linear and circular forms of the lag and lead operator. However, the importance of the distinction between reference and assignment is not pronounced for circular lag and lead operators because non-existent elements are not used in this case.

```
A reference to a non-existent element causes the default value (zero in this case) to be used, whereas
an attempt to assign to a non-existent element results in no assignment being made.
```

The next two sub-sections provide examples illustrating the use of the linear form of the lag and lead operators for reference and assignment. Section 13.5.3 will illustrate the use of the circular form of the lag and lead operator.

13.5.1 Linear Lag and Lead Operators - Reference

Consider the following example, where two parameters **a** and **b** are used to illustrate the linear lag and lead operators for reference.

```
set t time sequence / y-1987*y-1991 / ;
parameter a(t), b(t) ;
a(t) = 1986 + ord(t) ;
b(t) = -1; b(t) = a(t-1) ;
option decimals=0; display a, b ;
```

The option statement suppresses the decimal places from the display. The results are shown below.

```
---- 6 PARAMETER A
Y-1987 1987, Y-1988 1988, Y-1989 1989, Y-1990 1990, Y-1991 1991
---- 6 PARAMETER B
Y-1988 1987, Y-1989 1988, Y-1990 1989, Y-1991 1990
```

For a, as expected, the values 1987, 1988 up to 1991 are obtained corresponding to the labels y-1987, y-1988 and so on. b is initialized to -1.

For b, the assignment is done over all members of t, and for each, the value of a from the previous period is assigned to the current member of b. If no previous period, as with y-1987, zero is used, and b('y-1987') becomes zero, replacing the previous value of -1.

13.5.2 Linear Lag and Lead Operators - Assignment

Consider the following example, where two parameters a and c are used to illustrate the assignment of linear lag and lead operators.

```
set t time sequence / y-1987*y-1991 / ;
parameter a(t), c(t) ;
a(t) = 1986 + ord(t) ;
c(t) = -1; c(t+2) = a(t) ;0; display a, c;
```

The results are shown below,

```
---- 6 PARAMETER A
Y-1987 1987, Y-1988 1988, Y-1989 1989, Y-1990 1990, Y-1991 1991
---- 6 PARAMETER C
Y-1987 -1, Y-1988 -1, Y-1989 1987, Y-1990 1988, Y-1991 1989
```

The assignment to a is explained in Section 13.5.1. The assignment to c is different. It is best to spell it out in words. For each member of t in sequence, find the member of c associated with t+2. If it exists, replace its value with that of a(t). If not (as with y-1990 and y-1991) make no assignment. The first member of t is y+1987,

and therefore the first assignment is made to c('y-1989') which takes the value of a('y-1987'), viz., 1987. No assignments at all are made to c('y-1987') or c('y-1988'): these two retain their previous values of -1.

The lag (or lead) value does not have to be an explicit constant: it can be arbitrary expression, provided that it evaluates to an integer. If it does not, error messages will be produced. A negative result causes a switch in sense (from lag to lead, for example). The following is guaranteed to set d(t) to all zero:

d(t) = d(t - ord(t));

13.5.3 Circular Lag and Lead Operators

The following example illustrates the use of circular lag and lead operators.

The results are shown below,

```
---- 7 PARAMETER VAL

SPRING 10, SUMMER 15, AUTUMN 12, WINTER 8

---- 7 PARAMETER LAGVAL2

SPRING 12, SUMMER 8, AUTUMN 10, WINTER 15

---- 7 PARAMETER LEADVAL

SPRING 8, SUMMER 10, AUTUMN 15, WINTER 12
```

The parameter lagval2 is used for reference while lagval1 if used for assignment. Notice that the case of circular lag and lead operators does not lead to any non-existent elements. The difference between reference and assignment is therefore not important. Note that the following two statements from the example above,

```
lagval2(s) = val(s--2);
leadval(s++1) = val(s);
```

are equivalent to

lagval2(s++2) = val(s) ; leadval(s) = val(s--1) ;

The use of reference and assignment have been reversed with no difference in effect.

13.6 Lags and Leads in Equations

The principles established in the previous section follow quite naturally into equation definitions. A lag or lead operation in the body of an equation (to the right of the '..' symbol) is a reference, and if the associated label is not defined, the term vanishes. A lag or lead to the left of the '..' is a modification to the domain of definition of the equation. The linear form may cause one or more individual equations to be suppressed.

All lag and lead operands must be exogenous.

The next two sub-sections provide examples illustrating the use of the linear form of the lag and lead operators in equations for reference and to modify the domain of its definition. Section 13.5.3 will illustrate the use of the circular form of the lag and lead operator in equations.

13.6.1 Linear Lag and Lead Operators - Domain Control

Consider the following example adapted from [RAMSEY],

```
sets t time periods /1990*2000/
     tfirst(t) first period
     tlast(t) last period;
tfirst(t) = yes$(ord(t) eq 1);
tlast(t) = yes$(ord(t) eq card(t) );
display tfirst, tlast;
variables k(t)
                capital stock (trillion rupees)
                investment (trillion rupees per year) ;
         i(t)
equations kk(t)
                 capital balance (trillion rupees)
         tc(t)
                 terminal condition(provides for post-term growth) ;
kk(t+1)..
          k(t+1)
                       = e = k(t) + i(t);
tc(tlast).. g*k(tlast) =l= i(tlast);
```

The declaration of t is included, as are a couple of dynamic sets that are used to handle the first and last periods (terminal conditions) in a clean way.

The interesting equation is kk, the capital balance. The set t contains members 1990 to 2000, and so there will be a capital stock constraint for 1991 to 2000. Spelling out the constraint for 1991,

k('1991') =e= k('1990') + i('1990') ;

The lead operator on the domain of definition has restricted the number of constraints generated so that there are no references to non-existent variables: the generated problem will have 10 kk constraints defining the relationship between the 11 k capital values.

The other interesting point in the [RAMSEY] excerpt is that the constraint tc is explicitly defined only for the final period because of the assignment to the set tlast. Notice the use of dynamic sets to control the domain of the two equations. The set tfirst is also used in other parts of the model to set initial conditions, particularly the capital stock in the first period, k('1990').

13.6.2 Linear Lag and Lead Operators - Reference

In the example discussed in Section 13.6.1, equation kk can be rewritten with equivalent effect as

```
kk(t)$(not tfirst(t)).. k(t+1) =e= k(t) + i(t);
```

The dollar condition will cause one of the individual equations to be suppressed.

However, note that using lags and leads in the equation domain will always cause one or more individual equations to be suppressed, and this may not be desirable in every case. Consider the following modified set of constraints to the one discussed in the previous example. It is expressed with the lag and lead operators being used to control the domain of the equation definition.

```
kk(t+1).. k(t+1) =e= k(t) + i(t);
kfirst(tfirst) k(tfirst) =e= k0;
```

Here, the important boundary is the one at the beginning of the set rather than at the end. This can be expressed more compactly as

```
kk(t).. k(t) =e= k(t-1) + k0$tfirst(t) + i(t-1);
```

In general, the choice between using lag and lead operators as reference or in domain control is often a matter of taste.

13.6.3 Circular Lag and Lead Operators

In the case of circular lag and lead operators, the difference between its use in domain control and as reference is not important because it does not lead to any equations or terms being suppressed. Consider the following artificial example,

In this example, four individual examples are generated. They are listed below.

```
avail(summer) =e= prodn(spring) + sold(spring) ;
avail(autumn) =e= prodn(summer) + sold(summer) ;
avail(winter) =e= prodn(autumn) + sold(autumn) ;
avail(spring) =e= prodn(winter) + sold(winter) ;
```

Note that none of the equations are suppressed.

13.7 Summary

This chapter introduced the concept of ordering in sets. All the features in GAMS that dealt with this issue including the **ord** and **card** functions, as well as the linear and circular forms of the lag and lead operators were described in detail.

 $\mathbf{14}$

The Display Statement

14.1 Introduction

In this chapter we will provide more detail about display statements, including the controls that a user has over the layout and appearance of the output. These controls are a compromise to provide some flexibility. The display statement will not provide a publication quality reporting function, but is instead aimed for functionality that is easy to use, and provides graceful defaults. The execution of the display statement allows the data to be written into the listing file only.

14.2 The Syntax

In general, the syntax in GAMS for the display statement is:

display ident-ref | quoted text {, ident-ref | quoted text}

Ident-ref means the name without domain lists or driving indices of a set or parameter, or a sub-field of an equation or variable. The identifier references and the text can be mixed and matched in any order, and the whole statement can be continued over several lines.

The output produced by a display consists of labels and data. For sets, the character string **yes** (indicating existence) is used instead of values.

Solution Only the non-default values are displayed for all data types.

The default value is generally zero, except for the .lo and .up subtypes of variables and equations. The default values for these are shown in table 14.1.

14.3 An Example

An example of a display statement is given below.

The resulting listing file will contain the following section that corresponds to the display statement.

$\begin{array}{c c} & . \ lo & . \ up \\ \hline Variable & & & \\ & & \\ positive & 0 & +INF \\ free & -INF & +INF \\ negative & -INF & 0 \\ integer & 0 & 100 \\ binary & 0 & 1 \\ \hline Equation & & & \\ & =g= & 0 & +INF \\ =n= & -INF & +INF \\ =c= & 0 & +INF \\ =l= & -INF & 0 \\ =e= & 0 & 0 \end{array}$				
Variablepositive0+INFfree-INF+INFnegative-INF0integer0100binary01Equation=g=0+INF=n=-INF+INF=c=0+INF=l=-INF0=e=00			.lo	. up
$\begin{array}{c cccc} positive & 0 & +INF\\ free & -INF & +INF\\ negative & -INF & 0\\ integer & 0 & 100\\ binary & 0 & 1\\ \end{array}$ $Equation & & & \\ =g= & 0 & +INF\\ =n= & -INF & +INF\\ =c= & 0 & +INF\\ =l= & -INF & 0\\ =e= & 0 & 0 \end{array}$	Variable			
$ \begin{array}{cccccc} {\rm free} & -{\rm INF} & +{\rm INF} \\ {\rm negative} & -{\rm INF} & 0 \\ {\rm integer} & 0 & 100 \\ {\rm binary} & 0 & 1 \\ \end{array} \\ \hline Equation & & & \\ =g= & 0 & +{\rm INF} \\ =n= & -{\rm INF} & +{\rm INF} \\ =c= & 0 & +{\rm INF} \\ =l= & -{\rm INF} & 0 \\ =e= & 0 & 0 \\ \end{array} $		positive	0	+INF
$\begin{array}{cccc} negative & -INF & 0 \\ integer & 0 & 100 \\ binary & 0 & 1 \\ \\ Equation & & & \\ =g= & 0 & +INF \\ =n= & -INF & +INF \\ =c= & 0 & +INF \\ =l= & -INF & 0 \\ =e= & 0 & 0 \end{array}$		free	-INF	+INF
$\begin{array}{cccc} & \text{integer} & 0 & 100 \\ & \text{binary} & 0 & 1 \\ \hline \\ Equation & & & \\ =g= & 0 & +\text{INF} \\ =n= & -\text{INF} & +\text{INF} \\ =c= & 0 & +\text{INF} \\ =l= & -\text{INF} & 0 \\ =e= & 0 & 0 \end{array}$		negative	-INF	0
binary 0 1 Equation $=g= 0 +INF$ $=n= -INF +INF$ $=c= 0 +INF$ $=l= -INF 0$ $=e= 0 0$		integer	0	100
Equation=g=0+INF=n=-INF+INF=c=0+INF=l=-INF0=e=00		binary	0	1
=g= 0 +INF $=n= -INF +INF$ $=c= 0 +INF$ $=l= -INF 0$ $=e= 0 0$	Equation			
=n= -INF +INF $=c= 0 +INF$ $=l= -INF 0$ $=e= 0 0$		=g=	0	+INF
=c= 0 +INF =l= -INF 0 =e= 0 0		=n=	-INF	+INF
=1= -INF 0 =e= 0 0		=c=	0	+INF
=e= 0 0		=1=	-INF	0
		=e=	0	0

Table 14.1: Default values for .lo and .up subtypes

	5 first a s	set
	5 SET	S
S1, S2	2, S3,	S4
	5 then a pa	arameter
	5 PARAMETER	R P
S1 0.330,	S3 0.670	0
	5 then the	activity level of a variable
	5 VARIABLE	V.L
	T5	Τ7
S1	0.109	0.221
S3	0.221	0.449

Note that the only the non-zero values are displayed. In the case of multi-dimensional identifiers, the data is reported in a tabular form that is easy to read.

14.4 The Label Order in Displays

The default layout of a display for identifiers of different dimensionality is summarized in table 14.2. The figures in the table refer to the index position in the domain list of the identifier. As an example, if we display c, where c has been declared as c(i,j,k,l), then the i labels (the first index) will be associated with the planes or individual sub-tables, the j and k with the row labels, and the l (the fourth and last index) with the column headings.

Numbers of Indices	Plane	Index Position(s) on the Row	Column
1		List Format	1
2	-	1	2
3	-	1,2	3
4	1	2,3	4
5	1,2	$3,\!4$	5
6	1,2,3	4,5	6

Table 14.2: Default layout of display output

For 7 to 10 indices, the natural progression is followed. The labels vary slowest for the first index position, and
quickest for the highest. Within each index position the order is the GAMS entry order of the labels.

The order of the indices is always as in the declaration statement for the symbol. One can declare them in the order that is found appealing, or make an assignment to a new identifier with a different order.

The only way to change the order in which the labels for each index position appear on display output is to change the order of appearance of the labels in the GAMS program. This is most easily done by declaring a set whose only purpose is to list all the labels in the order that is needed. Make this set the very first declaration in the GAMS program.

14.4.1 Example

Consider the following example. X has four dimensions or index positions. It is initialized using parameter format and then displayed as shown below:

```
set
       i
           first index
                             /first, second /
           second index
                            /one, two, three /
       i
           third index
       k
                             /a, b /
           fourth index
                             /i, ii /
       1
                                      ;
parameter x(i,j,k,l) a four dimensional structure /
                                                           -6.3161
       second.one.a.i
                              +inf.
                                      first .three.b.i
                           5.63559,
                                       second.two .b.i
                                                           19.8350
       first .one.b.i
       second.one.b.ii
                         -17.29948.
                                      first .two
                                                   .b.ii
                                                           10.3457
       first .two.a.ii
                           0.02873,
                                      second.one
                                                   .a.ii
                                                            1.0037
       second.two.a.ii
                              +inf,
                                      first .two
                                                   .a.i
                                                           -2.9393
                           0.00000
                                                  /;
       first .one.a.ii
display x;
```

This code fragment produces the following output:

```
12 PARAMETER X
____
                                       a four dimensional structure
INDEX 1 = first
                   i
                                ii
one
     .b
               5.636
              -2.939
                            0.029
two
     .a
two
     .b
                            10.346
              -6.316
three.b
INDEX 1 = second
                             ii
                 i
              +INF
                          1.004
one.a
                        -17.299
one.b
two.a
                           +TNF
two.b
            19.835
```

Notice that there are two sub-tables, one for each label in the first index position. Note that the zero in the list for x('first', 'one', 'a', 'ii') has vanished, since zero values are suppressed in each sub-table separately. The order of the labels is not the same as in the input data list.

14.5 Display Controls

GAMS allows the user to modify the number of row and column labels in the display listing, as well as the accuracy of the data being displayed. The global display controls allows the user to affect more than one display statement. If specific data need to be listed in a particular format, the local display controls can be used to over-ride the global controls. The next two sub-sections will deal with each of these display controls in turn.

14.5.1 Global Display Controls

The simplest of these options is the one controlling the number of digits shown after the decimal point. It affects numbers appearing in all display output following the option statement, unless changed for a specific identifier as shown below. The general form of the statement is: 'option decimals = value;' where value is an integer between 0 and 8. If you use 0, the decimal point is suppressed as well. The width of the number field does not change, just the number of decimals, but this may cause numbers which would normally be displayed in fixed to appear in E-format, i.e., with the exponent represented explicitly.

Consider the following extension to the example discussed in the previous section.

```
option decimals = 1; display x ;
```

GAMS has rounded or converted numbers to E-format where necessary and the output is as follows:

```
____
         12 PARAMETER X
                                       a four dimensional structure
INDEX 1 = first
                   i
                               ii
                 5.6
one
     .b
                -2.9 2.873000E-2
two
     .a
two
    .b
                             10.3
three.b
                -6.3
INDEX 1 = second
                 i
                             ii
              +INF
                            1.0
one.a
one.b
                          -17.3
two.a
                           +INF
              19.8
two.b
```

14.5.2 Local Display Control

It is often more useful to control the number of decimals for specific identifiers separately. Using a statement whose general form can do this:

option ident:d-value:

Ident represent the name of a parameter, variable or equation, and d-value must be (as before) in the range 0 and 8. Exactly d-value places of decimals will be shown on all displays of ident that follow. This form can be extended to control layout of the data. The general form is:

option ident:d-value:r-value:c-value ;

Here **r-value** means the number of index positions that are combined to form the row label and **c-value** means the number on the column headers.

The example discussed in the previous section is further extended in order to illustrate the local display control.

option x :5:3:1; display x;

and the output:

---- 12 PARAMETER X a four dimensional structure

ii

i

first .one	.b	5.63559	
first .two	.a	-2.93930	0.02873
first .two	.b		10.34570
first .three	e.b	-6.31610	
second.one	.a	+INF	1.00370
second.one	.b		-17.29948
second.two	.a		+INF
second.two	.b	19.83500	

Five places of decimals are shown, and three labels are used to mark the rows and one on the column. Since this is a four-dimensional structure, there are no remaining indices to be used as sub-table labels (on the plane), and we now have the results in one piece. The option statement is checked for consistency against the dimensionality of the identifier, and error messages issued if necessary. Here is an example that puts two indices on each of the row and column labels, and retains five decimal places:

option x:5:2:2; display x ;

The output is :

	12 PARAMETER X	а	four dimens	ional structure
	a.i	a.ii	b.i	b.ii
first .on	e		5.63559	
first .tw	o -2.93930	0.02873		10.34570
first .th	ree		-6.31610	
second.on	e +INF	1.00370		-17.29948
second.tw	0	+INF	19.83500	

14.5.3 Display Statement to Generate Data in List Format

This is a special use of the local display controls to generate data in list format using the display statement. This is when all the labels are spelled out for each value as in the parameter style of data initialization. The format of the option is option:ident:d-value:0:c-value; and in this case the c-value specifies the maximum number of items displayed on a line. The actual number will depend on the page width and the number and length of your labels.

Using the same example as in the previous sections, the following extension:

option x:5:0:1; display x;

changes the output to look like below:

	12	PARAME	TER X	a	four	dimensional	structure
first	.one	.b.i	5.63559				
first	.two	.a.i	-2.93930				
first	.two	.a.ii	0.02873				
first	.two	.b.ii	10.34570				
first	.three	e.b.i	-6.31610				
second	.one	.a.i	+INF				
second	.one	.a.ii	1.00370				
second	.one	.b.ii	-17.29948				
second	.two	.a.ii	+INF				
second	.two	.b.i	19.83500				

This output nicely illustrates the label order used. The first index varies the slowest, the last the fastest, and each one runs from beginning to end before the next one to the left advances. This ordering scheme is also used on equation and column lists and on the solution report, all produced by the solve statement.

15

The Put Writing Facility

15.1 Introduction

In this chapter, the **put** writing facility of the GAMS language is introduced. The purpose of this writing facility is to output individual items under format control onto different files. Unlike the **display** statement, the entire set of values for indexed identifiers cannot be output using a single **put** statement (identifiers are the names given to data entities such as the names for **parameters**, **sets**, **variables**, **equations**, **models**, etc). While its structure is more complex and requires more programming than is required for the display statement, there is much greater flexibility and control over the output of individual items.

In this chapter, the working of the put writing facility is described as well as the syntax for accessing files and globally formatting documents using file suffixes for various attributes of a file. The put writing facility enables one to generate structured documents using information that is stored by the GAMS system. This information is available using numerous suffixes connected with identifiers, models, and the system. Formatting of the document can be facilitated by the use of file suffixes and control characters.

The put writing facility generates documents automatically when GAMS is executed. A document is written to an external file sequentially, a single page at a time. The current page is stored in a buffer, which is automatically written to an external file whenever the page length attribute is exceeded. Consequently, the put writing facility only has control of the current page and does not have the ability to go back into the file to alter former pages of the document. However, while a particular page is current, information placed on it can be overwritten or removed at will.

15.2 The Syntax

The basic structure of the put writing facility in its simplest form is:

file fname(s);
put fname;
put item(s);

where **fname** represents the name used inside the GAMS model to refer to an external file. Items are any type of output such as explanatory text, labels, parameters, variable or equation values. In the basic structure shown above, the first line defines the one or more files which you intend to write to. The second line assigns one of these defined files as the current file, that is the file to be written to. Lastly, the third line represents the actual writing of output items to the current file.

15.3 An Example

It is instructive to use a small example to introduce the basics of the put writing facility. The example will be based on the transportation model [TRNSPORT]. The following program segment could be placed at the end of the transportation model to create a report:

```
file factors /factors.dat/, results /results.dat/ ;
put factors ;
put 'Transportation Model Factors'///
    'Freight cost ', f,
    @1#6, 'Plant capacity'/;
loop(i, put @3, i.tl, @15, a(i)/);
put /'Market demand'/;
loop(j, put @3, j.tl, @15, b(j)/);
put results;
put results;
put 'Transportation Model Results'// ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

In the first line, the internal file names factors and results are defined and connected to the external file names factors.dat and results.dat. These internal file names are used inside the model to reference files, which are external to the model. The second line of this example assigns the file factors.dat as the current file, that is the file which is currently available to be written to.

In the third line of the example, writing to the document begins using a put statement with the textual item 'Transportation Model Factors'. Notice that the text is quoted. The slashes following the quoted text represent carriage returns. The example continues with another textual item followed by the scalar f. Notice that these output items are separated with commas. Blanks, commas, and slashes serve as delimiters for separating different output items. As mentioned above, the slash is used as a carriage return. Commas and blank spaces serve as item delimiters. These delimiters leave the cursor at the next column position in the document following the last item written. In most cases, the blank and the comma can be used interchangeably; however, the comma is the stronger form and will eliminate any ambiguities.

In the fifth line of the program above, the cursor is repositioned to the first column of the sixth row of the output file where another textual item is written. The cursor control characters **#** and **@** serve to reposition the cursor to a specific row or column as designated by the row or column number following the cursor control character. Lastly, the put statement is terminated with a semicolon.

Next, the parameters **a** and **b** are written along with their corresponding set labels. Only one element of the index set can be written using a put. To write the entire contents of the parameters **a** and **b**, the put statement is embedded inside a loop which iterates over the index set. In the example above, the set element labels are identified using their set identifier and the suffix .tl. As can be seen, the set element labels are located starting in the third column and the parameter **a** at column 15. The example continues with the display of another quoted textual item followed by the parameter **b**. When executed, the **factors.dat** file will look like:

Transportation Model Factors

Freight cost	90.00
Plant capacity seattle san-diego	350.00 600.00
Market demand	
new-york	325.00
chicago	300.00
topeka	275.00

This output has been formatted using the default file format values. The methods to change these defaults will be described later in this chapter.

In the last two lines of the example, the file results.dat is made current and the values associated with the variable x along with their corresponding set element index labels are written line by line. The output results of

the variable \mathbf{x} are formatted by specifying a field width of eight spaces with four of these spaces reserved for the decimal. Notice that the local formatting options are delimited with colons. The results.dat file will look like:

Transportation Model Results

seattle	new-york	0.0000
seattle	chicago	300.0000
seattle	topeka	0.0000
san-diego	new-york	325.0000
san-diego	chicago	0.0000
san-diego	topeka	275.0000

With just this brief introduction to the put writing facility, it is easy to envision its many uses such as report writing, providing output to a file for use by another computer program, or simply the display of intermediate calculations. But, the surface of the put writing facility has just barely been scratched. In the sections that follow, the many features and structure of the put writing facility are described in more detail, along with examples.

15.4 Output Files

As noted earlier, the put statement allows the user to write to external files. This section describes the various features related to the use of external files.

15.4.1 Defining Files

The complete syntax for defining files is:

```
file fname text / external file name /
```

where file is the keyword used to define files. Fname is the internal file name and is used inside the GAMS model to refer to an external file. External files are the actual files that output is written to. During file declaration, the external file name and explanatory text are optional. When the external file name is omitted, GAMS will provide a system specific default external file name, often fname.put. Note that multiple files can be defined using a single file statement. Consider the following example:

file class1
 class2 this defines a specific external file /report.txt/
 con this defines access to the console (screen) for PC systems;

The first output file is recognized in the model by the name class1 and corresponds to the default file class1.put for a PC system. The second output file is recognized in the model by the name class2 and it corresponds to the defined external file report.txt. Lastly, the special internal file name con is defined to write output to the console (screen) for a PC systems. Writing to the screen can be useful to advise the user of various aspects of the model during the model's execution.

15.4.2 Assigning Files

The put statement is used both to assign the current file and to write output items to that file. The complete syntax for using the put statement is:

put fname item(s) fname item(s) . . . ;

As indicated by this syntax, multiple files can be sequentially written using a single **put** statement. Note that only one file is current at a time. After the output items following an internal file name are written, the current file is reassigned based on the next internal file name in the statement. The last internal file name used in a **put** statement remains as the current file until a subsequent **put** statement uses an internal file name.

15.4.3 Closing a File

The keyword putclose is used to close a file during the execution of a GAMS program. The syntax is as follows:

putclose myfile item(s)

where myfile is the internal name of the file to be closed, and item(s) are the final entries into the file before it is closed. If the internal file name is omitted from the putclose statement, the current put file is closed. Note that after using the putclose command, the file does not have to be redefined in order to use it again. Simply make the file current and use put statements as would be done normally. Of course, the existing file will either be overwritten or appended to depending on the value of the append file suffix.

One application where this is useful is to write the solver option file from within the GAMS model. Option file statements can be written using put and the file closed with a putclose prior to the solve statement. This makes the option file available for use by the solver.

The following example shows the creation and closing of an option file for the MINOS solver:

file opt Minos option file / minos.opt /;
put opt;
put 'Iteration limit 500'/
 'Feasibility tolerance 1.0E-7'/;
putclose opt;

This program segment would be placed inside the GAMS model prior to the solve statement.

15.4.4 Appending to a File

The put writing facility has the ability to append to or overwrite an existing file. The file suffix .ap determines which operation occurs. The default suffix value 0 overwrites the existing file while the value 1 causes appending to the file. Let's consider our report.txt file to be an existing file.

Using the following statement appends output items to it:

class2.ap = 1;

Any items put into report.txt will from that point on be added to the end of the existing file contents. If the file had not existed, the file would be created.

15.5 Page Format

The pages within files can also be structured using file suffixes to specify many attributes such as the printing format, page size, page width, margins, and the case which text is displayed in. The following file suffixes can be used for formatting:

- print control (.pc) Used to specify the format of the external file. The options 4,5,6, and 8 create delimited files, which are especially useful when preparing output for the direct importation into other computer programs such as spreadsheets.
 - 0 Standard paging based on the current page size. Partial pages are padded with blank lines. Note that the .bm file suffix is only functional when used with this print control option.
 - 1 FORTRAN page format. This option places the numeral one in the first column of the first row of each page in the standard FORTRAN convention.
 - 2 Continuous page (default). This option is similar to .pc option zero, with the exception that partial pages in the file are not padded with blank lines to fill out the page.
 - 3 ASCII page control characters inserted.

- 4 Formatted output; Non-numeric output is quoted, and each item is delimited with a blank space.
- 5 Formatted output; Non-numeric output is quoted, and each item is delimited with commas.
- 6 Formatted output; Non-numeric output is quoted, and each item is delimited with tabs.
- 7 Fixed width; Fills up line with trailing blanks.
- 8 Formatted output; Each item is delimited with a blank space.
- page size (.ps) Used to specify the number of rows (lines) which can be placed on a page of the document. Can be reset by the user at any place in the program. However, an error will result if set to a value less than the number of rows which have already been written to the current page. Maximum value is 130. The default value is 60
- page width (.pw) Used to specify the number of columns (characters) which can be placed on a single row of the page. Can be reset by the user at any place in the program. However, an error will result if set to a value less than the number of rows or columns which have already been written to the current page. The default value is 255.
- top margin (.tm) Number of blank lines to be placed at the top margin of the page. These lines are in addition to the number of lines specified in the .ps file suffix. Default value is 0.
- bottom margin (.bm) Number of blank lines to be placed in the bottom margin of the page. These lines are in addition to the number of lines specified in the .ps file suffix. This is functional with .pc option 0 only. Default value is 0.
- alphabetic case (.case) Used to specify the case in which alphabetic characters are displayed in the output file.
 - 0 (default) Causes mixed case to be displayed.
 - 1 Causes the output to be displayed in upper case regardless of the case used for the input.

To illustrate the use of these file suffixes, the following example involves formatting report.txt so that the pages are 72 spaces wide with 58 lines of output, an additional top margin of 6 lines, using ASCII page control characters (inserted every 64 lines), and with the output displayed in upper case.

```
file class2 /report.txt/ ;
class2.pw = 72; class2.ps = 58; class2.tm = 6;
class2.pc = 3; class2.case = 1;
```

Using a value of 4, 5, or 6 for the print control suffix (.pc) will cause data to be squeezed and therefore will ignore spacing information provided by the user through the @ character. However, these values can be used to pass data on to be read by spreadsheets.

15.6 Page Sections

There are three independent writing areas on each page of a document. These areas are the title block, the header block, and the window. This is quite useful when there are sections of a page which remain relatively constant throughout a document. Title and header blocks are often used to provide organizational information in a document with the window being used for specific reporting.

These writing areas are always sequentially located on the page in the order shown on the following diagram. It is important to note that the title and header blocks are essentially the same as the window and use exactly the same syntax rules. However, the window is required in each page of your document, while the title and headers are optional. Also note that once the window is written to, any further modifications of the title or header blocks will be shown on subsequent pages and not the current page. Writing to the window is what ultimately forces a page to be written



In the illustrative example described in Section 15.3, all the data was written to the window. A title block might have been included, if more elaboration were needed, to provide the model name along with the page number. In addition, a header block might have been used to display a disclaimer or an instruction, which we wanted consistently, repeated on every page. Once this information is placed in the title or header blocks, it is displayed on each page thereafter unless modified. This could be especially useful for a long document covering many pages.

15.6.1 Accessing Various Page Sections

Each of these areas of a page are accessed by using different variations of the keyword put. These variations are:

puttl	write to title block
puthd	write to header block
put	write to window

The size of any area within a given page is based entirely on the number of lines put into it. Note that the total number of lines for all areas must fit within the specified page size. If the total number of lines written to the title and header block equals or exceeds the page size, an overflow error will be displayed in the program listing. When this occurs, this means there is no room remaining on the page to write to the window.

As mentioned above, the syntax for writing an output item to any of the three possible writing areas of the page is basically the same, the only difference being the choice of put keyword. This is illustrated by writing to the title block of our report.dat file:

puttl class2 'GAMS Put Example' ;

In this case, the text 'GAMS Put Example' has been placed in the first column of the first row of the title block. Any subsequent pages in the report.dat file will now start with this information.

If the title block was modified or the header block was started after the window of the current page has been written to, these modifications would appear in the next page and not the current page.

15.6.2 Paging

Paging occurs automatically whenever a page is full. However, note that the window must be used in order for the page to be written to the output file. When a page has no output in its window, the page is not written to file regardless of whether there are output items in the title or header blocks. To force a page that has an empty window out to file, simply write something innocuous to the window such as:

put '';

Now the window of the page has been initiated and it will be written.

15.7 Positioning the Cursor on a Page

The cursor is positioned at the space immediately following the last character written unless the cursor is specifically moved using one of the following cursor control characters:

#n Move cursor position to row **n** of current page

- **@n** Move cursor position to column **n** of current line
- / Move cursor to first column of next line. Also acts as a delimiter between output items

In addition to numerals, any expression or symbol with a numeric value can be used to follow the # and @ characters. The following example illustrates the use of these position controls to write out the value of a parameter a(i,j) in a tabular form:

```
file out; put out;
scalar col column number /1/ ;
loop(i,
        loop (j, put @col a(i,j); col=col+10; ) ; put / ;
        );
```

15.8 System Suffixes

The complete list of system suffixes that can be used to recover information about the GAMS run are:

.date	program execution date
.ifile	input file name
.ofile	output file name
.page	current file page
.rdate	restart file date
.rfile	restart file name
.rtime	restart file time
.sfile	save file name
.time	program execution time
.title	title of the model as specified by \texttt{title}

As an illustration, consider the example discussed in the previous section. One can add page numbers to the title of the report file by modifying the puttl statement to read

puttl class2 'GAMS Put Example', @65,'page ',system.page ///;

This causes the word page followed by the page number to appear on the title of every page starting at column 65.

15.9 Output Items

Output items for the **put** statement are of the following forms:

- *text* Any quoted text, set element label or text, any identifier symbol text or contents of the system suffixes.
- numeric Values associated with parameters, variables, equations, or any of the model suffixes.

set values Represent existence of set elements and carry the values yes or no only.

The methods for identifying and using each of these different types of output items are described in the following sub-sections.

15.9.1 Text Items

Output items, which are quoted text, are any combination of characters or numbers set apart by a pair of single or double quotes. However, the length of quoted text, as well as any output item, has a limit. No portion of the output item can be placed outside of the page margin. When the page width is exceeded, several asterisks are placed at the end of the line and a **put error** is recorded in the program listing.

In addition to quoted text, the output of other text items is possible through the use of system and identifier suffixes. The identifier suffixes are:

identifier symbol text (.ts) Displays the text associated with any identifier

set element labels (.tl) Displays the individual element labels of a set

set element text (.te(index)) Displays the text associated with an element of a set. Notice that the .te
suffix requires a driving index. This driving index controls the set, which will be displayed and does not
necessarily have to be the same as the controlled set. Often a subset of indices of the controlled set is
used.

text fill (.tf) Used to control the display of missing text for set elements.

- 0 no fill
- 1 fill existing only
- 2 (default) fill always

The following example illustrates these ideas:

The resulting file out.put will look like:

subset of sites
i3 San Francisco
i4 Los Angeles
i5 i5

In this example, the symbol text for the identifier of the subset j is written first. This is followed with the labels for the subset j and the associated element text found in its domain, that is, the set i. Notice the driving set j is used for the element text specification of the set i. Since there was no set element text associated with the i5 element of set i, the set element label was displayed again. By placing the following before the last line:

out.tf = 0;

The missing element text is now no longer replaced with the label text. The resulting file **out.put** file would now look like:

subset of sites i3 San Francisco i4 Los Angeles i5

15.9.2 Numeric Items

The syntax used for the display of numeric items is generally easier to work with. To output a parameter, only the identifier along with its index set (as appropriate) has to be used. To output a variable or equation value, the identifier is combined with one of the variable and equation suffixes. The variable and equation suffixes are:

.1	level or marginal value
.lo	lower bound
.m	marginal or dual value
.prior	priority
.scale	scaling
.up	upper bound

15.9.3 Set Value Items

Set value items are easy to work with. To output the set value, only the identifier along with its index set has to be used. In the example from Section 15.9.1, consider altering the loop statement to read:

loop(i, put i.tl, j(i), ' ',i.te(j) /);

The resulting output file looks like follows:

${\tt subset}$	of si	ites
i1	NO	Seattle
i2	NO	Portland
i3	YES	San Francisco
i4	YES	Los Angeles
i5	YES	

The second columns represents whether the element belongs to set j or not.

15.10 Global Item Formatting

It is often important to be able to control the display format of output items. In this section we describe how this is done. For formatting purposes, output items are classified into four categories. These are labels, numeric values, set values, and text. For each, global formatting of the field width and field justification is possible.

15.10.1 Field Justification

The possible global justifications are *right* (value 1), *left* (value 2), and *center* (value 3). The field justification is represented by the following file suffixes:

- .1j label justification (default 2)
- .nj numeric justification (default 1)
- .sj set value justification (default 1)
- .tj text justification (default 2)

15.10.2 Field Width

This is done using the following file suffixes:

.lw	label field width (default 12)
.nw	numeric field width (default 12)
.SW	set value field width (default 12), (maximum 20)
.tw	text field width (default 0)

The field width is specified with the number of spaces to be allocated to the field. Variable length field widths are possible by using a suffix value of 0. This forces the field width to match the exact size of the item being displayed. If a textual output item does not fit within the specified field, truncation occurs to the right. For numeric output items, the decimal portion of a number is rounded or scientific notation used to fit the number within the given field. If a number is still too large, asterisks replace the value in the output file.

As an example, to set the global numeric field width to four spaces from its default of 12 in the file out.put, we would use the following statement:

out.nw = 4;

15.11 Local Item Formatting

It is often useful to format only specific **put** items. For this, we use the local format feature, which overrides global format settings. The syntax of this feature is as follows:

item:{<>}width:decimals;

The item is followed by a justification symbol, the field width, and the number of decimals to be displayed. The specification of the number of decimals is only valid for numeric output. The following local justification symbols are applicable:

- > right justified
- < left justified

put f:<6:2 ;

<> center justified

Omitting any of the components causes their corresponding global format settings to be used. As with global formatting, when the field width is given a value of 0, the field width is variable in size. The item, width, and decimals are delimited with colons as shown above. The use of the local format feature as well as the inclusion any of the components for justification, field width, or the number of decimals is entirely optional.

The following example shows some examples of the local formatting feature:

```
* default justification and a field width of variable size
* with no decimals
loop(i, put dist(i):0:0 /);
put 'Right justified comment':>50,
    'Center justified truncated comment':<>20;
* left justified scalar with a six space field width and
* two decimals
```

15.12 Additional Numeric Display Control

In addition to the numeric field width and the numeric justification as mentioned in the previous section, the following file suffixes can also be globally specified for numeric display:

- number of decimals displayed (.nd) Sets the number of decimals displayed for numeric items. A value of 0 results in only the integer portion of a number being displayed. The maximum value is 10. The default value is 2.
- numeric round format (.nr) Allows one to display a numeric value in scientific notation, which would otherwise be displayed as zero because of being smaller than the number of decimals allowed by the .nd suffix. This situation occurs when a number is smaller than the .nd specification, but is larger than the zero tolerance level set by .nz. In many situations, it is important to know that these small values exist. The default is 1.
 - 0 displayed in F or E format
 - 1 rounded to fit fields
 - 2 displayed in scientific notation
- numeric zero tolerance (.nz) Sets the tolerance level for which a number will be rounded to zero for display purposes. When it is set equal to zero, rounding is determined by the field width. Default value is 1.0e-5.

The maximum size of a displayed number must fit within 20 spaces using at most 10 significant digits. The remaining 10 spaces are used for the sign, exponential notation, or padding with zeros.

15.12.1 Illustrative Example

The following illustrative example shows the results of different combinations of these numeric file suffixes. The example uses five combinations of the numeric file suffixes .nd, .nz, .nr, and .nw. Four number values, each of which is shifted by three decimal places from its predecessor, are used with these suffix combinations. The combinations are chosen to show various format results when these suffix values are used together in put statements:

```
c suffix combinations /
                                 comb1 * comb4
set
                                                  1
        value indices
                                 value1* value3
                                                 /;
     v
                              1
        suffix(c.*)
                      numeric suffix combinations
table
           nd
                   nz
                           nr
                                    nw
                    0
                            0
            3
                                    12
comb1
comb2
            3
                 1e-5
                             0
                                    12
comb3
            3
                 1e-5
                             1
                                    12
                    0
comb4
            8
                             0
                                    10
comb5
            6
                 1e-5
                             2
                                    12;
parameter value(v)
                      test values
      / value1
                   123.4567
                     0.1234567
        value2
        value3
                     0.0001234567 / ;
file out; put out; out.nj=2; out.lw=10;
out.cc=11:
loop(v, put v.tl:21);
loop(c, out.nd=suffix(c,"nd");
     out.nz=suffix(c,"nz");
     out.nr=suffix(c,"nr");
     out.nw=suffix(c,"nw");
     put / c.tl;
loop(v, put @(ord(v)*21-10), value(v)));
```

For readability, the numeric values have purposely been made left justified using the .nj suffix since the numeric field width is changed as the model goes through the suffix combinations. The following is the resulting file out.put, which shows the value/suffix combinations:

alue1	value2	value3
23.457	0.123	1.2345670E-4
23.457	0.123	1.2345670E-4
23.457	0.123	0.000
.23457E+2	0.12345670	0.00012346
23.456700	0.123457	0.000123
	alue1 23.457 23.457 23.457 2.23457 23.457 23.456700	alue1 value2 23.457 0.123 23.457 0.123 23.457 0.123 23.457 0.123 .23457E+2 0.12345670 23.456700 0.123457

Notice that in comb1, the display of values switch to exponential notation when a value becomes smaller than the number of decimal places allowed. This is triggered by the suffix .nr being set to zero. Of particular interest is value3 for comb2 and comb3. Value3 is greater than the zero tolerance level in .nz, but smaller than the number of decimals allowed by .nd. In comb2, since .nr is set to zero, the value is displayed in exponential format. In comb3, .nr is set to 1, so this small value is rounded to 0. In comb5, value1 is rounded to an integer because of .nd being set to 0.

15.13 Cursor Control

Having described the display of various output items using the **put** statement, this section describes features available to position these items in the output file. GAMS has several file suffixes which determine the location of the cursor and the last line of the file. These suffixes can also be used to reposition the cursor or reset the last line. As such, they are instrumental in formatting output items in documents. These suffixes are grouped by the title, header, or window writing area for which they are valid.

15.13.1 Current Cursor Column

These suffixes have numeric values corresponding to coordinates in the window of the page. Because of this, they can be used in conjunction with cursor control characters to manipulate the position of the cursor in the output file.

- .cc current cursor column in window
- .hdcc header current column
- .tlcc title current column
- The convention for updating the values stored for the .cc suffix is that it are updated at the conclusion of a put statement. Consequently, the .cc value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed.

The following example illustrates the updating of the cursor control suffixes and the use of cursor control characters. The example is trivial but instructive:

```
scalar lmarg left margin /6/;
file out; put out;
put @(lmarg+2) 'out.cc = ', out.cc:0:0 ' ';
put @out.cc 'x'/ @out.cc 'y'/ @out.cc 'z ';
put 'out.cc = ' out.cc:0:0;
```

The following is the resulting file out.put:

```
out.cc = 1 x
y
z out.cc = 23
```

Initially, the scalar lmarg is set to a specific value to use as an alignment tab. Symbols which hold common alignment values such as margins or tabs are often useful for large structured documents. The first put statement uses the current column cursor control character to relocate the cursor. In this example, the cursor is moved to column 8where out.cc and its value is displayed.

The second put statement illustrates the updating of the cursor control suffixes by writing the letters x, y, and z on three different lines. Each is preceded by the cursor being moved to the out.cc value. Initially, the value for the cursor control suffice is 20. Since a single put statement is used for these three items, the out.cc value remains constant and consequently the letters end up in the same column. Following this put statement, the out.cc value is updated to 23, which is the location of the cursor at the end of the second put statement (note the additional blank spaces displayed with the letter z).

15.13.2 Current Cursor Row

These suffixes have numeric values corresponding to coordinates in the window of the page. Because of this, they can be used in conjunction with cursor control characters to manipulate the position of the cursor in the output file.

.cr	current cursor row in window
.hdcr	header current row
.tlcr	title current row

The convention for updating the values stored for the .cr suffix is that it are updated at the conclusion of a put statement. Consequently, the .cr value remains constant throughout the writing of items for the next put statement, even if multiple items are displayed. It's behavior is similar to that of .cl.

15.13.3 Last Line Control

These suffixes control the last line used in a writing area.

.11	last line used in window
.hdll	header last line

.tlll title last line

Unlike the row and column control, the last line suffix is updated continuously. Last line suffixes are especially useful for modifying the various writing areas of a page.

- The .tlll and .hdll suffixes may not hold values applicable to the current page because when the title or header blocks are modified, they correspond to the title or header blocks of the next page whenever the window has been written to on the current page.
- Not only can this suffix be used to determine the last line used in a writing area, but it can also be used to delete lines within this area.

In the following example, the header section will be completely deleted by resetting the .hdll suffix to 0.

```
file out;
puthd out 'This header statement will be eliminated';
out.hdll = 0;
```

In this example, a header is initially written. By changing the .hdll suffix to 0, the cursor is reset to the top of the header block. Consequently the header will not be written unless something new is added to the header block.

15.14 Paging Control

In addition to the automatic paging, which occurs when the bottom of the page is reached, a page can also be written to file early. The keyword **putpage** is used to do this. **Putpage** forces the current page to immediately be written to file, making a new page available for **put** statements. In its simplest form, the keyword **putpage** is used by itself to eject the current page. Additionally, it can be used with output items. When it is used with output items, the page is written to file including the output items contained in the **putpage** statement. The **putpage** statement is in fact another variation of the **put** statement. In the following statement, the quoted text is placed in the current page, which is then written to the file **out.put**:

putpage out 'This text is placed in window and the page ends';

Two additional file suffixes that can help the user in determining when to page a file are:

- last page (.1p) Indicates the number of pages that are already in the document. Note that setting this to 0 does not erase the pages that have previously been written to the file.
- window size (.ws) Shows the number of rows, which can be placed in the window, considering the number of lines that are in the title and header blocks of the current page and the existing page size. The .ws file suffix value is calculated by GAMS and is not changeable by the user. This suffix is useful for manual pagination when used in conjunction with the .11 file suffix.

15.15 Exception Handling

In this section, the topic of exception handling is dealt with. As with other GAMS statements, dollar control exception handling can be used with **put** statements to control whether particular output items are displayed. In the following example, the put statement is only displayed if the dollar condition is true. If it is not, the put statement is ignored:

```
put$(flag gt 10) 'some output items';
```

15.16 Source of Errors Associated with the Put Statement

There are two types of errors that can occur when using the put writing facility: syntax errors and **put** errors. The following subsections discuss each of these types of errors.

15.16.1 Syntax Errors

Syntax errors are caused by the incorrect usage of the GAMS language. These errors are the same or are similar to what one finds elsewhere with GAMS such as unmatched parentheses, undefined identifiers, uncontrolled sets, or the incorrect use of a keyword or suffix. These errors are detected during program compilation and are always fatal to program execution. Errors of this kind are identified in the program listing at the location of the error with a \$ symbol and corresponding error numbers. The program listing includes a brief description of the probable cause of the error.

15.16.2 Put Errors

Put errors are unique to the put writing facility. This type of error occurs during program execution and is caused when one or more of the file or page attributes are violated. These errors are non-fatal and are listed at the end of the program listing. They typically occur when a put statement attempts to write outside of a page, such as moving the cursor with the **@** character to a location beyond the page width. Other typical errors are the inability to open a specified file, the overflow of a page, or an inappropriate value being assigned to a suffix. For many of these errors, an additional set of asterisks will be placed at the location of the error in the output file.

Since put errors are non-fatal and are not overemphasized in the output file, their presence is sometimes overlooked. Without reviewing the program listing, these put errors might go undetected, especially in large output files. Consequently, GAMS has included the following file suffix to help one detect errors:

.errors Allows one to display the number of put errors occurring in a file.

To illustrate its use, the following statement could be inserted at any point of a program to detect the number of errors, which have occurred up to its location. The choice of output file could be the same file, a different file, or the console as appropriate:

```
putpage error ///'*** put errors: ', out.errors:0:0,' ***'/;
```

In this example it is assumed that the files out.put and error.put have previously been defined with a file statement. With this statement, the number of put errors that occur in the file out.put are displayed in the file error.put. Using putpage would allow the immediate display to the screen of a PC system at the location of this statement if the console had been the output device.

15.17 Simple Spreadsheet/Database Application

This last section provides a simple example of the preparation of output for spreadsheets, databases, or other software packages, which allow importation of delimited files. As mentioned in Section 15.3, output items can be prepared with comma delimiters and text items in quotes. This is implemented by using .pc suffix value 5. Delimited files are different than normal put files. All output items are written with variable field widths and separated by delimiters. Consequently, all global and local format specifications for field widths and justification are ignored by GAMS. Note that the number of decimals for numeric items can still be specified with the .nd file suffix. Each item is written immediately following the previous delimiter on the same line unless the cursor is reset.

Avoid horizontal cursor relocations in a program, which creates a delimited file. Horizontally relocating the cursor in a delimited file is potentially damaging since a delimiter could be overwritten.

While the comma is the most common delimiting character for spreadsheets, other delimiters like blank space and tab characters can also be used.

15.17.1 An Example

In the following example, the capacity sub-table of the [MEXSS] report program is prepared as a delimited file. The following program segment demonstrates .pc suffix value 5. The program segment could be placed at the end of the original [MEXSS] model:

The first line of this program segment creates the file **out.put** as the delimited file. Notice that in the remainder of this program, field widths, justifications, and horizontal cursor relocations are completely avoided. All text items are quoted. The following is the resulting output file:

```
"CAPACITY (tons)","AHMSA","FUNDIDORA","SICARTSA","HYLSA","HYLSAP"
"BLAST FURNACES",3.25,1.40,1.10,0.00,0.00
"oPEN HEARTH FURNACES",1.50,0.85,0.00,0.00,0.00
"BASIC OXYGEN CONVERTERS",2.07,1.50,1.30,0.00,0.00
"DIRECT REDUCTION UNITS",0.00,0.00,0.00,0.98,1.00
"ELECTRIC ARC FURNACES",0.00,0.00,0.00,1.13,0.56
```

Notice that each item is delimited with a comma and that textual output is quoted.

16

Programming Flow Control Features

16.1 Introduction

The previous chapters have focused on the ability of GAMS to describe models. This chapter will describe the various programming features available in GAMS to help the advanced user. The various programming flow control features discussed in this chapter are

Loop Statement For Statement If-Else Statement While Statement

Each of these statements will be discussed in detail in the following sections.

16.2 The Loop Statement

The loop statement is provided for cases when parallel assignments are not sufficient. This happens most often when there is no analytic relationship between, for example, the values to be assigned to a parameter. It is, of course, also useful to have a looping statement for general programming - for example, the production of reports with the **put** statement.

16.2.1 The Syntax

The syntax of the loop statement is,

```
loop(controlling_domain[$(condition)],
    statement {; statement}
    ;
```

If the controlling_domain consists of more than one set, then parentheses are required around it.

The loop statement causes GAMS to execute the statements within the scope of the loop for each member of the driving set(s) in turn. The order of evaluation is the entry order of the labels. A loop is thus another, more general, type of indexed operation. The loop set may be dollar-controlled and does not need to be static or nested. Loops may be controlled by more than one set.

Solutions One cannot make declarations or define equations inside a loop statement.

It is illegal to modify any controlling set inside the body of the loop.

16.2.2 Examples

Consider a hypothetical case when a growth rate is empirical:

```
set t / 1985*1990 /
parameter pop(t) / 1985 3456 /
growth(t) / 1985 25.3, 1986 27.3, 1987 26.2
1988 27.1, 1989 26.6, 1990 26.6 /;
```

The loop statement is then used to calculate the cumulative sums

```
loop(t, pop(t+1) = pop(t) + growth(t));
```

in an iterative rather than a parallel way. In this example there is one statement in the scope of the loop, and one driving, or controlling, set.

A loop is often used to perform iterative calculations. Consider the following example, which uses finds square roots by Newton's method. This example is purely for illustration - in practice, the function sqrt should be used. Newton's method is the assertion that if x is an approximation to the square root of v, then (x + v/x)/2 is a better one

```
set i "set to drive iterations" / i-1*i-100 /;
parameter value(i) "used to hold successive approximations" ;
scalars
                "number whose square root is needed" /23.456 /
        target
                "final approximation to sqrt(target)"
        sgrtval
        curacc
                "accuracy of current approximation"
                "required relative accuracy" / 1.0e-06 / ;
       reltol
abort$(target <= 0) "argument to newton must be positive", target;</pre>
value("i-1") = target/2 ; curacc = 1 ;
loop(i$(curacc > reltol),
    value(i+1) = 0.5*(value(i) + target/value(i));
    sqrtval = value(i+1);
    curacc = abs (value(i+1)-value(i))/(1+abs(value(i+1)))
);
abort$(curacc > reltol) "square root not found"
option decimals=8:
display "square root found within tolerance", sqrtval, value;
```

The output is:

---- 18 square root found within tolerance ---- 18 PARAMETER SQRTVAL = 4.84313948 final approximation to sqrt(target) ---- 18 PARAMETER VALUE used to hold successive approximations i-1 11.72800000, i-2 6.86400000, i-3 5.14062471, i-4 4.85174713 i-5 4.84314711, i-6 4.84313948, i-7 4.84313948

16.3 The If-Elseif-Else Statement

The **if-else** statement is useful to branch conditionally around a group of statements. In some cases this can also be written as a set of dollar conditions, but the **if** statement may be used to make the GAMS code more readable. An optional **else** part allows you to formulate traditional **if-then-else** constructs.

16.3.1 The Syntax

The syntax for an if-elseif-else statement is:

```
if (condition,
    statements;
{elseif condition, statements; }
[else statements;]
);
```

where condition is a logical condition.

IF One cannot make declarations or define equations inside an if statement.

16.3.2 Examples

Consider the following set of statements

```
\begin{array}{l} p(i)\$(f <= 0) = -1 ;\\ p(i)\$((f > 0) \mbox{ and } (f < 1)) = p(i)**2 ;\\ p(i)\$(f > 1) = p(i)**3 ;\\ q(j)\$(f <= 0) = -1 ;\\ q(j)\$((f > 0) \mbox{ and } (f < 1)) = q(j)**2 ;\\ q(j)\$(f > 1) = q(j)**3 ; \end{array}
```

They can be expressed using the if-elseif-else statement as

```
if (f <= 0,
    p(i) = -1;
    q(j) = -1;
elseif ((f > 0) and (f < 1)),
    p(i) = p(i)**2;
    q(j) = q(j)**2;
else
    p(i) = p(i)**3;
    q(j) = q(j)**3;
);
```

The body of the **if** statement can contain **solve** statements. For instance, consider the following bit of GAMS code:

```
if ((ml.modelstat eq 4),
* model ml was infeasible
* relax bounds on x and solve again
    x.up(j) = 2*x.up(j);
    solve ml using lp minimizing lp;
else
    if ((ml.modelstat ne 1),
        abort "error solving model ml;
    );
);
```

The following GAMS code is illegal since one cannot define equations inside an if statement.

```
if (s gt 0,
     eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is illegal since one cannot make declarations inside an if statement.

16.4 The While Statement

The while statement is used in order to loop over a block of statements.

16.4.1 The Syntax

The syntax of the while statement is:

```
while(condition,
    statements;
);
```

Solution of the statement.

16.4.2 Examples

One can use while statements to control the solve statement. For instance, consider the following bit of GAMS code that randomly searches for a global optimum of a non-convex model:

In this example, a non-convex model is solved from 1000 random starting points, and the global solution is tracked. The model [PRIME] from the model library illustrates the use of the while statement through an example where the set of prime numbers less than 200 are generated

The following GAMS code is illegal since one cannot define equations inside a while statement.

The following GAMS code is illegal since one cannot make declarations inside a while statement.

16.5 The For Statement

The for statement is used in order to loop over a block of statements.

16.5.1 The Syntax

The syntax is:

Note that i is not a set but a parameter. Start and end are the start and end, and incr is the increment by which i is changed after every pass of the loop.

- IF One cannot make declarations or define equations inside a for statement.
- The values of start, end and incr need not be integer. The start and end values can be positive or negative real numbers. The value of incr has to be a positive real number.

16.5.2 Examples

One can use **for** statements to control the **solve** statement. For instance, consider the following bit of GAMS code that randomly searches for a global optimum of a non-convex model:

In this example, a non-convex model is solved from 1000 random starting points, and the global solution is tracked. The use of real numbers as **start**, **end** and **incr** can be understood from the following example,

The resulting listing file will contain the following lines,

 2 PARAMETER S	=	-3.400
 2 PARAMETER S	=	-2.000
 2 PARAMETER S	=	-0.600

Notice that the value of s was incremented by 1.4 with each pass of the loop as long as it did not exceed 0.3. The following GAMS code is illegal since one cannot define equations inside a for statement.

The following GAMS code is illegal since one cannot make declarations inside a for statement.

17

Special Language Features

17.1 Introduction

This chapter introduces special features in GAMS that do not translate across solvers, or are specific to certain model types. These features can be extremely useful for relevant models, and are among the most widely used.

17.2 Special MIP Features

Some special features have been added to GAMS to help in simplifying the modeling of MIP problems. Two special types of discrete variables are defined and discussed. Finally, creating priorities for the discrete variables is discussed. The solvers use this information when solving the problem.

17.2.1 Types of Discrete Variables

The following types of discrete variables have been discussed so far in the book,

binary variables These can take on values of 0 or 1 only.

integer variables These can take on integer values between the defined bounds. The default lower and upper bounds are 0 and 100 respectively.

In addition to these two, two new types of discrete variables that are introduced in this section. Both these variables exploit special structures in MIP models during the solution phase. These are the following

- Special Ordered Sets (SOS) The precise definition of special ordered sets differ from one solver to another and the development of these features has been driven more by internal algorithmic consideration than by broader modeling concepts. GAMS offers sos1 and sos2 variables as two types of compromise features that model special ordered sets. Sections 17.2.2 and 17.2.3 discuss these two types of variables in greater detail.
- Semi-continuous variables GAMS offers semicont and semiint variables to model this class of variables. These are explained in Sections 17.2.3 and 17.2.4.

The presence of any of the above types of discrete variables requires a mixed integer model and all the discreteness is handled by the branch and bound algorithm in the same way as binary and general integer variables are handled.

17.2.2 Special Order Sets of Type 1 (SOS1)

At most one variable within a SOS1 set can have a non-zero value. This variable can take any positive value. Special ordered sets of type 1 are defined as follows,

sos1 Variable s1(i), t1(k,j), w1(i,j,k) ;

The members of the innermost (the right-most) index belongs to the same set. For example, in the sets defined above, s1 represents one special ordered set of type 1 with i elements, t1 defines k sets of j elements each, and w1 defines (i,j) sets with k elements each.

- The default bounds for SOS1 variables are 0 to $+\infty$. As with any other variable, the user may set these bounds to whatever is required.
- The user can, in addition, explicitly provide whatever convexity row that the problem may need through an equation that requires the members of the SOS set to be less than a certain value. Any such convexity row would implicitly define bounds on each of the variables.

Consider the following example,

```
sos1 Variable s1(i) ;
Equation defsoss1 ;
defsoss1.. sum(i,s1(i)) =l= 3.5 ;
```

The equation defsoss1 implicitly defines the non-zero values that one of the elements of the SOS1 variable s1 can take.

A special case of SOS1 variables is when exactly one of the elements of the set have to be non-zero. In this case, the defsoss1 equation will be

```
defsoss1.. sum(i,s1(i)) =e= 3.5 ;
```

A common use of the use of this set is for the case where the non-zero value is 1. In such cases, the SOS1 variable behaves like a binary variable. It is only treated differently by the solver at the level of the branch and bound algorithm. For example, consider the following example to model the case where at most one out of n options can be selected. This is expressed as

```
sos1 variable x(i)
equation defx ;
defx.. sum(i,x(i)) =l= 1 ;
```

The variable \mathbf{x} can be made binary without any change in meaning and the solution provided by the solver will be indistinguishable from the SOS1 case.

The use of special ordered sets may not always improve the performance of the branch and bound algorithm. If there is no natural *order* the use of binary variables may be a better choice. A good example of this is the assignment problem.

I™ Not all MIP solvers allow SOS1 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. Any model that contains these variables may not be transferable among solvers. Please verify how the solver you are interested in handles SOS1 variables by checking the relevant section of the Solver Manual.

17.2.3 Special Order Sets of Type 2 (SOS2)

At most two variables within a SOS2 set can have non-zero values. The two non-zero values have to be adjacent. The most common use of SOS2 sets is to model piece-wise linear approximations to nonlinear functions.

The default bounds for SOS2 variables are 0 to $+\infty$. As with any other variable, the user may set these bounds to whatever is required.

Special ordered sets of type 2 are defined as follows,

sos2 Variable s2(i), t2(k,j), w2(i,j,k) ;

The members of the innermost (the right-most) index belongs to the same set. For example, in the sets defined above, s2 represents one special ordered set of type 2 with i elements, t2 defines k sets of j elements each, and w2 defines (i,j) sets with k elements each.

[PRODSCHX] shows SOS type formulations with binary, SOS1 and SOS2 sets. The default bounds for SOS variables are 0 to $+\infty$. As with any other variable, the user may set these bounds to whatever is required.

Not all MIP solvers allow SOS2 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. Any model that contains these variables may not be transferable among solvers. Please verify how the solver you are interested in handles SOS2 variables by checking the relevant section of the Solver Manual.

17.2.4 Semi-Continuous Variables

Semi-continuous variables are those whose values, if non-zero, must be above a given minimum level. This can be expressed algebraically as: Either x = 0 or $L \le x \le U$.

By default, this lower bound (L) is 1 and the upper bound (U) is $+\infty$. The lower and upper bounds are set through .lo and .up. In GAMS, a semi-continuous variable is declared using the reserved phrase semicont variable. The following example illustrates its use.

semicont variable x ;
x.lo = 1.5 ; x.up = 23.1 ;

The above slice of code declares the variable \mathbf{x} to be semi-continuous variable that can either be 0, or can behave as a continuous variable between 1.5 and 23.1.

- Not all MIP solvers allow semi-continuous variables. Please verify that the solver you are interested in can handle semi-continuous variables by checking the relevant section of the Solver Manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than 0. GAMS will flag an error if it finds that this is not the case.

17.2.5 Semi-Integer Variables

Semi-integer variables are those whose values, if non-zero, must be integral above a given minimum value. This can be expressed algebraically as: Either x = 0 or $x \in \{L, \ldots, U\}$

By default, this lower bound (L) is 1 and the upper bound (U) is 100. The lower and upper bounds are set through .lo and .up. In GAMS, a semi-integer variable is declared using the reserved phrase semiint variable. The following example illustrates its use.

semiint variable x ;
x.lo = 2 ; x.up = 25 ;

The above slice of code declares the variable x to be semi-continuous variable that can either be 0, or can take any integer value between 2 and 25.

- Not all MIP solvers allow semi-integer variables. Please verify that the solver you are interested in can handle semi-integer variables by checking the relevant section of the Solver Manual.
- The lower bound (L) has to be less than the upper bound (U), and both bounds have to be greater than 0. GAMS will flag an error during model generation if it finds that this is not the case.
- The bounds for semiint variables have to take integer values. GAMS will flag an error during model generation if it finds that this is not the case.

17.2.6 Setting Priorities for Branching

The user can specify an order for picking variables to branch on during a branch and bound search for MIP models through the use of priorities. Without priorities, the MIP algorithm will determine which variable is the most suitable to branch on. The GAMS statement to use priorities for branching during the branch and bound search is:

```
mymodel.prioropt = 1 ;
```

where mymodel is the name of the model specified in the model statement. The default value is 0 in which case priorities will not be used.

Using the .prior suffix sets the priorities of the individual variables. Note that there is one prior value for each individual component of a multidimensional variable. Priorities can be set to any real value. The default value is 1. As a general rule of thumb, the most important variables should be given the highest priority.

The following example illustrates its use,

```
z.prior(i,'small') = 3 ;
z.prior(i,'medium') = 2 ;
z.prior(i,'large') = 1 ;
```

In the above example, z(i, 'large') variables are branched on before z(i, 'small') variables.

The lower the value given to the .prior suffix, the higher the priority for branching.

All members of any SOS1 or SOS2 set should be given the same priority value since it is the set itself which is branched upon rather than the individual members of the set.

17.3 Model Scaling - The Scale Option

The rules for good scaling are exclusively based on algorithmic needs. GAMS has been developed to increase the efficiency of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as *natural* as possible. The units of measurement are one part of this natural notation, and there is unfortunately a potential conflict between what the modeler thinks is a good unit and what constitutes a well-scaled model.

17.3.1 The Scale Option

To facilitate the translation between a natural model and a well scaled model, GAMS has introduced the concept of a scale factor, both for variables and equations. The notations and definitions are quite simple. Scaling is turned off by default. Setting the model suffix .scaleopt to 1 turns on the scaling feature. For example,

```
model mymodel /all/ ;
mymodel.scaleopt = 1 ;
solve mymodel using nlp maximizing dollars ;
```

The statement should be inserted somewhere after the model statement and before the solve statement. In order to turn scaling off again, set the model.scaleopt parameter to 0 before the next solve.

The scale factor of a variable or an equation is referenced with the suffix .scale, i.e. the scale factor of variable x(i) is referenced as x.scale(i). Note that there is one scale value for each individual component of a multidimensional variable or equation. Scale factors can be defined using assignment statements. The default scale factor is always 1.

GAMS scaling is in most respects hidden from the user. The solution values reported back from a solution algorithm are always reported in the user's notation. The algorithm's versions of the equations and variables are

only reflected in the derivatives in the equation and column listings in the GAMS output if the options limrow and limcol are positive, and the debugging output from the solution algorithm generated with sysout option set to on.

17.3.2 Variable Scaling

The scale factor on a variable, V_s , is used to relate the variable as seen by the user, V_u , to the variable as seen by the algorithm, V_a , as follows: $V_a = V_u/V_s$

For example, consider the following equation,

```
positive variables x1,x2 ;
equation eq ;
eq.. 200*x1 + 0.5*x2 =l= 5 ;
x1.up = 0.01; x2.up = 10 ;
x1.scale = 0.01; x2.scale = 10 ;
```

By setting x1.scale to 0.01 and x2.scale to 10, the model seen by the solver is,

```
positive variables xprime1,xprime2 ;
equation eq ;
eq.. 2*xprime1 + 5*xprime2 =l= 5 ;
xprime1.up = 1; xprime2.up = 1 ;
```

Note that the solver does not see the variables x1 or x2, but rather the scaled (and better-behaved) variables xprime1 and xprime2.

^{ISF} Upper and lower bounds on variables are automatically scaled in the same way as the variable itself.

Solutions Integer and binary variables cannot be scaled.

17.3.3 Equation Scaling

Similarly, the scale factor on an equation, G_s , is used to relate the equation as seen by the user, G_u , to the equation as seen by the algorithm, G_a , as follows: $Ga = G_u/G_s$

For example, consider the following equations,

```
positive variables y1,y2 ;
equation eq1, eq2 ;
eq1.. 200*y1 + 100*y2 =l= 500 ;
eq2.. 3*y1 - 4*y2 =g= 6 ;
```

By setting eq1.scale to 100, the model seen by the solver is,

```
positive variables y1,y2 ;
equation eqprime1, eq2 ;
eqprime1.. 2*y1 + 1*y2 =l= 5 ;
eq2.. 3*y1 - 4*y2 =g= 6 ;
```

The user may have to perform a combination of equation and variable scaling until a well-scaled model is obtained.

Consider the following example,

positive variables x1,x2 ;
equation eq1, eq2 ;
eq1.. 100*x1 + 5*x2 =g= 20 ;
eq2.. 50*x1 - 10*x2 =l= 5 ;
x1.up = 0.2 ; x2.up = 1.5 ;

Setting the following scale values:

x1.scale = 0.1 ; eq1.scale = 5 ; eq2.scale = 5 ;

will result in the solver seeing the following well scaled model,

```
positive variables xprime1,x2 ;
equation eqprime1, eqprime2 ;
eqprime1.. 2*xprime1 + x2 =g= 4 ;
eqprime2.. xprime1 - 2*xprime2 =l= 1 ;
xprime1.up = 2 ; x2.up = 1.5 ;
```

17.3.4 Scaling of Derivatives

For nonlinear models, the derivatives also need to be well scaled. The derivatives in the scaled model seen by the algorithm, i.e. $d(G_a)/d(V_a)$ are related to the derivatives in the user's model, $d(G_u)/d(V_u)$ through the formula: $d(G_a)/d(V_a) = d(G_u)/d(V_u) \cdot V_s/G_s$.

The user can affect the scaling of derivatives by scaling both the equation and variable involved.

Appendix

Α

Glossary

acronym A GAMS data type used to give logical classifications to data points.

alias An alternative name for a set.

- **algorithm** This term may be used in two ways. It is either a prescription for how to solve a problem, or a particular solver system.
- assignment The statement used to change values associated with an identifier.
- **basic** A classification of a row or column that is in the basis maintained by solution methods that use linear programming iterations.
- binding An inequality constraint is binding when the value of the associated slack is zero.
- **bounds** Upper and lower limits on the possible values that a column may assume in a feasible solution. May be *infinite*, meaning that no limit is imposed.
- **column** An individual decision variable in the model seen by a solver program. Many may be associated with one GAMS variable
- **compilation** The initial phase of GAMS processing, when the program is being checked for syntax and consistency.
- **constant set** A set is constant if it remains unchanged. It has to be initialized with a set definition statement and cannot be changed using assignment statement. Sets used in domain definitions must be constant. Sets used in lag operations must be ordered as well. Sometimes the word static is used instead of constant.
- **constraint** A relationship between columns that must hold in a feasible solution. There may be many constraints associated with one GAMS equation.
- **continuous** There are two contexts. First a classification of a function. A plot of the function values will be a line without breaks in it. Second, a classification of variables. A continuous variable may assume any value within its bounds.

controlling sets See driving sets.

- data types Each symbol or identifier has to be declared to be one of the seven data types, which are set, parameter, variable, equation, model, file and acronym. The keywords scalar and table do not introduce separate data types but rather comprise a shorthand way to declare a symbol to be a parameter that will use a particular format for specifying initial values.
- **declaration** The entry of a symbol and the specification of its data type. A declaration may include the specification of initial values, and then it is more properly called a definition.
- default The value used, or the action taken, if the user provides no information.

- **definition** The definitions of the algebraic relationships in an equation are the assignment of initial values to parameters or of elements to sets as part of the initial declaration of the identifier.
- definition statements Units that describe symbols, assign initial values to them, and describe symbolic relationships. Some examples of the set, parameter, table, and model statements, and the equation definition statement.
- direction Either maximization or minimization, depending on whether the user is interested in the largest or the smallest possible value for the objective function.
- discontinuous A classification of a function. A plot of the function values will be a line with breaks in it.
- discrete A discrete variable (type binary or integer) may not assume any value between the bounds, but must assume integer values.
- **dollar control option** Directives or options used to control input or output detail associated with the GAMS compiler.
- **dollar operator** An operator used for exceptions handling in assignment statements and in equation definitions.
- **domain checking** The check that ensures that only legal label combination are used on every assignment to, or reference of, an identifier.
- domain definition The label combinations whose data will be updated in an assignment statement, or that will generate an individual constraint in an equation definition.
- **domain restriction condition** The alteration to the domain of definition caused when a dollar operator is used on the left (of the '= in an assignment or of the '...' in an equation definition).
- driving set The set that determine the domain of definition, or that control and index operation such as sum.
- **dynamic set** A set is dynamic if it has been changed with an assignment statement. Dynamic sets cannot be used with lag operations or in domain definitions.
- endogenous Data values that change when a solve statement is processed. In GAMS most often associated with variables.
- equation The GAMS data type used to specify required relationships between activity levels of variables.
- **execution** The second phase of GAMS processing, when GAMS is actually carrying out data transformations or generating a model.
- **execution statements** Instructions to carry out actions such as data transformations, model solutions, and report generation. Some examples are the assignment and the **option**, **display**, **loop** and **solve** statements.
- exogenous Data values known before a solve statement is processed, and not changed by the solve. In GAMS most often parameters.

explanatory text See text.

exponent A scale factor used to conveniently represent very large or small numbers.

- extended arithmetic The usual computer arithmetic is extended to include plus and minus infinity (+inf and -inf) and a special value for an arbitrarily a small number (i.e. one which is close to zero) known as epsilon (eps). Also, not available (na) can be used to indicate missing data, and undefined (undf) is the result of illegal operation. GAMS allows extended arithmetic for all operations and functions. The library problem [CRAZY] demonstrates extended arithmetic by showing the results for all operations and functions.
- e-format The representation of numbers when an exponent is used explicitly. For example, 1.1E+07.

feasible Often used to describe a model that has at least one feasible solution (see below).
- **GAMS coordinator** The person who looks after the administration of a GAMS system, and who will know what solvers are available and can tell you who to approach for help with GAMS problems. Unlikely to apply to personal computer versions.
- identifiers Names given to data entities. Also called symbols.
- index position(s) Another way of describing the set(s) that must be used when referencing a symbol of dimensionality one or more (i.e., a vector or a matrix).
- inequality constraint A constraint in which the imposed relationship between the columns is not fixed, but must be either greater than or equal to, or less than or equal to, a constant. The GAMS symbols =g= and =1= are used in equation definitions to specify these relationships.
- **infeasible** Used to describe either a model that has no feasible solution, or an intermediate solution that is not feasible (although feasible solutions may exist). See **feasible**, above.
- initialization Associating initial values with sets or parameters using lists as part of the declaration or definition, or (for parameters only) using table statements.
- **list** One of the ways of specifying initial values. Used with sets or parameters, most often for one-dimensional but also for two and higher dimensional data structures.
- list format One of the ways in which sets and parameters, can be initialized and all symbol classes having data can be displayed. Each unique label combination is specified in full, with the associated non-default value alongside.
- **marginal** Often called reduced costs or dual values. The values, which are meaningful only for non-basic rows or columns in optimal solutions, contain information about the rate at which the objective value will change of if the associated bound or right hand side is changed.

matrix element See nonzero element

- **model generation** The initial phase of processing a solve statement: preparing a problem description for the solver.
- model list A list of equations used in a model, as specified in a model statement.
- **nonbasic** A column that is not basic and (in nonlinear problems) not superbasic. Its value will be the same as the one of the finite bounds (or zero if there are no finite bounds) if the solution is feasible.
- **nonlinear nonzero** In a linear programming problem, the nonzero elements are constant. In a nonlinear problem, one or more of them vary because their values depend on that of one or more columns. The ratio of nonlinear (varying) to linear (constant) non linear zero elements is a good indicator of the pervasiveness of non-linearities in the problem.
- **nonoptimal** There are two contexts. First, describing a *variable*: a non-basic variable that would improve the objective value if made basic. The sign of the marginal value is normally used to test for non-optimality. Second, for a solution: other *solutions* exists with better objective values.
- **nonsmooth** A classification of function that does not have continuous first derivatives, but has continuous function values. A plot of the function values will be a line with *kinks* in it.
- **nonzero element** The coefficient of a particular column in a particular row if it is not zero. Most mathematical programming problems are sparse meaning that only a small proportion of the entries in the full tableau of dimensions *number of rows* by *number of columns* is different from zero.
- **objective row (or function)** Solver system require the specification of a row on (for nonlinear systems) a function whose value will be maximized or minimized. GAMS users, in contrast, must specify a scalar variable.

objective value The current value of the objective row or of the objective variable.

- objective variable The variable specified in the solve statement.
- optimal A feasible solution in which the objective value is the best possible.
- **option** The statement that allows users to change the default actions or values in many different parts of the system.
- ordered set A set is ordered if its content has been initialized with a set definition statement and the entry order of the individual elements of the set has the same partial order as the chronological order of the labels. A set name alone on the left-hand side of an assignment statement destroys the ordered property. Lag and Ord operations rely on the relative position of the individual elements and therefore require ordered sets. Ordered sets are by definition constant.
- output A general name for the information produced by a computer program.
- output file A disk file containing output. A GAMS task produces one such file that can be inspected.
- **parameter** A constant or group of constants that may be a scalar, a vector, or a matrix of two or more dimensions. Of the six data types in GAMS.
- **problem type** A model class that is dependent on functional form and specification. Examples are linear, nonlinear, and mixed integer programs.
- **program** A GAMS input file that provides a representation of the model or models.
- relational operator This term may be used in two ways. First, in an equation definition it describes the type of relationships the equation specifies, for example equality, as specified with the =e= symbol. Second, in a logical expression, the symbols eq, ne, lt and so on are also called relational operators, and are used to specify a required relationship between two values.
- right hand side The value of constant term in a constraint.
- scalar One of the forms of parameter inputs. Used for single elements.
- set A collection of elements (labels). The set statement is used to declare and define a set.
- simplex method The standard algorithm used to solve linear programming problems.
- slack The amount by which an inequality constraint is not binding.
- **slack variable** An artificial column introduced by a solver into a linear programming problem. Makes the implementation of simplex method much easier.
- smooth A classification of a function that has continuous first derivatives.
- **solver** A computer code used to solve a given problem type. An example is GAMS/MINOS, which is used to solve either linear or nonlinear programming problems.
- statements Sometimes called units. The fundamental building block of GAMS programs. Statements or sentences that define data structures, initial values, data modifications, and symbolic relationships. Examples are table, parameter, variable, model, assignment and display statements.

static set See constant set

superbasic In nonlinear programming, a variable that it is not in the basis but whose value is between the bounds. Nonlinear algorithms often search in the space defined by the superbasic variables.

symbol An identifier.

table One of the ways of initializing parameters. Used for two and higher dimensional data structures.

text A description associated with an identifier or label.

type See data type, problem type or variable type.

unique element A label used to define set membership.

- variable type The classification of variables. The default bounds are implicit in the type, and also whether continuous or discrete. The types are free, positive, binary, integer, semicont, semiint and negative.
- vector A one-dimensional array, corresponding to a symbol having one index position.
- **zero default** Parameter values are initially set to zero. Other values can be initialized using **parameter** or **table** statements. Assignment statements have to be used thereafter to change parameter values.

Β

The GAMS Model Library

Professor Paul Samuelson is fond of saying that he hopes each generation economists will be able to "stand on the shoulders" of the previous generation. The library of models included with the GAMS system is a reflection of this desire. We believe that the quality of modeling will be greatly improved and the productivity of modelers enhanced if each generation can stand on the shoulders of the previous generation by beginning with the previous models and enhancing and improving them. Thus the GAMS systems includes a large library, collectively called GAMSLIB.

The models included have been selected not only because they collectively provide strong shoulders for new users to stand on, but also because they represent interesting and sometimes classic problems. For example the tradeoff between consumption and investment is richly illustrated in the Ramsey problem, which can be solved using nonlinear programming methods. Examples of other problems included in the library are production and shipment by firms, investment planning in time and space, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and relational databases.

Another criterion for including models in the library is that they illustrate the modeling capabilities GAMS offers. For example, the mathematical specification of cropping patterns can be represented handily in GAMS. Another example of the system's capability is the style for specifying initial solutions as staring points in the search for the optimal solution of dynamic nonlinear optimization problems.

Finally, some models have been selected for inclusion because they have been used in other modeling systems. Examples are network problems and production planning models. These models permit the user to compare how problems are set up and solved in different modeling systems.

Most of the models have been contributed by GAMS users. The submission of new models is encouraged. If you would like to see your model in a future release of the library, please send the model and associated documents and reports to GAMS Development Corporation.

The most convenient way (Windows only) to access the model library is from within the GAMS IDE by going through: File \rightarrow Model Library \rightarrow Open GAMS Model Library. A window will pop up and give you access to all models.

Another way to access the library is through the gamslib command. This command copies a model from the library directory into the current directory. If you enter gamslib without any parameters, the command syntax will be displayed as shown below:

```
>gamslib modelname [target]
```

or

```
>gamslib modelnum [target]
```

where modelname is the modelname, modelnum is the model sequence number, and target is the target file name. If the target file name is not provided, the default is modelname.gms. For example, the [TRNSPORT] model could be copied in any of the following ways

>gamslib	trnsport	target	file:	trnsport.gms
>gamslib	1			trnsport.gms
>gamslib	trnsport myname			myname
>gamslib	1 myname			myname

The full and annotated list of the models of the GAMS Model Library is available at: $\rm http://www.gams.com/modlib/modlib.htm$

 \mathbf{C}

The GAMS Call

The entire GAMS system appears to the user as a single call that reads the input files and produces output files. Several options are available at this level to customize the GAMS run and define the output desired. Although details will vary with the type of computer and operating system used, the general operating principles are the same on all machines.

C.1 The Generic "no frills" GAMS Call

The simplest way to start GAMS is to enter the command

>gams myfile

from the system prompt and GAMS will compile and execute the GAMS statements in the file myfile. If a file with this name cannot be found, GAMS will look for a file with the extended name myfile.gms. The output will be written by default on the file myfile.lst. For example, the following statement compiles and executes the example problem [TRNSPORT] from the GAMS model library,

>gams trnsport

The output goes by default to the file trnsport.lst.

C.1.1 Specifying Options through the Command Line

GAMS allows for certain options to be passed through the command line. The syntax of the simple GAMS call described in Section C.2 is extended to look as follows,

>gams myfile key1=value1, key2=value2, ...

where key1 is the name of the option that is being set on the command line, and value1 is the value to which the option is set. Depending on the option, value1 could be a character string, an integer number or a real number. For example, consider the following commands to run [TRNSPORT] from the GAMS model library,

```
gams trnsport o myrun.lst lo 2
gams trnsport -o myrun.lst -lo 2
gams trnsport o=myrun.lst lo=2
gams trnsport -o=myrun.lst -lo=2
```

All the four commands above are equivalent, and each directs the output listing to the file myrun.lst. o is the name of the option, and it is set to myrun.lst. In addition, in each case, the log of the run is redirected to the file trnsport.log.

C.2 List of Command Line Parameters

The options available through the command line are grouped into the following functional categories affecting

the specific GAMS run	input file processing	other files
system settings	output in listing file	

Table C.1 briefly describes the various options in each of the categories. Section C.3 contains a reference list of all options available through the command line with detailed description for each.

Parameters c	ontrolling the specific GAMS run:		
action	processing options	bratio	basis acceptance threshold
cerr	sets compile time error limit	domlim	domain error limit
dumpopt	workfile dump option	eolonly	singe keyword-value pair option
errmsg	error message option	error	parameter error message
etlim	elapsed time limit in seconds	execmode	limits on call, execute, put and echo commands
forcework	force workfile translation	forlim	GAMS looping limit
fsave	force workfile to be written	heaplimit	maximum Heap size allowed in Mb
holdfixed	treat fixed variables as constants	integer1	integer communication cell
iterlim	iteration limit	keep	keep scratch files
nodlim	node limit in branch and bound tree	optca	absolute optimality criterion
optcr	relative optimality criterion	procdir	run or process directory
reslim	solve time limit	seed	random number seed
threads	number of threads to be used by a solver	timer	instruction timer threshold in milli seconds
warnings	number of warnings permitted	WORKIACTOR	work space multiplier
workspace	work space in MD	zerores	set results of certain operations to zero
zeroresrep	report use of zerores as a warning		
Parameters c	ontrolling system settings:		
charset	extended character set	cns	default CNS solver
curdir	sets current directory	dnlp	default DNLP solver
emp	default EMP solver	errnam	error message file name
errorlog	error message lines written to the log file	execerr	execution time error limit
griddir	grid file directory	gridscript	grid submission script
ide	Integrated Development Environment	inputdir	sets input search path
inputdir1	sets individual input search path	jobtrace	string written at the end of a trace
libincdir	sets library include directory	license	sets license file name
Tb	default LP solver	maxprocdir	maximum number of process directories
mcp	default MCP solver	minip	default MINLP solver
mip	default MDEC solver	miqcp	default MIQOP solver
mpec	accommand line perspector include file	nip	neiver license file neme
parmille	sots put directory	pricense	default OCP solver
roforonco	sumbal reference file	qcp rminln	default BMINI P solver
rmin	default BMIP solver	rmiach	default BMIOCP solver
rmpec	default RMPEC solver	scrdir	sets scratch directory
scrext	scratch extension for temporary files	scriptevit	script executed at the end of a GAMS run
scriptfrst	first line written to gamsnext	scriptnext	script mailbox file name
scrnam	scratch name	solvelink	solver link ontions
solvercntr	solver control file name	solverdict	solver dictionary file name
solverinst	solver instruction file name	solvermatr	solver matrix file name
solversolu	solver solution file name	solverstat	solver status file name
subsys	sets configuration file name	symbol	symbol file name
sys11	dynamic resorting	sysdir	sets system directory
sysincdir	sets system library directory	trace	trace file name
traceopt	trace file format option	user1	strings passed to subsystems
workdir	sets working directory		
Parameters a	ffecting input file processing:		
g205	sets version compatibility	input	sets input file name
multipass	controls multiple pass facility	on115	generate certain errors
optdir	option file directory	optfile	option file indicator
stringchk	controls string substitution check	tabin	sets tab spacing
Parameters a	ffecting output in listing file:		
appendout	output listing file append option	case	sets output case
dformat	sets date format	limcol	default column listing
limrow	default row listing	output	sets output file name
		- askas	sop at into interio

pagecontr	page control	pagesize	sets page size
pagewidth	sets page width	profile	global execution profiling option
profiletol	profile time tolerance	solprint	solution report print option
stepsum	controls step summary in listing file	suppress	compilation listing option
sysout	solver Status file reporting option	tformat	sets time format
Parameters a	ffecting other files		
appendexpand	append to expand file	appendlog	log file append option
dumpparms	controls parameter logging	expand	expand file name
ferr	sets compilation error message file name	filecase	casing of new file names (put, gdx, ref etc.)
gdx	GAMS data exchange file name	gdxcompress	compression of generated gdx file
gdxconvert	conversion to older version of gdx file	logfile	sets log file name
logline	controls amount of line tracing to log file	logoption	log file option
profilefile	write profile information to this file	restart	sets restart file name
- save	sets save file name	savepoint	save solver point in GDX file
xsave	compressed save file name	-	-

Table C.1: GAMS command line parameters

C.3 Detailed Description of Command Line Parameters

This section describes each of the command line parameters in detail. These parameters are in alphabetical order for easy reference. In each of the following options, an abbreviation and the default value, if available, are bracketed.

action (a=ce) Processing option

GAMS currently processes the input file in multiple passes. The three passes in order are:

Compilation During this pass, the file is compiled, and syntax errors are checked for. Data initialization statements like scalar, parameter, and table statements are also processed during this stage.

Execution During this stage, all assignment statements are executed.

- Model Generation During this stage, the variables and equations involved in the model being solved are generated.
- Values:
- c compile only
- e execute only
- ce compile and execute
- r restart
- g glue code generation

The **a=e** setting can only be used during restart on files that have previously been compiled, since models first need to be compiled before they can be executed.

appendexpand (ae=1) append to expand file

This option controls the manner of file opening of the expand file.

- Values: 0
- 0 reset expand file
 - 1 append to expand file

appendlog (al=0) log file append option

This option is used in conjunction with the 10=2 setting where the log from the GAMS run is redirected to a file. Setting this option to 1 will ensure that the log file is appended to and not rewritten.

Values: 0 reset log file 1 append to log file

appendout (ao=0) output listing file append option

Setting this option to 1 will ensure that the listing file is appended to and not rewritten.

Values: 0 reset listing file

1 append to listing file

bratio (bratio=0.25) basis acceptance threshold

The value specified for this option causes a basis to be discarded if the number of basic variables is smaller than bratio times the number of equations. The range is 0 to 1 while setting bratio to 1 will always cause the basis to be discarded, which is sometimes needed with nonlinear problems, and setting bratio to 0 will accept any basis.

case	(case=0)	output case option
T 7-1	0	

values:	0	write listing file in mixed case
	1	write listing file in upper case only

cerr (cerr=0) compile time error limit

The compilation will be aborted after n errors have occurred. By default, there is no error limit and GAMS compiles the entire input file and collects all the compilation errors that occur. If the file is too long and the compilation process is time consuming, **cerr** could be used to set to a low value while debugging the input file.

Values:0 no error limitn stop after n errors

charset (charset=1) extended character set

 Values:
 0 use limited GAMS characters set

 1 accept any character in comments and text items (foreign language characters)

cns (cns=text) default CNS solver

curdir (curdir=*text*) set current directory

This option sets the current directory. It is useful when GAMS is called from an external system like Visual Basic. If not specified, it will be set to the directory the GAMS module is called from.

dformat (df=0) date format

This option controls the date format in the listing file. The three date formats correspond to the various conventions used around the world. For example, the date December 2, 1996 will be written as 12/02/96 with the default df value of 0, as 02.12.96 with df=1, and as 96-12-02 with df=2.

Values:	0	$\rm mm/dd/yy$
	1	dd.mm.yy
	2	yy-mm-dd

dnlp (dnlp=text) default DNLP solver

domlim (domlim=0) domain error limit

This option sets the maximum number of domain errors. It allows errors to occur up to the given number during solution.

dumpopt (dumpopt=0) workfile dump option

This option creates a GAMS file of input that will reproduce results encapsulating all include files into one GAMS file. If activated a file will be written containing GAMS source code for the entire problem. The file name is the input file name plus the extension dmp.

- Values:
- 0 no dumpfile1 extract referenced data from the restart file using original set element names
- 2 extract referenced data from the restart file using new set element names
- 3 extract referenced data from the restart file using new set element names and drop symbol text
- 4 extract referenced symbol declarations from the restart file
- 11 write processed input file without comments
- 21 write processed input file with all comments

To illustrate the use of the dumpopt option, [TRNSPORT] has been split into two files. The first file (say trans1.gms) contains most of the original file except for the solve statement, and looks as follows,

```
sets
                            / seattle, san-diego /
       i
           canning plants
       j
           markets
                            / new-york, chicago, topeka / ;
parameters
       a(i)
             capacity of plant i in cases
              seattle
                          350
         1
              san-diego
                          600
                               1
            demand at market j in cases
       b(j)
              new-york
                          325
                          300
              chicago
              topeka
                          275 / ;
table d(i,j) distance in thousands of miles
                    new-york
                                   chicago
                                                topeka
      seattle
                       2.5
                                     1.7
                                                  1.8
      san-diego
                       2.5
                                     1.8
                                                  1.4
scalar f freight in dollars per case per thousand miles /90/;
parameter c(i,j) transport cost in thousands of dollars per case ;
 c(i,j) = f * d(i,j) / 1000 ;
variables
   x(i,j)
           shipment quantities in cases
            total transportation costs in thousands of dollars ;
    z
positive variable x ;
equations
    cost
                define objective function
    supply(i)
                observe supply limit at plant i
    demand(j)
                satisfy demand at market j ;
cost ..
               z =e= sum((i,j), c(i,j)*x(i,j));
supply(i) ..
              sum(j, x(i,j)) =l= a(i) ;
              sum(i, x(i,j)) =g= b(j);
demand(j) ..
model transport /all/ ;
```

All comments have been removed from [TRNSPORT] for brevity. Running this model and saving the work files through the save parameter leads to the generation of eight work files. The second file (say trans2.gms) generated from [TRNSPORT] looks as follows,

```
solve transport using lp minimizing z ; display x.l, x.m ;
```

One can then run trans2.gms restarting from the saved work files generated from running trans1.gms. The result obtained is equivalent to running [TRNSPORT].

In order to use the dumpopt parameter effectively, it is required that the first line in the restart file is the solve statement.

To illustrate the use of the dumpopt option, run the second model using the following command

gams trans2 r=trans dumpopt=1

where trans is the name of the saved files generated through the save parameter from trans1.gms. A new file trans2.dmp is created as a result of this call, and looks as follows,

```
* This file was written with DUMPOPT=1 at 11/30/11 08:43:06
    INPUT = C:\Fred\GAMS options\test\trnsport2.gms
*
    DUMP = C:\Fred\GAMS options\test\trnsport2.dmp
* RESTART = C:\Fred\GAMS options\test\trans1.g0?
            with time stamp of 11/30/11 08:40:41
* You may have to edit this file and the input file.
* There are 5 labels
Set WorkFileLabelOrder dummy set to establish the proper order /
   seattle,san-diego,new-york,chicago,topeka /;
Model transport;
Variable z total transportation costs in thousands of dollars;
Set i(*) canning plants /
   seattle,san-diego /
Set j(*) markets /
   new-york, chicago, topeka /
Parameter c(i,j) transport cost in thousands of dollars per case /
   seattle.new-york 0.225,seattle.chicago 0.153,seattle.topeka 0.162,
   san-diego.new-york 0.225, san-diego.chicago 0.162, san-diego.topeka 0.126 /
Positive Variable x(i,j) shipment quantities in cases;
Parameter a(i) capacity of plant i in cases /
   seattle 350,san-diego 600 /
Parameter b(j) demand at market j in cases /
   new-york 325, chicago 300, topeka 275 /
Equation demand(j) satisfy demand at market j;
Equation supply(i) observe supply limit at plant i;
Equation cost define objective function;
       *** EDITS FOR INPUT FILE ***
*** END OF DUMP ***
```

Note that all the data entering the model in the solve statement has been regenerated. The parameter d has not been regenerated since it does not appear in the model, but the parameter c is. Changing the value of the parameter dumpopt will result in alternate names being used for the identifiers in the regenerated file.

dumpparms (dp=0) GAMS parameter logging

The dumpparms parameter provides more detailed information about the parameters changed or set by the user, GAMS or the IDE during the current run.

- Values: 0 no logging
 - 1 lists accepted parameters
 - 2 log of file operations plus parameters

Note that with dp=2, all the file operations are listed including the full path of each file on which any operation is performed.

emp (emp=text) default EMP solver

eolonly (ey=0) single key-value pair option

By default, any number of keyword-value pairs can be present on the same line. This parameter is an immediate switch that forces only one keyword-value pair to be read on a line. If there are more than one such pairs on a line, then this option will force only the first pair to be read while all the other pairs are ignored.

Values:

0 any number of keys or values1 only one key-value pair on a line

errmsg (errmsg=0) error message option

This option controls the location in the listing file of the messages explaining the compilation errors.

Values:

- 0 error messages at the end of compiler listing
- 1 error messages immediately following error line
- 2 no error messages

To illustrate the option, consider the following slice of GAMS code:

set i /1*10/ ; set j(i) /10*11/;
parameter a(jj) / 12 25.0 / ;

The listing file that results from running this model contains the following section,

```
1 set i /1*10/ ; set j(i) /10*11/;
**** $170
2 parameter a(jj) / 12 25.0 / ;
**** $120
3
120 Unknown identifier entered as set
170 Domain violation for element
**** 2 ERROR(S) 0 WARNING(S)
```

Note that numbers (\$170 and \$120) flag the two errors as they occur, but the errors are explained only at the end of the source listing. However, if the code is run using the option errmsg=1, the resulting listing file contains the following,

1 set i /1*10/ ; set j(i) /10*11/; **** \$170 **** 170 Domain violation for element

```
2 parameter a(jj) / 12 25.0 /;
**** $120
**** 120 Unknown identifier entered as set
3
**** 2 ERROR(S) 0 WARNING(S)
```

Note that the explanation for each error is provided immediately following the error marker.

errnam (errnam=text) error message file name

Used to change the name GAMSERRS.TXT. The name text will be used as is.

error (error = text) Force a parameter error with message text

Forces a parameter error with given message string. This option is useful if one needs to incorporate GAMS within another batch file and need to have control over the conditions when GAMS is called. To illustrate the use of the error option, the default GAMS log file from running a model with the option error=hullo.

```
*** ERROR = hullo
*** Status: Terminated due to parameter errors
--- Erasing scratch files
Exit code = 6
```

errorlog (er=99) error message lines written to the log file

Values: 0 no error messages are written to log file *n* number of lines for each error written to log file

etlim (etl=+inf) elapsed time limit

This option controls the time limit for a GAMS job. The system will terminate with a compilation or execution error if the limit is reached.

execerr (execerr=0) execution time error limit

Entering or processing a solve statement with more than execerr errors will abort.

Values: 0 no errors allowed

n n errors allowed

execmode (execmode=0) limits on call, execute, put and echo commands

Values:

- 0 everything allowed
- 1 interactive shells in \$call and execute commands are prohibited
- 2 all \$call and execute commands are prohibited
- 3 \$echo or put commands can only write below the working or scratch directory
- 4 \$echo and put commands are not allowed

expand (ef = text) expand file name

The expand parameter generates a file that contains information about all the input files processed during a particular compilation. The names of the input files are composed by completing the name with the current directory. The following example illustrates the use of the expand parameter. Consider the following slice of code,

```
parameter a ; a = 0 ;
$include file2.inc
$include file2.inc
```

The content of the include file file2.inc is shown below,

a = a+1 ; display a ;

Running the model with the command line flag expand myfile.fil results in the creation of the file myfile.fil. The content of this file is provided below,

1	INPUT	0	0	0	1	7	E:\TEMP\FILE1.GMS
2	INCLUDE	1	1	2	2	4	E:\TEMP\FILE2.INC
3	INCLUDE	1	1	3	5	7	E:\TEMP\FILE2.INC

The first row always refers the parent file called by the GAMS call. The first column gives the sequence number of the input files encountered. The second column refers to the type of file being referenced. The various types of files are

0	INPUT	1	INCLUDE
2	BATINCLUDE	3	LIBINCLUDE
4	SYSTNCLUDE		

The third column provides the sequence number of the parent file for the file being referenced. The fifth column gives the local line number in the parent file where the **\$include** appeared. The sixth column gives the global (expanded) line number which contained the **\$include** statement. The seventh column provides the total number of lines in the file after it is processed. The eighth and last column provides the name of the file.

ferr (ferr = text) compilation error message file

Instructs GAMS to write error messages into a file. Completing the name with the scratch directory and the scratch extension composes the file name. The default is no compilation error messages. This option can be used when GAMS is being integrated into other environments like Visual Basic. The error messages that are reported in the listing file can be extracted through this option and their display can be controlled from the environment that is calling GAMS.

To illustrate the option, consider the following slice of GAMS code used to explain the errmsg option. Calling GAMS on this code with ferr=myfile.err, will result in a file called myfile.err being created in the scratch directory. This file contains the following lines:

0	0	0	0 D:\GAMS\NEW.LST
1	1	170	31 D:\GAMS\NEW.GMS
2	2	120	14 D:\GAMS\NEW.GMS

The first column refers to the global row number of the error in the listing file. The second column refers to the row number of the error in the individual file where the problem occurs. This will be different from the first column only if the error occurs in an include file. In this case, the second column will contain the line number in the include file where the error occurs, while the first number will contain the global line number (as reported in the listing file) where the error occurs. The number in the third column refers to the error number of the error. The fourth number refers to the column number of the error in the source file. The fifth column contains the individual file in which the error occurred.

filecase (filecase=0) casing of new file names (put, gdx, ref etc.)

This option allows one to alter the file name casing GAMS uses in saving put, gdx, ref etc. files. It only works with new file names but for example it won't create trnsport.REF if TRNSPORT.ref already exists.

Values: 0 causes GAMS to use default casing

- 1 causes GAMS to uppercase filenames
- ${\bf 2} \quad {\rm causes \ GAMS \ to \ lowercase \ filenames}$

forcework (fw=0) force workfile translation

Most of the work files generated by GAMS using the save option are in binary format. The information inside these files will change from version to version. Every attempt is made to be backward compatible and ensure that all new GAMS systems are able to read save files generated by older GAMS systems. However, at certain versions, we are forced to concede default incompatibility (regarding save files, not source files) in order to protect efficiency. The **forcework** option is used to force newer GAMS systems into translating and reading save files generated by older systems.

Values: 0 no translation 1 try translation

for lime (for $lim=2^{32}-1$) GAMS looping limit

This option specifies the maximum number of allowable executions of Control Structures involving a For, While or Repeat before GAMS signals an execution error and terminates the control structure.

fsave (fsave=0) force workfile to be written

This option allows to save a file even in the face of execution or other errors. How it works depends on the **save** option.

- Values: 0 workfile only written to file specified by save if no errors occur
 - 1 workfile always written to file specified by save or if save is not present to a file made up by GAMS

The option value 1 is mainly used by solvers that can be interrupted from the terminal.

g205 (g205=0) GAMS version 2.05 backward compatability

This option sets the level of the GAMS syntax. This is mainly used for backward compatibility. New key words have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow one to run such models.

- Values:
 - 0 latest syntax1 syntax from Release 2.05 only
 - 2 syntax from the first version of Release 2.25 only

For example, the word if is a key word in GAMS introduced with the first version of Release 2.25. Setting the g205=1 option allows if to be used as an identifier since it was not a keyword in Release 2.05. As another example, the word for is a key word in GAMS introduced with the later versions of Release 2.25. Setting the g205=2 option allows for to be used as an identifier since it was not a keyword in the first version of Release 2.25.

Using values of 1 or 2 for g205 will not permit the use of enhancements to the language introduced in the later versions.

gdx (gdx=text) GAMS data exchange file name

This option specifies the name of the GAMS data exchange file and causes such a file (a gdx file) to be written containing all data in the model at the end of the job. Setting gdx to the string 'default' causes GAMS to create a gdx file with the gms file root name and a gdx extension. Thus gams trnsport with gdx=default will cause GAMS to write the gdx file trnsport.gdx.

gdxcompress (gdxcompress=0) compression of generated gdx file

This option specifies whether the files are compressed or not.

Values:	
---------	--

Values:

0 do not compress gdx files1 compress gdx files

Note that version 5 gdx files do not support compression.

gdxconvert (gdxconvert=v7) conversion to older version of gdx file

This option specifies in which format the gdx files will be written.

- v5 version 5 gdx file, does not support compression
 - v6 version 6 gdx file
 - v7 version 7 gdx file

griddir (gdir=text) Grid file directory

This option sets the grid file directory.

gridscript (gscript=gmsgrid) grid submission script

This option provides the name of a script file to submit grid computing jobs. If only the file name is given the file is assumed to be located in the system directory. A fully qualified name can be given as well. Advanced knowledge of how GAMS sets up and calls solvers is needed for successful use.

heaplimit (hl=1e20) maximum heap size allowed in Mb

This option allows to limit the amount of memory a GAMS job can use during compilation and execution. If the needed data storage exceeds this limit, the job will be terminate.

holdfixed (holdfixed=0) treat fixed variables as constants

This option can reduce the problem size by treating fixed variables as constants.

- Values: 0 fixed variables are not treated as constants
 - 1 fixed variables are treated as constants

ide (ide=0) Integrated Development Environment flag

This option instructs GAMS to write special instructions to the log file that are in turn read by the IDE.

- Values:
- 1 causes the writing to occur (default when running under the IDE)
- 0 does not (default outside of IDE)

input (i=text) input file name

Completing the input file name with the current directory composes the final name. If such a file does not exist and the extension was not specified, the standard input extension is attached and a second attempt is made to open an input file.

inputdir (idir) input search path

In general, GAMS searches for input and include files in the current working directory only. This option allows the user to specify additional directories for GAMS to search for the input files. A maximum of 40 separate directories can be included with the directories separated by Operating System specific symbols. On a PC the separator is a semicolon (;) character, and under Unix it is the colon (:) character. Note that libinclude and sysinclude files are handled differently, and their paths are specified by libincdir and sysinclir.

Consider the following illustration:

gams myfile idir \mydir;\mydir2

The search order for the file myfile (or myfile.gms) and all included files in PC systems is as follows: (1) current directory, (2) directories specified by inputdir (\mydir and \mydir2 directories) in order. Under Unix, the corresponding command is

gams myfile idir \mydir:\mydir2

inputdir1 to inputdir40 (idir1=text,...,idir40=text) input search path

The same information as in **inputdir** can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to **inputdir** is important because the earlier **inputdir** directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

gams myfile idir1 mydir1 idir2 mydir2

Note that the search order in this case is as follows:

- 1. current directory
- 2. mydir1
- 3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows:

- 1. current directory
- 2. mydir2
- 3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

integer1 to integer5 (integer1=number) integer communication cell

This integer communication cell that can contain any integer number.

iterlim (iterlim=200000000) iteration limit

This option specifies the maximum number of allowable solver iterations, before the solver terminates the run.

jobtrace (jt=text) string written at the end of a trace

The string is written to the trace file at the end of a GAMS job.

keep (keep=0) keep scratch files

This parameter tells GAMS whether to keep the temporary files generated during a run.

- Values: 0 delete all files
 - 1 keep intermediate files

libincdir (ldir=text) library include directory

Used to complete a file name for **\$libinclude**. If the ldir option is not set, the sub-directory **inclib** in the GAMS system directory is searched.

- Unlike idir, additional directories cannot be set with ldir. The string passed will be treated as one directory. Passing additional directories will cause errors.
- Note that if the ldir parameter is set, the default library include directory is not searched.

Consider the following illustration,

gams myfile ldir mydir

GAMS searches for any referenced \$libinclude file in the directory mydir.

license (license=text) license file name

This option should only be used by advanced users attempting to override internal license information. The file name is used as given. The default license file is gamslice.txt in the GAMS system directory.

limcol (limcol=3) default column listing

This option controls the number of cases written to the LST file for each named variable in a model.

Values: n first n columns listed

limrow (limrow=3) default row listing

This option controls the number of cases written to the LST file for each named equation in a model.

Values: n first n rows listed

logfile (lf = text) log file name

This option is used in conjunction with the lo option. If lo is set to 2, then this option will specify the name of the log file name. The name provided by the option is completed using the current directory. If no logfile is given but the value of lo is 2, then the file name will be input file name with the extension .log

To illustrate the use of the logfile option, run [TRNSPORT] with the options lo=2 and lf=myfile.log. The resulting log file is redirected to myfile.log, and looks as follows:

--- Starting compilation --- trnsport.gms(69) 3 Mb --- Starting execution: elapsed 0:00:00.002 --- trnsport.gms(44) 4 Mb --- Generating LP model transport --- trnsport.gms(65) 4 Mb ___ 6 rows 7 columns 19 non-zeroes --- Executing CPLEX: elapsed 0:00:00.007 IBM ILOG CPLEX Jul 14, 2011 23.7.1 WEX 26779.26792 WEI x86_64/MS Windows Cplex 12.3.0.0 Reading data... Starting Cplex... Tried aggregator 1 time. LP Presolve eliminated 1 rows and 1 columns. Reduced LP has 5 rows, 6 columns, and 12 nonzeros. 0.00 sec. Presolve time =

188

Iteration	Dual Objective	In Variable	Out Va	riable		
1	73.125000	x(seattle.new-york)	demand(new-york)	slack		
2	119.025000	x(seattle.chicago)	demand(chicago)	slack		
3	153.675000	x(san-diego.topeka)	demand(topeka)	slack		
4	153.675000	x(san-diego.new-york)	<pre>supply(seattle)</pre>	slack		
LP status(1):	optimal					
Optimal soluti Objective :	ion found. 153.675000					
Restarting	g execution					
trnsport.g	gms(65) 2 Mb					
Reading so	olution for model	transport				
Executing	Executing after solve: elapsed 0:00:00.046					
trnsport.gms(67) 3 Mb						
*** Status: No	*** Status: Normal completion					
Job trnspo	ort.gms Stop 11/3	0/11 05:27:11 elapsed (0:00:00.046			

logline (ll=2) amount of line tracing to log file

This option is used to limit the number of line tracing sent out to the log file during the compilation phase of a GAMS run. Values of 0 and 1 are special. Setting 11=0 will cause the line tracing to be suppressed for all phases of the GAMS processing.

Values:

- 0 all line tracing suppressed
 - 1 limited line tracing
 - 2 full line tracing

The log file that results from running [TRNSPORT] with the option 11=0 is shown below,

```
--- Starting compilation
--- Starting execution: elapsed 0:00:00.003
--- Generating LP model transport
--- 6 rows 7 columns 19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.007
IBM ILOG CPLEX
                Jul 14, 2011 23.7.1 WEX 26779.26792 WEI x86_64/MS Windows
Cplex 12.3.0.0
Reading data...
Starting Cplex...
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time =
                  0.00 sec.
Iteration
               Dual Objective
                                         In Variable
                                                               Out Variable
    1
                   73.125000
                                x(seattle.new-york) demand(new-york) slack
    2
                   119.025000
                                 x(seattle.chicago) demand(chicago) slack
    3
                   153.675000
                                x(san-diego.topeka)
                                                      demand(topeka) slack
                   153.675000 x(san-diego.new-york)
    4
                                                      supply(seattle) slack
LP status(1): optimal
Optimal solution found.
Objective :
                    153.675000
--- Restarting execution
--- Reading solution for model transport
--- Executing after solve: elapsed 0:00:00.096
*** Status: Normal completion
--- Job trnsport.gms Stop 11/30/11 05:42:55 elapsed 0:00:00.097
```

Comparing this output to the one shown in the example of option logfile, one can see that the line numbers are absent from the log file.

logoption (lo=1) log file option

This option controls the location of the output log of a GAMS run. By default, GAMS directs the log of the run to the screen/console. If lo=2, the log is redirected to a file. With lo=3 all the output goes to the standard output. If no file name is provided for the log through the lf option, the file name will be the input file name with the extension .log.

Values:

- $0 \quad {\rm no} \ {\rm log} \ {\rm output}$
- 1 log output to screen/console
- 2 log output to file
- 3 log output to stdout

To illustrate the use of the lo option, run [TRNSPORT] with the options lo=2. The resulting log file, trnsport.log, looks exactly as shown in the example of option logfile.

lp (lp=text) default LP solver

maxprocdir (maxprocdir=26) Maximum number of process directories

This option controls the maximum number of workfile directories that can be generated by GAMS. By default they are called 225a, 225b, ..., 225aa, 225ab ...

mcp (mcp=text) default MCP solver

minlp (minlp=text) default MINLP solver

mip (mip=text) default MIP solver

miqcp (miqcp=text) default MIQCP solver

mpec (mpec=text) default MPEC solver

multipass (mp=0) multipass facility

This option allows slices of GAMS code to be independently checked for syntax errors. This option is useful when a large model is being put together from smaller pieces.

Values: 0 standard compilation 1 check out compilation

Consider the following example,

```
a(i) = b(i)*5 ;
b(i) = c(j) ;
```

By default, running a file containing just the two statements shown above results in the following listing file,

```
1 a(i) = b(i)*5;
**** $140$120$140
2 b(i) = c(j);
**** $140$120$149
120 Unknown identifier entered as set
140 Unknown symbol
149 Uncontrolled set entered as constant
**** 6 ERROR(S) 0 WARNING(S)
```

None of the sets i, or j have been defined or initialized, and the identifiers a, b and c have not been defined. Further, an assignment cannot be made without the right hand side of the assignment being known. In both assignments in the example above, there is no data available for the right hand side. Running the model with the setting mp=1 results in the following listing file,

```
1 a(i) = b(i)*5;
2 b(i) = c(j);
**** $149
Error Messages
149 Uncontrolled set entered as constant
**** 1 ERROR(S) 0 WARNING(S)
```

Note that the statements in the example have now been processed independently of its context. They are now checked only for consistency. GAMS now assumes that sets i and j, as well as the identifiers a, b, and c are defined and, if necessary, initialized elsewhere. The only error that is reported is the inconsistency of indices in the second statement.

nlp (nlp=text) default NLP solver

nodlim (nodlim=0) node limit in branch and bound tree

This option specifies the maximum number of nodes to process in the branch and bound tree for a MIP problem. The default value 0 is interpreted as 'not set'.

on115 (on115=0) generate certain errors

Generate errors for unknown unique elements in an equation.

Values: 0 no error messages

1 issue error messages

optca (optca=0.0) absolute optimality criterion

This option specifies an absolute termination tolerance for use in solving all mixed-integer models.

optcr (optcr=0.1) relative optimality criterion

This option specifies a relative termination tolerance for use in solving all mixed-integer models.

optdir (optdir=text) option file directory

This keyword gives the name of the directory to be used by GAMS for solver option files. If not specified, it will be set to the current working directory.

optfile (optfile=0) option file indicator

This option initializes the modelname.optfile parameter to the value set. Modelname is the name of the model specified in the model statement. For example, the file myfile contains the slice of GAMS code

```
model m /all/ ;
solve m using nlp maximizing dollars ;
```

Consider the following call,

gams myfile optfile=1

The option file that is being used after this assignment is **solvername.opt**, where **solvername** is the name of the solver that is specified. For CONOPT, the option file is called **conopt.opt**; for MINOS, it is **minos.opt**. The names that you can use are listed in the Solver Manual.

Setting modelname.optfile in the GAMS input file overrides the value of the optfile parameter passed through the command line.

To allow different option file names for the same solver, the optfile parameter can take other values as well. Formally, the rule is optfile=n will use solvename.opt if n = 1, and solvename.opX, solvename.oXX or solvename.XXX, where X's are the characters representing the value of n, for n > 1 and will use no option file at all for n = 0. For example, the following optfile values profile the option file names for the CONOPT solver

0	no option file used	1	conopt.opt
2	conopt.op2	26	conopt.o26
345	conopt.345	1234	conopt.1234

output (o=text) output file name

If no name is given, the input file name is combined with the current directory and the standard output file extension (LST) is applied. If the output parameter is given as a file name without an absolute path, using the current directory composes the final name. If the absolute path is included in the file name, then the name is used as given.

Consider the following examples,

```
gams trnsport
gams trnsport o=trnsport.out
gams trnsport o=c:\test\trnsport.out
```

The first call will create an output file called trnsport.lst (for PC and Unix platforms) in the current directory. The second call will create a file called trnsport.out in the current directory. The last call will create the file as listed. If the directory c:\test does not exist, GAMS will exit with a parameter error.

pagecontr (pc=3) page control

This option affects the page control in the listing file.

Values:

- 0 no page control with padding
- 1 Fortran style line printer format
- $2 \quad {\rm no \ page \ control, \ no \ padding}$
- **3** Form feed character for new page

pagesize (ps=58) page size

This is the number of lines that are used on a page for printing the listing file. The lower bound is 0 which is interpreted as +inf. That means that everything is printed to one page.

pagewidth (pw=255) print width

This option sets the print width on a page in the listing file. The range is 72 to 32767. If the value is outside this range, the default value of 255 will be used. Note that under the IDE pagewidth is set to 80.

parmfile (pf=text) command line parameter include file

This option specifies the name of a secondary customization parameter file to use. It is used to augment the command line adding more command line parameters from a file. It is read from the current directory unless a path is specified.

plicense (plicense=text) privacy license file name

This keyword tells the name of a privacy license file that contains file encryption codes. A full path should be used.

procdir (procdir=text) run or process directory

This keyword gives the name of the directory where the work files generated by GAMS will be stored. This directory must already exist, otherwise an error is triggered. If not specified, it will be set to 225 with characters a,b,c,... attached.

profile (profile=0) global execution profiling option

This option initializes the profile option (see Appendix \mathbf{E}) to the value set, and allows the profile of a GAMS run to be printed in the listing file. The profile contains the individual and cumulative time required for the various sections of the GAMS model.

- Setting the **profile** option through the option statement in the GAMS input file overrides the value of the profile parameter passed through the command line.
- Values:
- 0 no profiling1 minimum profiling
- n profiling depth for nested control structures

A value of 0 does not cause an execution profile to be generated. A value of 1 reports execution times for each statement and the number of set elements over which the particular statement is executed. A value of 2 reports specific times for statements inside control structures like loops etc. Running [TRNSPORT] with profile=1 provides the following additional information in the listing file,

 1 ExecInit		0.000	0.000	SECS	3	МЪ	
 44 Assignme	nt c	0.000	0.000	SECS	4	МЪ	6
 65 Solve In	it transport	0.000	0.000	SECS	4	МЪ	
 57 Equation	cost	0.000	0.000	SECS	4	МЪ	1
 59 Equation	supply	0.000	0.000	SECS	4	МЪ	2
 61 Equation	demand	0.000	0.000	SECS	4	МЪ	3
 65 Solve Fi	ni transport	0.000	0.000	SECS	4	МЪ	19
 65 GAMS Fin	i	0.015	0.015	SECS	4	МЪ	
 1 ExecInit		0.000	0.000	SECS	2	МЪ	
 65 Solve Re	ad transport	0.000	0.000	SECS	2	МЪ	
 67 Display		0.000	0.000	SECS	3	МЪ	
 67 GAMS Fin	i	0.000	0.000	SECS	3	Mb	

The first column provides the line number in the input file of the statement being executed. The second column provides the type of statement being executed.

ExecInit denotes the beginning of the execution phase of the GAMS input file.

GAMS Fini denotes the end of this phase.

Note that GAMS finishes processing of an input file as soon as a solve statement is processed, and passes control to the solver being called. After the solver is done, GAMS restarts. This causes two ExecInit GAMS Fini pairs to be generated for [TRNSPORT].

Assignment c denotes an assignment statement involving the identifier c.

Solve Init, Solver Fini are book ends enclosing the generation of the model [TRNSPORT].

Note that only equations are listed, and not variables. This happens because GAMS uses an equation based scheme to generate a model. The third and fourth columns provide the individual time needed to execute the statement, and the cumulative time taken by the GAMS system so far. The last column gives the number of assignments generated in the specified line.

At the end of the log file a profile summary is created which contains (up to) ten of the slowest execution steps. For example such a summary looks like this:

 Profile	Summary	(184 records	s processed)
0.062	3621	GAMS Fini	
0.047	3621	Solve Read	wsisn
0.046	3529	Equation	divcnlsea (86)
0.032	3621	Solve Fini	wsisn (39489)
0.016	3274	Assignment	wnr (2502)
0.016	3447	Equation	cost (15)
0.016	3475	Equation	laborc (180)
0.016	3519	Equation	waterbaln (180)
0.016	3546	Equation	subirrc (84)
0.015	3030	Assignment	gwtsa (273)

Note that this summary does not belong to [TRNSPORT] but is more capable for illustration because the summary of [TRNSPORT] is empty.

profile (pfile=text) write profile information to this file

This option causes profiling information to be written to a file. Note that profiling information is only created with profile=1 or profile=2. For example such a file looks like this:

1	-1	0.000	0.003 ExecInit
45	6	0.000	0.004 Assignment c
66	-1	0.000	0.004 Solve Init transport
58	1	0.000	0.004 Equation cost
60	2	0.000	0.004 Equation supply
62	3	0.000	0.004 Equation demand
66	19	0.015	0.004 Solve Fini transport
66	-1	0.000	0.004 GAMS Fini
1	-1	0.000	0.002 ExecInit
66	-1	0.000	0.002 Solve Read transport
68	-1	0.000	0.003 Display
68	-1	0.000	0.003 GAMS Fini

profiletol (ptol=0.0) profile time tolerance

This option specifies the minimum time a statement must use to appear in the profile generated output.

putdir (pdir=text) put directory

This option specifies the directory where the put files are generated and saved. If not specified, it will be set to the working directory. This option does not work if an absolute file name is provided through the file statement.

qcp (qcp=text) default QCP solver

reference (rf = text) symbol reference file

If specified, all symbol references will be written to this file. Setting **rf** or **Reference** to the string 'default' causes GAMS to create a reference file with the GMS file root name and a REF extension. Thus gams trnsport with **rf=default** will cause GAMS to write the reference file trnsport.ref.

reslim (reslim=1000) solve time limit

This option specifies the maximum time in seconds that the computer can run during execution of a solver, before the solver terminates the run.

restart (r=text) restart file name

This option provides the name of the save files to restart from. The final name is composed by completing the file name with the current directory and the standard workfile extension. The name provided for the restart file follows the same convention as that of the save file (see command line parameter save).

rminlp (rminlp=text) default RMINLP solver

rmip (rmip=text) default RMIP solver

rmiqcp (rmiqcp=text) default RMIQCP solver

rmpec (rmpec=text) default RMPEC solver

```
save (s=text) save file name
```

The final name is composed by completing the save file name with the current directory and the standard workfile extension. Eight save files are generated, so the name provided by the user for the save file should be such that GAMS can generate eight names from it. GAMS distinguishes file names from their extensions. If no extension is provided by the user, GAMS adds the extensions g01 through g08 to name the eight saved work files. The presence of a ? character in the save file name is used by GAMS to substitute the numbers 1 through 8 in its place.

The following table illustrates through examples, the generation of names for the save files by GAMS from the name provided through the **save** parameter.

```
myfile:
               myfile.g01,
                              myfile.g02,
                                                   myfile.g08
               myfile1.g01,
                             myfile2.g02,
mvfile?:
                                                   mvfile8.g08
                                             . . . .
                              myfile.002,
               myfile.001,
myfile.00?:
                                             ...,
                                                   myfile.008
              myfile1.wrk,
                             mvfile2.wrk.
                                                   myfile8.wrk
myfile?.wrk:
                                             . . . ,
myfile?.??:
               myfile1.111, myfile2.222,
                                                   myfile8.888
                                             . . . .
```

On Unix platforms the ? character is a special character and may require a backslash character (\) in front of it in order to be interpreted correctly. The name myfile? should be written on this platform as myfile\?.

savepoint (sp=0) save solver point in GDX file

This option tells GAMS to save a point format GDX file that contains the information on the current solution point.

Values:

- 0 no point GDX file is to be saved
 - 1 a point gdx file from the last solve is to be saved
 - 2 a point gdx file from every solve is to be saved

scrdir (sd=text) scratch directory

This option sets the scratch directory where the intermediate files generated by GAMS and the various solvers are stored. The files in the scratch directory are used by GAMS and the various solvers to communicate with each other. The scratch directory and all its contents are usually deleted at the end of a GAMS run. If not specified, the scratch directory will be set to the default one generated by GAMS.

scrext (se=dat) Scratch extension for temporary files

This keyword gives the name of the extension for the GAMS temporary files generated during execution.

scriptexit (scriptexit=text) program or script executed at the end of a GAMS run

By default GAMS does not call an exit script anymore. If this is required, the GAMS parameter ScriptExit has to be set explicitly to the script that should be called after GAMS terminates. An empty template of an exit script can be found in the GAMS system directory (gmsxitnt.cmd (Windows) or gmsxitus.run (Unix)).

scriptfrst (sf=text) first line written to gamsnext

The default is an empty string and the *first* line is not written.

scriptnext (script=gamsnext) script mailbox file name

scrnam (sn=text) scratch name

Name stem used to complete the names of intermediate work files. This name stem has to have at least one '?'. Name will be completed with the scratch directory and the standard scratch name extension.

seed (seed=3141) random number seed

This option specifies the seed used for the pseudo random number generator.

solprint (solprint=1) solution report print option

This option controls the printing of the model solution to the listing file.

- Values:
- 0 remove solution listings following solves
- 1 include solution listings following solves
- 2 suppress all solution information

solvelink (sl=0) solver link options

This option controls GAMS function when linking to solve.

Values:

- 0 GAMS operates as always
- 1 the solver is called from a shell and GAMS remains open
- 2 the solver is called with a spawn (if possible as determined by GAMS) or a shell (if the spawn is not possible) and GAMS remains open
- 3 GAMS starts the solution and continues in a Grid computing environment
- 4 GAMS starts the solution and waits (same submission process as 3) in a Grid computing environment
- 5 the problem is passed to the solver in core without use of temporary files

solvercntr (scntr=text) solver control file name default name override

Name completed with scratch directory and scratch extension.

solverdict (sdict=*text*) solver dictionary file name default name override Name completed with scratch directory and scratch extension.

solverinst (sinst=text) solver instruction file name default name override

Name completed with scratch directory and scratch extension.

solvermatr (smatr = text) solver matrix file name default name override Name completed with scratch directory and scratch extension.

solversolu (ssolu=text) solver solution file name default name override Name completed with scratch directory and scratch extension.

solverstat (sstat=text) solver status file name default name override

Name completed with scratch directory and scratch extension.

stepsum (stepsum=0) step summary option

This option controls the generation of a step summary of the processing times taken by GAMS during a given run.

Values: 0 no step summary

1 step summary printed

To illustrate the use of the **stepsum** option, the default GAMS lst file from running [TRNSPORT] with the option **stepsum=1** contains the following step summaries.

STEP	SUMMARY:	0.000	0.000	STARTUP
		0.000	0.000	COMPILATION
		0.000	0.000	EXECUTION
		0.000	0.000	CLOSEDOWN
		0.000	0.000	TOTAL SECONDS
		0.008	0.008	ELAPSED SECONDS
		3.949	3.949	MAX HEAP SIZE (Mb)
STEP	SUMMARY:	0.000	0.000	STARTUP
		0.000	0.000	COMPILATION
		0.000	0.000	EXECUTION
		0.000	0.000	CLOSEDOWN
		0.000	0.000	TOTAL SECONDS
		0.089	0.096	ELAPSED SECONDS
		2.884	3.949	MAX HEAP SIZE (Mb)

The first step summary occurs before the model is sent to the solver, and the second occurs after the solver completes its task and returns control back to GAMS. The first column reports time for the individual section of the run, while the second column reports accumulated times including previous sections.

stringchk (stringchk=0) string substitution check for %xxx% symbols

This option affects the result of the check for %xxx% symbols.

- Values:
- 0 no substitution and no error if symbol is undefined
 - 1 error if symbol undefined
 - 2 remove entire symbol reference if undefined and no error

subsys (subsys = text) configuration file name

This option is only to be used by advanced users attempting to override internal sub-system information. The file name is used as given. The default sub-systems file is GMSCMPNT.TXT on Windows or GMSCMPUN.TXT on Unix.

suppress (suppress=0) compiler listing option

This option suppresses the echoing of the contents of the input file(s) to the listing file. This parameter is similar in functionality to the **\$offlisting** dollar control option.

Values: 0 standard compiler listing

- 1 suppress compiler listing
- The **\$offlisting** and **\$onlisting** dollar control options effect the listing file only if **suppress** is set to 0. If **suppress** is set to 1, the input file(s) is not echoed to the listing file, and these dollar control options have no effect on the listing file.

symbol (symbol = text) symbol file name

Writes a partial symbol table to be used in conjunction with reference files.

sys11 (sys11=0) dynamic resorting

Speed-up for expressions containing constant indices or indices that are not in the natural order at the cost of increased memory use.

- Values: 0 automatic optimization/restructuring of data
 - 1 no optimization
 - 2 always optimize/restructure

sysdir (sysdir=text) system directory

This option sets the GAMS system directory. This option is useful if there are multiple systems installed on the machine, or when GAMS is called from an external system like Visual Basic.

system library search directory (sdir=text) system library search directory

Used to complete a file name for **\$sysinclude**. If the sdir option is not set, the GAMS system directory is searched.

- Unlike idir, additional directories cannot be set with sdir. The string passed will be treated as one directory. Passing additional directories will cause errors.
- Note that if the sdir parameter is set, the default system include directory is not searched.

Consider the following illustration,

gams myfile sdir mydir

GAMS searches for any referenced \$sysinclude file in the directory mydir.

sysout (sysout=0) expand solution output

This option controls the incorporation of additional solver generated output (that in the solver status file) to the listing file.

- Values: 0 suppress additional solver generated output
 - 1 include additional solver generated output

tabin (tabin=8) tab spacing

This option sets the tab spacing. By default, tabs are not allowed in GAMS. However, the most common setting is 8 which results in the positions of the tabs corresponding to columns $1, 9, 17, \ldots$ and the intermediate columns being replaced by blanks.

- Values:
- 0 tabs are not allowed
 - 1 tabs are replaced by blanks
 - *n* tabs are 1, n + 1, 2n + 1

tformat (tf=0) time format

This option controls the time format in the listing file. The three date formats correspond to the various conventions used around the world. For example, the time 7:45 PM will be written as 19:45:00 with the default tf value of 0, and as 19.45.00 with tf=1.

Values: 0 hh:mm:ss

1 hh.mm.ss

threads (threads=1) number of threads to be used by a solver

This option controls the number of threads or CPU cores to be used by a solver.

Values: -n

- -n number of cores to leave free for other tasks
 - $0\quad {\rm use \ all \ available \ cores}$
 - n use n cores (will be reduced to the available number of cores if n is too large)

timer (timer=0) instruction timer threshold in milli seconds

Only details about internal GAMS intructions that took more than n milli seconds are echoed to the log.

Values:

0 interpreted as +inf, no details echoedn echo all details about internal GAMS intructions that took more than n

milli seconds to the log

trace (trace=text) trace file name

The trace file name is completed using the current directory.

traceopt (traceopt=0) trace file format option

Values:

0 solver and GAMS step trace without headers

- 1 solver and GAMS step trace
- 2 solver step trace only
- 3 trace file format used for GAMS performance world
- 5 trace file with all available trace fields

user1 to user5 (u1=text,...,u5=text) strings passed to the subsystems

This options permit user entry of text for up to 5 user-defined options.

warnings (warnings=MAXINT) number of warnings permitted

This option specifies the maximum number of allowable warnings, before the run terminates.

workdir (wdir=curdir) sets working directory

This option sets the working directory. This option is useful when GAMS is called from an external system like Visual Basic. If not specified, it will be set to the curdir directory.

workfactor (workfactor=1.0) work space multiplier

This option tells the solver how much workspace to allocate for problem solution relative to the GAMS estimate.

workspace (workspace=number) workspace in Mb

This option tells the solver how much workspace in Megabytes to allocate for problem solution. If not given by the user the solver can choose the size and usually it is the memory available on the machine.

xsave (xs=text) extended save file name

In older GAMS systems (versions older than 21.7) the name of a save file written in ASCII format so it is platform independent and can be moved to machines with different operating systems, otherwise like **save**. In GAMS systems from release 22.3 and newer it causes writing of compressed save files.

zerores (zerores=0.0) set results of certain operations to zero

This option specifies the threshold value for internal rounding to zero in certain operations.

zeroresrep (zeroresrep=0) report underflow as a warning

This option causes GAMS to issue warnings whenever a rounding occurs because of zerores.

Values:

- 0 no warning when a rounding occurs because of zerores
 - 1 issue warnings whenever a rounding occurs because of zerores

 \mathbf{D}

Dollar Control Options

D.1 Introduction

The *Dollar Control Options* are used to indicated compiler directives and options. Dollar control options are not part of the GAMS language and must be entered on separate lines recognized by a \$ symbol in the first column. A dollar control option line may be placed anywhere within a GAMS program and it is processed during the compilation of the program. The \$ symbol is followed by one or more options identified by spaced. Since the dollar control options are not part of the GAMS language, they do not appear on the compiler listing unless an error had been detected. Dollar control option lines are not case sensitive and a continued compilation uses the previous settings.

D.1.1 Syntax

In general, the syntax in GAMS for dollar control options is as follows,

```
$option_name argument_list {option_name argument_list}
```

where option_name is the name of the dollar control option, while argument_list is the list of arguments for the option. Depending on the particular option, the number of arguments required can vary from 0 to many.

- Solution No blank space is permitted between the \$ character and the first option that follows.
- In most cases, multiple dollar control options can be processed on a line. However, some dollar control options require that they be the first option on a line.
- The effect of the dollar control option is felt immediately after the option is processed.

An example of a list of dollar control options is shown below,

\$title Example to illustrate dollar control options
\$onsymxref onsymlist

Note that there is no blank space between the \$ character and the option that follows. The first dollar control option **\$title** sets the title of the pages in the listing file to the text that follows the option name. In the second line of the example above, two options are set - **\$onsymtref** and **\$onsymlist**. Both these options turn of the echoing of the symbol reference table and listings to the listing file.

D.2 List of Dollar Control Options

The Dollar Control Options are grouped into nine major functional categories affecting

input comment format reference maps environment variables input data format program control macro definitions output format GDX operations Compression and encrypting

Table D.1 briefly describes the various options in each of the categories. Section D.3 contains a reference list of all dollar control options in alphabetical order with detailed description for each.

Non-default settings are reported before the file summary at the end of a GAMS listing as a reminder for future continued compilations. This is only relevant if a restart file has been requested with the GAMS call.

Options affe	cting input comment format:		
comment	set the comment character	offNestCom	turn off nested comments
eolCom	set end-of-line comment character	offText	off text mode
inlineCom	set in line comment character	onEolCom	turn on end-of-line comments
maxCol	set right hand margin of input file	onInline	turn on in-line comments
minCol	set left hand margin of input file	onMargin	turn on margin marking
offFolCom	turn off end-of-line comments	onNestCom	turn on nested comments
offInling	turn off in line commonts	onTowt	on text following lines are comments
offMorrin	turn off margin marking	OHIEXU	on text – following times are comments
orinargin	turn on margin marking		
Options affe	cting input data format:		
dollor	sot the dollar character	onEmboddod	allow embedded text or data in set and parame
dorrar	set the donar character	OILLIDedded	tor statements
offDolim	delimited data statement syntax off	onEmpty	allow ampty data initialization statements
offDelim	a financia data statement syntax on	onEmpty	allow empty data initialization statements
offDigit	on number precision check	onEnd	allow alternate program control syntax
offEmbedded	no embedded text or data allowed	onEps	interpret EPS as 0
offEmpty	disallow empty data initialization statements	onGlobal	force inheritance of parent file settings
offEnd	disallow alternate program control syntax	onUNDF	allow UNDF as input
offEps	disallow interpretation of EPS as 0	onWarning	convert certain domain errors into warnings
offGlobal	disallow inheritance of parent file settings	use205	Release 2.05 language syntax
offUNDF	do not allow UNDF as input	use225	Release 2.25 Version 1 language syntax
offWarning	do not convert domain errors into warnings	use999	latest language syntax
onDelim	delimited data statement syntax on	version	test GAMS compiler version number
onDigit	on number precision check		
Options affe	cting output format:		
double	double-spaced listing follows	onEcho	start of block echo with substitution
echo	echo text	onEchoS	start of block echo with substitution
echoN	echo a string to a file without ending the line	onEchoV	start of block echo without substitution
eiect	advance to next page	onInclude	include file name echoed to listing file
hidden	ignore text and do not list	onListing	input lines echoed to listing file
lines	next number of lines have to fit on page	onLog	reset line logging
100	send message to the log	onPut	start of block put without substitution
offDollar	turn the listing of DCO lines off	onPutS	start of block put with substitution
offEcho	and of block echo	onPutV	start of block put with substitution
offIncludo	turn off listing of include file names	onVorbatim	start verbatim copy if dumpont > 10
offlicting	turn off achoing input lines to listing flo	onverbatim nomemis	start verbatilli copy if dumpopt ≥ 10
officient	turn on echoing input lines to listing me	remark	comment line with suppressed line number
offLog		single	single-spaced listing follows
offPut	end of block put	stars	set **** characters in listing file
offVerbatim	stop verbatim copy	sTitle	set subtitle and reset page
onDollar	turn the listing of DCO lines on	title	set title, reset subtitle and page
Options affe	cting listing of reference maps:		
- offSumIiet	off symbol list	on Sum I ist	on symbol list
offSumYPof	off symbol cross reference listing		on symbol cross reference listing
offuEllist	off unique element listing	onUFIList	on unique element listing
offuellist	off unique element more reference	OHUEILISU	on unique element insting
OIIUEIAREI	on unique element cross reference	ONUEIXKEI	on unique element cross insting
Options affe	cting program control:		
abort	issue an error message and abort compilation	ifThenE	if Then-else If structure with expression evalua-
			tion
batInclude	include file with substitution arguments	ifThenI	ifThen-elseIf structure with case insensitive com-
			pare
call	executes program during compilation	include	include file
call Async	executes another program asynchronously	kill	kill data connected with identifier
clear	reset all data for an identifier to its default values	label	label name as entry point from \$ roto
clearError	clear compilation errors	libInclude	include file from library directory
clearError	clear complication errors	TIDIUCIUGE	turns of slobal antions
clearErrors	clear compliation errors	offGlobal	turns off global options

	else clause	offMulti	turns off redefinition of data
elseIf	elseIf structure with case sensitive compare	offRecurse	disable recursive include files
elseIfE	elseIf structure with expression evaluation	onGlobal	turns on global options
elseIfI	elseIf structure with case insensitive compare	onMulti	turn on redefinition of data
endif	closing of ifThen/ifThenE/ifThenI control struc-	onRecurse	enable recursive include files
	ture		
error	issue an error message	maxGoto	maximum number of jumps to the same label
exit	exit from compilation	phantom	defines a phantom element
funcLibIn	load extrinsic function library	shift	DOS shift operation
rato	so to line with given label name	stop	stop compilation
biddenColl	go to fille with given laber hame	scop	include file from system directory
	executes another program (model)	systiciude	taminate according to a second second
11	conditional processing, case sensitive	terminate	terminate compliation and execution
ITE	if statement with expression evaluation	warning	issue compilation warning
111	conditional processing, case insensitive		
ifThen	if Then-else If structure with case sensitive com-		
	pare		
CDV	•		
GDX operat	tions:		
gdxIn	open GDX file for input	loadDCR	load symbols from GDX file - domain checked -
			replace
gdxOut	open GDX file for output	loadM	load symbols from GDX file - domain filtered -
Ũ			merge
load	load symbols from GDX file - domain filtered	loadR	load symbols from GDX file - domain filtered -
			replace
loadDC	load symbols from CDX file - domain checked	unload	unload symbols into CDX file
loadDCM	Load symbols from CDX file domain checked	unitoau	unioad symbols into ODA me
TOADCH	Load symbols from GDA me - domain checked -		
	merge		
Environmen	t variables:		
,			
dron	drop a scoped environment variable	set	define a scoped environment variable
urop			
dropEnv	drop an OS environment variable	setArgs	define local environment variables using argu-
dropEnv	drop an OS environment variable	setArgs	define local environment variables using argument list
dropEnv dropGlobal	drop an OS environment variable drop a global environment variable	setArgs setComps	define local environment variables using argu- ment list unpack dotted names into local variables
dropEnv dropGlobal dropLocal	drop an OS environment variable drop a global environment variable drop a local environment variable	setArgs setComps setDDList	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters
dropEnv dropGlobal dropLocal escape	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol	setArgs setComps setDDList setEnv	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable
dropEnv dropGlobal dropLocal escape eval	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari-	setArgs setComps setDDList setEnv setGlobal	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable
dropEnv dropGlobal dropLocal escape eval	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able	setArgs setComps setDDList setEnv setGlobal	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable
dropEnv dropGlobal dropLocal escape eval evalLocal	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari-	setArgs setComps setDDList setEnv setGlobal	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable
dropEnv dropGlobal dropLocal escape eval evalLocal	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able	setArgs setComps setDDList setEnv setGlobal setLocal	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable
dropEnv dropGlobal dropLocal escape eval evalLocal	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able	setArgs setComps setDDList setEnv setGlobal setLocal	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment variable
dropEnv dropGlobal dropLocal escape eval evalLocal evalGlobal	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able	setArgs setComps setDDList setEnv setGlobal setLocal setNames	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables
dropEnv dropGlobal dropLocal escape eval evalLocal evalGlobal	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able	setArgs setComps setDDList setEnv setGlobal setLocal setNames	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables
dropEnv dropGlobal dropLocal escape eval evalLocal evalGlobal prefixPath	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable	setArgs setComps setDDList setEnv setGlobal setLocal setNames show	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables
dropEnv dropGlobal dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions:	setArgs setComps setDDList setEnv setGlobal setLocal setNames show	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables
dropEnv dropGlobal dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions:	setArgs setComps setDDList setEnv setGlobal setLocal setNames show	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables
dropEnv dropGlobal dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition	setArgs setComps setDDList setEnv setClobal setLocal setNames show onDotL	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments
dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments	setArgs setComps setDDList setEnv setClobal setLocal setNames show onDotL onExpand	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu-
dropEnv dropGlobal dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand onLocal	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand onLocal	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments limit .local nesting to one	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand onLocal onMacro	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments limit .local nesting to one do not recognize macros for expansion	setArgs setComps setDDList setEnv setClobal setLocal setNames show onDotL onExpand onLocal onMacro	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments limit .local nesting to one do not recognize macros for expansion	setArgs setComps setDDList setEnv setClobal setLocal setNames show onDotL onExpand onLocal onMacro	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro Compression	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments limit .local nesting to one do not recognize macros for expansion n and encrypting of source files:	setArgs setComps setDDList setEnv setClobal setLocal setNames show onDotL onExpand onLocal onMacro	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro Compression compress	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments limit .local nesting to one do not recognize macros for expansion and encrypting of source files: create compressed GAMS system file	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand onLocal onMacro	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion hide objects from user
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro Compression compress decompress	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments limit .local nesting to one do not recognize macros for expansion n and encrypting of source files: create compressed GAMS system file decompress a GAMS system file	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand onLocal onMacro	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion hide objects from user protect objects from user modification
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro Compress decompress encrypt	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments do not expand macros when processing macro arguments limit .local nesting to one do not recognize macros for expansion n and encrypting of source files: create compressed GAMS system file decompress a GAMS system file	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand onLocal onMacro hide protect purge	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion hide objects from user protect objects from user modification remove the objects and all associated data
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro Compress decompress encrypt expose	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not assume .l for variables in assignments limit .local nesting to one do not recognize macros for expansion n and encrypting of source files: create compressed GAMS system file decompress a GAMS system file remove all access control restrictions	setArgs setComps setDDList setEnv setGlobal setLocal setNames show onDotL onExpand onLocal onMacro hide protect purge	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion hide objects from user protect objects from user modification remove the objects and all associated data
dropEnv dropEnv dropLocal escape eval evalLocal evalGlobal prefixPath Macro defin macro offDotL offExpand offLocal offMacro Compression compress decompress encrypt expose	drop an OS environment variable drop a global environment variable drop a local environment variable define the % escape symbol evaluates and defines a scoped environment vari- able evaluates and defines a local environment vari- able evaluates and defines a global environment vari- able prefix the path environment variable itions: preprocessing macro definition do not expand macros when processing macro arguments limit .local nesting to one do not recognize macros for expansion n and encrypting of source files: create compressed GAMS system file decompress a GAMS system file remove all access control restrictions r Control Ontion	setArgs setComps setDDList setEnv setClobal setLocal setNames show onDotL onExpand onLocal onMacro hide protect purge	define local environment variables using argu- ment list unpack dotted names into local variables check double dash GAMS parameters define an OS environment variable define a global environment variable define a local environment variable unpack a filename into local environment vari- ables show current GAMS environment variables assume .1 for variables in assignments expand macros when processing macro argu- ments no limit on .local nesting recognize macros for expansion hide objects from user protect objects from user modification remove the objects and all associated data

Table D.1: Dollar control options

D.3 Detailed Description of Dollar Control Options

This section describes each of the dollar control options in detail. The dollar control options are listed in alphabetical order for easy reference. In each of the following dollar control options, the default value, if available, is

abort

This option will issue a compilation error and abort the compilation.

Consider the following example,

\$if not %system.filesys% == UNIX
\$abort We only do UNIX

This attempts to stop compilation if the operating system is not Unix. Running the above example on a non-Unix platform results in the compilation being aborted, and the following listing file,

```
2 $abort We only do UNIX
**** $343
Error Messages
343 Abort triggered by above statement
```

batinclude

The **\$batinclude** facility performs the same task as the **\$include** facility in that it inserts the contents of the specified text file at the location of the call. In addition, however, it also passes on arguments which can be used inside the include file:

\$batinclude file arg1 arg2 ...

The **\$batinclude** option can appear in any place the **\$include** option can appear. The name of the batch include file may be quoted or unquoted, while **arg1**, **arg2**,.. are arguments that are passed on to the batch include file. These arguments are treated as character strings that are substituted by number inside the included file. These arguments can be single unbroken strings (quoted or unquoted) or quoted multi-part strings.

The syntax has been modeled after the DOS batch facility. Inside the batch file, a parameter substitution is indicated by using the character % followed immediately by an integer value corresponding to the order of parameters on the list where %1 refers to the first argument, %2 to the second argument, and so on. If the integer value is specified that does not correspond to a passed parameter, then the parameter flag is substituted with a null string. The parameter flag %0 is a special case that will substitute a fully expanded file name specification of the current batch included file. The flag %\$ is the current \$\$ symbol (see \$dollar). Parameters are substituted independent of context, and the entire line is processed before it is passed to the compiler. The exception to this is that parameter flags appearing in comments are not substituted.

GAMS requires that processing the substitutions must result in a line of less than or equal to the maximum input line length (currently 255 characters).

The case of the passed parameters is preserved for use in string comparisons.

Consider the following slice of code,

\$batinclude "file1.inc" abcd "bbbb" "cccc dddd"

In this case, file1.inc is included with abcd as the first parameter, bbbb as the second parameter, and cccc dddd as the third parameter.

Consider the following slice of code,

```
parameter a,b,c ;
a = 1 ; b = 0 ; c = 2 ;
$batinclude inc2.inc b a
display b ;
$batinclude inc2.inc b c
display b ;
$batinclude inc2.inc b "a+5"
display b ;
```

where inc2.inc contains the following line,

%1 = sqr(%2) - %2;

the listing file that results is as follows,
```
1 parameter a,b,c ;
2 a = 1 ; b = 0 ; c = 2 ;
BATINCLUDE D:\GAMS\INC2.INC
4 b = sqr(a) - a ;
5 display b ;
BATINCLUDE D:\GAMS\INC2.INC
7 b = sqr(c) - c ;
8 display b ;
BATINCLUDE D:\GAMS\INC2.INC
10 b = sqr(a+5) - a+5 ;
11 display b ;
```

Note that the three calls to **\$batinclude** with the various arguments lead to GAMS interpreting the contents of batch include file in turn as

```
b = sqr(a) - a ;
b = sqr(c) - c ;
b = sqr(a+5) - a+5 ;
```

Note that third call is not interpreted as sqr(a+5)-(a+5), but instead as sqr(a+5)-a+5. The results of the display statement are shown at the end of the listing file,

 5	PARAMETER	В	=	0.000
 8	PARAMETER	В	=	2.000
 11	PARAMETER	В	=	40.000

The third call leads to b = sqr(6)-1+5 which results in b taking a value of 40. If the statement in the batch include file was modified to read as follows,

%1 = sqr(%2) - (%2);

the results of the display statement in the listing file would read,

 5 F	PARAMETER	В	=	0.000
 8 F	PARAMETER	В	=	2.000
 11 F	PARAMETER	В	=	30.000

The third call leads to b = sqr(6)-6 which results in b taking a value of 30.

A \$batinclude call without any arguments is equivalent to a \$include call.

call

Passes a followed string command to the current operating system command processor and interrupts compilation until the command has been completed. If the command string is empty or omitted, a new interactive command processor will be loaded.

Consider the following slice of code,

\$call 'dir'

This command makes a directory listing on a PC.

The command string can be passed to the system and executed directly without using a command processor by prefixing the command with an '=' sign. Compilation errors are issued if the command or the command processor cannot be loaded and executed properly.

\$call 'gams trnsport'
\$call '=gams trnsport'

The first call runs [TRNSPORT] in a new command shell. The DOS command shell does not send any return codes from the run back to GAMS. Therefore any errors in the run are not reported back. The second call, however, sends the command directly to the system. The return codes from the system are intercepted correctly and available to the GAMS system through the errorlevel DOS batch function.

Some commands (like copy on a PC and cd in Unix) are shell commands and cannot be spawned off to the system. Using these in a system call will create a compilation error.

Consider the following slice of code,

\$call 'copy myfile.txt mycopy.txt'

\$call '=copy myfile.txt mycopy.txt'

The first call will work on a PC, but the second will not. The copy command can only be used from a command line shell. The system is not aware of this command (Try this command after clicking Run under the Start menu in Windows. You will find that it does not work).

call.Async

Works like **\$call** but allows asynchronous job handling. This means you can start a job without waiting for the result. You can continue in your model and collect the return code of the job later. The function JobHandle can be used to get the Process ID (pid) of the last job started. With JobStatus(pid) one could check for the status of a job. Possible return values are:

- 0: error (input is not a valid PID or access is denied)
- 1: process is still running
- 2: process is finished with return code which could be accessed by errorlevel
- 3: process not running anymore or was never running, no return code available

With JobTerminate(pid) an interrupt signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

With JobKill(pid) a kill signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

clear

This option resets all data for an identifier to its default values:

\$clear id1 id2 ...

id1, id2, ... are the identifiers whose data is being reset. Note that this is carried out during compile time, and not when the GAMS program executes. Not all data types can be cleared - only set, parameter, equation and variable types can be reset.

Consider the following example,

set i /1*20/ ; scalar a /2/ ;
\$clear i a
display i, a ;

The **\$clear** option resets **i** and **a** to their default values. The result of the display statement in the listing file shows that **i** is now an empty set, and **a** takes a value of 0.

---- 3 SET I (EMPTY) ---- 3 PARAMETER A = 0.000

The two-pass processing of a GAMS file can lead to seemingly unexpected results. Both the dollar control options and the data initialization is done in the first pass, and assignments in the second, irrespective of their relative locations. This is an issue particularly with **\$clear** since data can be both initialized and assigned.

Consider the following example,

```
scalar a /12/ ;
a=5;
$clear a
display a ;
```

The scalar data initialization statement is processed during compilation and the assignment statement a=5 during execution. In the order that it is processed, the example above is read by GAMS as,

```
* compilation step
scalar a /12/;
$clear a
* execution step
a=5;
display a;
```

The example results in a taking a value of 5. The display statement in the resulting listing file is as follows, ---- 4 PARAMETER A = 5.000

clearError(s)

\$clearError and **\$clearErrors** clear GAMS awareness of compiler errors. Consider the following example,

scalar z /11/; \$eval x sqrt(-1) \$clearerror \$log %x% display z; Without the use of \$clearerror(s) the programm would not continue with the execution.

comment (*)

This option changes the start-of-line comment to a single character which follows immediately the **\$comment** keyword. This should be used with great care, and one should reset it quickly to the default symbol *****.

The case of the start-of-line comment character does not matter when being used.

Consider the following example, \$comment c
c now we use a FORTRAN style comment symbol
\$comment *

```
* now we are back how it should be
```

compress

Causes a file to be compressed into a packed file. The syntax is **\$compress source target** where **source** is the name of the source file to be compressed and **target** is the name for the resultant compressed file. Consider the following example where the model library file TRNSPORT is used,

\$call 'gamslib trnsport' \$compress trnsport.gms t2.gms \$include t2.gms

The first command retrieves the transport file and the second command compresses it. Then later the compressed file can be solved as it is treated like any other GAMS input file.

decompress

Causes a compressed file to be decompressed into an unpacked file. The syntax is **\$decompress source target** where **source** is the name of the compressed file to be decompressed and **target** is the name for the resultant decompressed file. Consider the following example where the model library file TRNSPORT is used,

\$call 'gamslib trnsport' \$compress trnsport.gms t2.gms \$decompress t2.gms t3.gms

The first command retrieves the transport file and the second command compresses it. The third command decompresses the compressed file and t3.gms is exactly the same as the original file trnsport.gms.

dollar (\$)

This option changes the current \$ symbol to a single character which follows immediately the \$dollar keyword. When a include file is inserted, all dollar control options are inherited, and the current \$ symbol may not be known. The special %\$ substitution symbol can be used to get the correct symbol (see \$batinclude).

Consider the following example, \$dollar # #hidden now we can use # as the '\$' symbol

double

The lines following the **\$double** statement will be echoed double spaced to the listing file.

Consider the following example,

set i /1*2/ ;
scalar a /1/ ;
\$double
set j /10*15/ ;
scalar b /2/ ;

The resulting listing file looks as follows,

```
1 set i /1*2/;
2 scalar a /1/;
4 set j /10*15/;
5 scalar b /2/;
```

Note that lines before the **\$double** option are listed singly spaced, while the lines after the option are listed with double space.

drop

Destroys a variable that was defined with **\$set**. The syntax is **\$drop varname**.

dropEnv

Destroys an operating system envorinment variable. The syntax is **\$dropenv varname**. For detailed infoemation check **\$setEnv**.

dropGlobal

Destroys a variable that was defined with **\$setGlobal**. The syntax is **\$dropGlobal** varname.

dropLocal

Destroys a variable that was defined with **\$setLocal**. The syntax is **\$dropLocal** varname.

echo

The echo option allows to write text to a file:

```
$echo 'text' > file
$echo 'text' >> file
```

These options send the message 'text' to the file file. Both the text and the file name can be quoted or unquoted. The file name is expanded using the working directory. The **\$echo** statement tries to minimize file operations by keeping the file open in anticipation of another **\$echo** to be appended to the same file. The file will be closed at the end of the compilation or when a **\$call** or any kind of **\$include** statement is encountered. The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file.

Consider the following example,

```
$echo >> echo
$echo The message written goes from the first non blank
$echo 'to the first > or >> symbol unless the text is' >> echo
$echo 'is quoted. The Listing File is %gams.input%. The" >> echo
$echo 'file name "echo" will be completed with' >> echo
$echo %gams.workdir%. >> echo
$echo >> echo
```

The contents of the resulting file echo are as follows,

```
The message written goes from the first non blank
to the first > or >> symbol unless the text is
is quoted. The Listing File is C:\PROGRAM FILES\GAMSIDE\CC.GMS. The
file name "echo" will be completed with
C:\PROGRAM FILES\GAMSIDE\.
```

echoN

Sends a text message to an external file like **\$echo** but writes no end of line marker so the line is repeatedly appended to by subsequent commands. Consider the following example,

```
$echoN 'Text to be sent' > 'aaa'
$echoN 'More text' >> aaa
$echoN And more and more and more >> aaa
$echo This was entered with $echo >> 'aaa'
$echo This too >> aaa
```

The created file **aaa** contains,

Text to be sentMore textAnd more and more and moreThis was entered with $\$

The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file.

Note that the text and the file name can be quoted or unquoted. By default the file name will go in the working directory.

eject

Advances the output to the next page.

Consider the following example,

```
$eject
Set i,j ;
Parameter data(i,j) ;
$eject
```

This will force the statements between the two **\$eject** calls to be reported on a separated page in the listing file.

else

\$else always appears together with an **\$ifThen[E/I]** statement. It is followed by an instruction which is executed if the matching if statement is not true.

elseIf

\$elseIf always appears together with an **\$ifThen[E/I]** statement. It is followed by another condition and instruction.

elseIfE

\$elseIfE does the same as **\$elseIf** but evaluates numerical values of the control variables.

elseIfI

\$elseIfI does the same as **\$elseIf** but is case insensitive.

encrypt

Causes a file to be converted into an encrypted file. The syntax is **\$encrypt source target** where **source** is the name of the source file to be encrypted and **target** is the name for the resulting encrypted file. Note that encryption requires a special GAMS license. The use of the **Plicense** parameter specifies the target license to be used as a user key for decrypting. This must be done with the same license as was used in encryption. Decryption is done when reading the GAMS system files. GAMS recognizes whether a file is compressed or encrypted and will process the files accordingly.

endIf

\$endIf must be matched with an **\$ifThen**, **\$ifThenE** or **\$ifThenI**.

eolcom (!!)

This option redefines the end-of-line comment symbol, which can be a one or two character sequence. By default the system is initialized to '!!' but not active. The **\$oneolcom** option is used to activate the end-of-line comment. The **\$eolcom** option sets **\$oneolcom** automatically.

Consider the following example,

```
$eolcom ->
set i /1*2/ ; -> set declaration
parameter a(i) ; -> parameter declaration
```

The character set \rightarrow serves as the end-of-line-comment indicator.

GAMS requires that one does not reset the **\$eolcom** option to the existing symbol.

The following code is illegal since **\$eolcom** is being reset to the same symbol as it is currently, **\$eolcom** -> **\$eolcom** ->

error

This option will issue a compilation error and will continue with the next line.

Consider the following example,

\$if not exist myfile
\$error File myfile not found - will continue anyway

Weilor file myllie not lound will continue anyway

This checks if the file myfile exists, and if not, it will generate an error with the comment 'File not found - will continue anyway', and then compilation continues with the following line.

escape

Allows one to print out or display the text sequence for the % syntax. It is employed using the syntax **\$escape symbol**. This renders all subsequent commands of the form **%symbol** to not have parameter substitution done for them and on display or in a put to come out as just a %. One may reverse this action with **\$escape %**. Consider the following example,

\$set tt DOIT

file it put it

```
display "first %tt%";
display "second %&tt%&";
put "display one ", "%system.date%" /;
put "display two " "%&system.date%&"/;
$escape &
display "third %tt%";
display "fourth %&tt%&";
put "display third ", "%system.date%" /;
put "display fourth " "%&system.date%&"/;
$escape %
```

```
display "fifth %tt%";
display "sixth %&tt%&";
put "display fifth ", "%system.date%" /;
put "display sixth " "%&system.date%&"/;
```

The resulting listing file contains,

```
---- 6 first DOIT

---- 7 second %&tt%&

---- 12 third DOIT

---- 13 fourth %tt%

---- 18 fifth DOIT

---- 19 sixth %&tt%&

and it.put contains,

display one 08/10/11
```

display two %&system.date%& display third 08/10/11 display fourth %system.date% display fifth 08/10/11 display sixth %&system.date%&

This is really only present to allow one to be able to write GAMS instructions from GAMS as one would not be able to use a put to write the symbols %gams.ps% otherwise.

eval

\$eval evaluates a numerical expression at compile time and places it into a scoped control variable. In turn one can use **\$if** and **\$ife** to do numeric testing on the value of this variable. The format is **\$eval** varname expression where the expression must consist of constants, functions or other control variables with numerical values.

For differences to **\$evalGlobal** and **\$evalLocal** consider the following example,

```
$evalGlobal Anumber 3**2+2
  $evalLocal Bnumber abs(-22)
  $eval Cnumber min(33,34)
   $ife %Anumber%=11 display "Anumber equals 11"
   $ife %Bnumber%=22 display "Bnumber equals 22"
  $ife %Cnumber%=33 display "Cnumber equals 33"
  $include test.gms
  $if %Dnumber%==44 display "Dnumber equals 44"
  $if NOT %Dnumber%==44 display "Dnumber does NOT equal 44"
  $if %Enumber%==55 display "Enumber equals 55"
  $if NOT %Enumber%==55 display "Enumber does NOT equal 55"
  $if %Fnumber%==66 display "Fnumber equals 66"
  $if NOT %Fnumber%==66 display "Fnumber does NOT equal 66"
where test.gms is
  $if %Anumber%==11 display "Anumber equals 11 in test.gms"
  $if NOT %Anumber%==11 display "Anumber does NOT equal 11 in test.gms"
  $if %Bnumber%==22 display "Bnumber equals 22 in test.gms"
  $if NOT %Bnumber%==22 display "Bnumber does NOT equal 22 in test.gms"
  $if %Cnumber%==33 display "Cnumber equals 33 in test.gms"
   $if NOT %Cnumber%==33 display "Cnumber does NOT equal 33 in test.gms"
```

```
$evalGlobal Dnumber 44
   $evalLocal Enumber 55
   $eval Fnumber 66
The resulting listing file contains,
             4 Anumber equals 11
   ____
             5 Bnumber equals 22
             6 Cnumber equals 33
             9 Anumber equals 11 in test.gms
            12 Bnumber does NOT equal 22 in test.gms
   ____
   ____
            13 Cnumber equals 33 in test.gms
            20 Dnumber equals 44
            23 Enumber does NOT equal 55
            25 Fnumber does NOT equal 66
```

Note that GAMS allows one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

\$evalglobal notunique 11
\$evallocal notunique 22
\$eval notunique 33
\$log %notunique%

\$log will echo 22, the current value of notunique, to the log file.

evalGlobal

\$evalGlobal evaluates a numerical expression at compile time and places it into a global control variable. In turn one can use \$if and \$ife to do numeric testing on the value of this variable. The format is \$evalGlobal varname expression where the expression must consist of constants, functions or other control variables with numerical values.

For differences to **\$eval** and **\$evalLocal** check the example in the desription of **\$eval**.

Note that GAMS allows one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

```
$evalglobal notunique 11
$evallocal notunique 22
$eval notunique 33
$log %notunique%
```

\$log will echo 22, the current value of notunique, to the log file.

evalLocal

\$evalLocal evaluates a numerical expression at compile time and places it into a local control variable. In turn one can use \$if and \$ife to do numeric testing on the value of this variable. The format is \$evalLocal varname expression where the expression must consist of constants, functions or other control variables with numerical values.

For differences to **\$eval** and **\$evalGlobal** check the example in the desription of **\$eval**.

Note that GAMS allows one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

\$evalglobal notunique 11
\$evallocal notunique 22
\$eval notunique 33
\$log %notunique%

\$log will echo 22, the current value of notunique, to the log file.

exit

This option will cause the compiler to exit (stop reading) from the current file. This is equivalent to having reached the end of file.

Consider the following example,

```
scalar a ; a = 5 ;
display a ;
$exit
a = a+5 ; display a ;
```

The lines following the **\$exit** will not be compiled.

Note that there is a difference to \$stop. If you have only one input file \$stop and \$exit will do the same thing. If you are in an include file, \$exit acts like an end-of file on the include file. However, if you encounter a \$stop in an include file, GAMS will stop reading all input.

expose

removes all privacy restrictions from the named item or items. The syntax is

\$expose item1 item2 ...

 \mathbf{or}

\$expose all

In the first case the privacy restrictions are removed only for the listed items and in the second case they are removed for all items. One can set these privacy restrictions with **\$hide** or **\$protect**. Note that a special license file is needed for this feature to work and that the expose only takes effect in subsequent restart files.

funcLibIn

Function libraries are made available to a model using the compiler directive:

\$FuncLibIn <InternalLibName> <ExternalLibName>

Similar to sets, parameters, variables, and equations, functions must be declared before they can be used: Function <InternalFuncName> /<InternalLibName>.<FuncName>/;

gdxIn

This dollar command is used in a sequence to load specified items from a GDX file. It is employed using the syntax **\$gdxIn filename** where filename gives the name of the GDX file (with or without the extension GDX) and the command opens the specified GDX file for reading. The next use of **\$gdxIn** closes the specified GDX file.

The command must be used in conjunction with the command **\$load**.

gdxOut

This dollar command is used in a sequence to unload specified items from a GDX file. It is employed using the syntax **\$gdxOut filename** where filename gives the name of the GDX file (with or without the extension GDX) and the command opens the specified GDX file for writing. The next use of **\$gdxOut** closes the specified GDX file.

The command must be used in conjunction with the command **\$unload**.

goto id

This option will cause GAMS to search for a line starting with '\$label id' and then continue reading from there. This option can be used to skip over or repeat sections of the input files. In batch include files, the target labels or label arguments can be passed as parameters because of the manner in which parameter substitution occurs in such files. In order to avoid infinite loops, one can only jump a maximum of 100 times to the same label.

Consider the following example,

```
scalar a ; a = 5;
display a ;
$goto next
a = a+5 ; display a ;
$label next
a = a+10 ; display a ;
```

On reaching the **\$goto next** option, GAMS continues from **\$label next**. All lines in between are ignored. On running the example, a finally takes a value of 15.

The **\$goto** and **\$label** have to be in the same file. If the target label is not found in the current file, and error is issued.

hidden

This line will be ignored and will not be echoed to the listing file. This option is used to enter information only relevant to the person manipulating the file.

Consider the following example,

\$hidden You need to edit the following lines if you want to: \$hidden \$hidden 1. Change form a to b \$hidden 2. Expand the set

The lines above serve as comments to the person who wrote the file. However, these comments will not be visible in the listing file, and are therefore hidden from view.

hiddenCall

\$hiddencall does the same as **\$call** but makes sure that the statement is neither shown on the log nor the listing file.

hide

hides the named items so they cannot be displayed or computed but still allows them to be used in model calculations (.. commands when the solve statement is executed). The syntax is \$hide item1 item2 ...

or

\$hide all

In the first case the listed items are hidden and in the second case all items are hidden. These restrictions can be removed with **\$expose** or **\$purge**.

Note that special license file is needed for this feature to work.

if

The **\$if** dollar control option provides the greatest amount of control over conditional processing of the input file(s). The syntax is similar to the IF statement of the DOS Batch language:

\$if [not] {exist filename | string1 == string2} new_input_line

The syntax allows for negating the conditional with a **not** operator followed either of two types of conditional expressions: a file operation or a string comparison. The result of the conditional test is used to determine whether to read the remainder of the line, which can be any valid GAMS input line.

The exist file operator can be used to check for the existence of the given file name specification. The string compare consists of two strings (quoted or unquoted) for which the comparison result is true only if the strings match exactly. Null (empty) strings can be indicated by an empty quote: ""

- The case of the strings provided either explicitly or, more likely, through a parameter substitution, is preserved and therefore will effect the string comparison.
- Quoted strings with leading and trailing blanks are not trimmed and the blanks are considered part of the string.
- If the string to be compared is a possibly empty parameter, the parameter operator must be quoted.

New_input_line is the remainder of the line containing the **\$if** option, and could be any valid GAMS input line.

The first non-blank character on the line following the conditional expression is considered to be the first column position of the GAMS input line. Therefore, if the first character encountered is a comment character the rest of the line is treated as a comment line. Likewise if the first character encountered is the dollar control character, the line is treated as a dollar control line.

An alternative to placing new_input_line on the same line as the conditional is to leave the remainder of the line blank and place new_input_line on the line immediately following the if line. If the conditional is found to be false, either the remainder of the line (if any) is skipped or the next line is not read.

Consider the following example,

\$if exist myfile.dat \$include myfile.dat

The statement above includes the file myfile.dat if the file exists. Note that the \$ character at the beginning of the \$include option is the first non-blank character after the conditional expression, if exist myfile.dat and is therefore treated as the first column position. The above statement can also be written as

```
$if exist myfile.dat
$include myfile.dat
```

Consider the following additional examples,

```
$if not "%1a" == a $goto labelname
$if not exist "%1" display "file %1 not found" ;
```

The first statement illustrates the use of the **\$if** option inside a batch include file where parameters are passed through the **\$batinclude** call from the parent file. The **\$if** condition checks if the parameter is empty, and if not processes the **\$goto** option. Note that the string comparison attempted, "**%1a**" == a, can also be done using **%1** == "".

The second statement illustrates using standard GAMS statements if the conditional is valid. If the file name passed as a parameter through the **\$batinclude** call does not exist, the GAMS display statement is processed.

In line and end of line comments are stripped out of the input file before processing for new_input_line. If either of these forms of comments appears, it will be treated as blanks.

Consider the following example,

```
parameter a ; a=10 ;
$eolcom ! inlinecom /* */
$if exist myfile.dat
/* in line comments */  ! end of line comments
a = 4 ;
display a;
```

The fourth line is ignored, and the fifth line involving an assignment setting **a** to 4 will be treated as the result of the conditional. So the result of the display statement would be the listing of a with a value of 4 if the file myfile.dat exists, and a value of 10 if the file does not exist.

It is suggested that a \$label not appear as part of the conditional input line. The result is that if the \$label appears on the \$if line, a \$goto to this label will re-scan the entire line thus causing a reevaluation of the conditional expression. On the other hand, if the \$label appears on the next line, the condition will not be reevaluated on subsequent gotos to the label.

The following example illustrates how an unknown number of file specifications can be passed on to a batch include file that will include each of them if they exist. The batch include file could look as follows,

```
/* Batch Include File - inclproc.bch */
/* Process and INCLUDE an unknown number of input files */
$label nextfile
$if "%1a" == a $goto end
$if exist "%1" $include "%1" /* name might have blanks */
$shift goto nextfile
$label end
```

The call to this file in the parent file could look like: \$batinclude inclproc.bch fil1.inc fil2.inc fil3.inc fil4.inc

ifE

The **\$ifE** dollar control option does the same as **\$if** but allows constant expression evaluation. There are two different forms of that statement.

\$ife expr1 == expr2 true if (expr1-expr2)/(1+abs(expr2)) < 1e-12
\$ife expr true if expr1 <> 0

There are different types of relations available for the first form:

```
= to test equality
== to test equality with a tolerance
> to test greater than
< to test less than
<> to test not equal
```

Consider the following example,

```
scalar a:
   $ife (log2(16)^2)=16 a=0; display a;
   $ife log2(16)^2 == 16 a=1; display a;
   $ife NOT round(log2(16)^2-16) a=2; display a;
   $ife round(log2(16)^2-16) a=3; display a;
   $ife round(log2(16)^2-17) a=4; display a;
This will create the following ouput,
             2 PARAMETER a
                                              =
                                                        1.000
   ____
   ____
             3 PARAMETER a
                                                        2.000
   ____
             5 PARAMETER a
                                                        4.000
```

ifI

The **\$ifI** statement is working like the **\$if** statement. The only difference is that **\$if** makes comparisons involving text in a case sensitive fashion while **\$ifI** is case insensitive.

ifThen

\$ifThen is a form of an **\$if** that controls whether a number of statements are active. The syntax for the condition is generally the same as for the **\$if** statement. The **\$ifthen** and **\$elseif** have a few variants and attributes that should be mentioned:

- \$ifThen is used to do case sensitive comparisons. It must be matched with an \$endIf.
- \$ifThenE is used to do numerical comparisons. It must be matched with an \$endIf.
- \$ifThenI is used to do case insensitive comparisons. It must be matched with an \$endIf.
- \$endIf must be matched with an \$ifThen, \$ifThenE or \$ifThenI.
- **\$else** is followed by an instruction which is executed if the matching **\$ifThen** statement is not true.
- **\$elseIf** has another comparison behind it.
- **\$elseIfI** is a case insensitive variant of **\$elseIf**.
- **\$elseIfE** is a numerical value evaluating variant of **\$elseIf**.
- The statements following directly an **\$ifThen**, **\$elseIf** or **\$else** on the same line can be a sequence of other dollar control statements or contain proper GAMS syntax. The statements following directly a **\$endif** can only contain another dollar control statement.
- A NOT may be used in the commands.
- One may add a tag to the **\$ifThen** and **\$endIf**. Then for example **\$ifThen.tagone** has to match with **\$endif.tagone** as shown below.

Consider the following example which illustrates the use of **\$ifThen** and **\$elseIf**.

```
$set x a
   $label two
   $ifthen %x% == a $set x 'c' $log $ifthen with x=%x%
   selseif %x% == b set x 'k' log elseif 1 with x=%x%
   $elseif %x% == c $set x 'b' $log $elseif 2 with x=%x%
   $else
                   $set x 'e' $log $else
                                              with x=%x%
   $endif $if NOT %x% == e $goto two
the resulting log file contains
   $ifthen with x=a
   $elseif 2 with x=c
   $elseif 1 with x=b
   $else
            with x=k
The next examples illustrates the use of tags
   $ifThen.one x == y
  display "it1";
  $elseIf.one a == a
  display "it2";
   $ifThen.two c == c
  display "it3";
   $endIf.two
   $elseIf.one b == b
```

```
display "it4";
$endIf.one
```

The resulting listing file contains

---- 2 it2

-- 4 it3

because the first condition (x == y) is obviously not true and the fourth condition (b == b) is not tested because the second one (a == a) was already true.

ifThenE

\$ifThenE does the same as **\$ifThen** but evaluates numerical values of the control variables.

ifThenI

\$ifThenI does the same as \$ifThen but is case insensitive.

include

The **\$include** option inserts the contents of a specified text file at the location of the call. The name of the file to be included which follows immediately the keyword **include** may be quoted or unquoted. Include files can be nested.

The include file names are processed in the same way as the input file is handled. The names are expanded using the working directory. If the file cannot be found and no extension is given, the standard GAMS input extension is tried. However, if an incomplete path is given, the file name is completed using the include directory. By default, the library include directory is set to the working directory. The default directory can be reset with the **idir** command line parameter.

The start of the include file is marked in the compiler listing. This reference to the include file can be omitted by using the **\$offinclude** option.

The following example illustrates the use of an include statement,

\$include myfile
\$include "myfile"

Both statements above are equivalent, and the search order for the include file is as follows:

- 1. myfile in current working directory
- 2. myfile.gms in current working directory
- 3. myfile and myfile.gms (in that order) in directories specified by idir parameter.
- The current settings of the dollar control options are passed on to the lower level include files. However, the dollar control options set in the lower level include file are passed on to the parent file only if the **\$onglobal** option is set.

Compiler errors in include files have additional information about the name of the include file and the local line number.

At the end of the compiler listing, an include file summary shows the context and type of include files. The line number where an include file has been called is given. For example, in the *Include File Summary* below we see that:

SEQ	GLOBAL TYPE	PARENT	LOCAL	FILENAME
1	1 INPUT	0	0	C:\TEST\TEST1.GMS
2	1 INCLUDE	1	1	.C:\TEST\FILE1.INC
3	6 INCLUDE	1	4	.C:\TEST\FILE2.INC
з	6 INCLUDE	1	4	.C:\IESI\FILE2.INC

The first column named SEQ gives the sequence number of the input files encountered. The first row always refers the parent file called by the GAMS call. The second column named GLOBAL gives the global (expanded) line number which contained the **\$include** statement. The third column named TYPE refers to the type of file being referenced. The various types of files are INPUT, INCLUDE, BATINCLUDE, LIBINCLUDE, and SYSINCLUDE. The fourth column named PARENT provides the sequence number of the parent file for the file being referenced. The fifth column named LOCAL gives the local line number in the parent file where the **\$include** appeared. In the example listed above, the include files file1.inc and file2.inc were included on lines 1 and 4 of the parent file test1.gms.

inlinecom (/* */)

This option redefines the in-line comment symbols, which are a pair of one or two character sequence. By default, the system is initialized to /* and */, but is not active. The **\$oninline** option is used to activate the in-line comments. The **\$inlinecom** option sets the **\$oninline** automatically.

Consider the following example,

\$inlinecom {{ }}
set {{ this is an inline comment }} i /1*2/ ;

The character pair $\{\{\}\}$ serves as the indicator for in-line comments.

GAMS requires that one not reset the **\$inlinecom** option to an existing symbol.

The following code is illegal since **\$inlinecom** is being reset to the same symbol as it is currently, **\$inlinecom** {{ }} **\$inlinecom** {{ }}

The **\$onnestcom** enables the use of nested comments.

kill

Removes all data for an identifier and resets the identifier, only the type and dimension are retained. Note that this is carried out during *compile time*, and not when the GAMS program executes. Not all data types can be killed - only set, parameter, equation and variable types can be reset.

Consider the following example,

set i / 1*20 /; scalar a /2/ \$kill i a

Note that this is different from **\$clear** in the case that after setting **\$kill**, **i** and **a** are treated as though they have been only defined and have not been initialized or assigned. The result of the **\$kill** statement above is equivalent to **i** and **a** being defined as follows,

set i ; scalar a ;

Unlike the **\$clear**, a **display** statement for **i** and **a** after they are killed will trigger an error.

label id

This option marks a line to be jumped to by a **\$goto** statement. Any number of labels can be used in files and not all of them need to be referenced. Re-declaration of a label identifier will not generate an error, and only the first occurrence encountered by the GAMS compiler will be used for future **\$goto** references.

Consider the following example,

```
scalar a ; a = 5 ;
display a ;
$goto next
a = a+5 ; display a ;
$label next
a = a+10 ; display a ;
```

On reaching the **\$goto next** option, GAMS continues from **\$label next**. All lines in between are ignored. On running the example, a finally takes a value of 15.

The **\$label** statement has to be the first dollar control option of multiple dollar control options that appear on the same line.

libinclude

Equivalent to **\$batinclude**:

\$libinclude file arg1 arg2 ...

However, if an incomplete path is given, the file name is completed using the library include directory. By default, the library include directory is set to the inclib directory in the GAMS system directory. The default directory can be reset with the ldir command line parameter.

Consider the following example,

\$libinclude abc x y

This call first looks for the include file [GAMS System Directory]/inclib/abc, and if this file does not exist, GAMS looks for the file [GAMS System Directory]/inclib/abc.gms. The arguments x and y are passed on to the include file to interpret as explained for the \$batinclude option.

Consider the following example,

\$libinclude c:\abc\myinc.inc x y

This call first looks specifically for the include file c:\abc\myfile.inc. The arguments x and y are passed on to the include file to interpret as explained for the **\$batinclude** option.

lines n

This option starts a new page in the listing file if less than n lines are available on the current page.

Consider the following example,

```
$hidden Never split the first few lines of the following table
$lines 5
table io(i,j) Transaction matrix
```

This will ensure that if there are less than five lines available on the current page in the listing file before the next statement (in this case, the table statement) is echoed to it, the contents of this statement are echoed to a new page.

load

This dollar command loads specified items from a GDX file. It is employed using the syntax **\$load item1** item2 ... but must be used in conjunction with the command **\$gdxIn**.

- **\$load** must be preceded and succeeded by a **\$gdxIn**. The preceding **\$gdxIn** specifies the GDX file name and opens the file. The succeeding **\$gdxIn** closes the file. More than one **\$load** can appear in between.
- When **\$load** is not followed by arguments this causes a listing of the GDX file contents to be generated.

Consider the following example, where transsol is the GDX file of the model trnsport.gms which can be found in the GAMS Model Library.

```
$gdxin transsol
  Sets i,j;
  $load i j
  Parameters a(i), b(j) ;
   $load a
  $load b
  Scalar f:
  $load f
  $gdxin
  display i,j,a,b,f;
The resulting listing file contains,
             9 SET i canning plants
   ____
   seattle
                 san-diego
             9 SET j markets
  new-vork.
                chicago ,
                             topeka
             9 PARAMETER a capacity of plant i in cases
   seattle
             350.000.
                         san-diego 600.000
             9 PARAMETER b demand at market j in cases
   new-york 325.000,
                        chicago 300.000,
                                              topeka
                                                       275,000
             9 PARAMETER f
                                                       90.000 freight in dollars pe
   ____
                                               _
                                                               r case per thousand m
                                                               iles
```

loadDC

\$loadDC is an alternative form of **\$load** but checks to see if the set element names being loaded are in the associated sets (i.e. checks the domain). Any domain violations will be reported and flagged as compilation errors. All other features are the same as discussed under **\$load**.

Consider the following example where transsol is the GDX file of the model trnsport.gms which can be found in the GAMS Model Library.

```
set i,j;
parameter b(i),a(j);
$gdxin transsol
$load i b
$loadDC j a
$gdxin transsol
```

Note that in transsol a depends on i and b depends on j in contrast to this example. While \$load i b works and b is just empty after that line \$loadDC j a triggers a domain violation error because in transsol a depends on i.

loadDCM

\$loadDCM does the same as \$loadM plus domain checking like \$loadDC.

loadDCR

\$loadDCR does the same as \$loadR plus domain checking like \$loadDC.

loadM

\$loadM is an alternative form of **\$load**. Instead of replacing an item or causing a domain violation error if the item was already initialized it merges the contents. Consider the following example where **transsol** is the GDX file of the model trnsport.gms which can be found in the GAMS Model Library.

```
set j /1*5/;
$gdxin transsol
$loadm j
display j;
$gdxin transsol
The resulting listing file contains
---- 4 SET j markets
```

C C

```
1 , 2 , 3 , 4 , 5 , new-york
chicago , topeka
```

loadR

\$loadR item1 item2 ... will replace the parameters or sets item1 item2 ... by the data stored in the current GDX file. It must be used in conjunction with the command **\$gdxIn**. Consider the following example, where **transsol** is the GDX file of the model trnsport.gms which can be found in the GAMS Model Library.

```
sets i / 1*3 /
j / 1*2 /;
$gdxin transsol
$loadr i j
$gdxin
display i,j;
```

The resulting listing file contains,

```
---- 6 SET i canning plants
seattle , san-diego
---- 6 SET j markets
new-york, chicago , topeka
```

log

This option will send a message to the log file. By default, the log file is the console. The default log file can be reset with the lo and lf command line parameters.

Example 2 Leading blanks are ignored when the text is written out to the log file using the \$log option.

All special % symbols will be substituted out before the text passed through the **\$log** option is sent to the log file.

Consider the following example,

```
$log
$log
The following message will be written to the log file
$log with leading blanks ignored. All special % symbols will
$log be substituted out before this text is sent to the log file.
$log This was line %system.incline% of file %system.incname%
$log
```

The log file that results by running the lines above looks as follows,

```
--- Starting compilation
--- CC.GMS(0) 138 Kb
The following message will be written to the log file
with leading blanks ignored. All special % symbols will
be substituted out before this text is sent to the log file.
This was line 5 of file C:\PROGRAM FILES\GAMSIDE\CC.GMS
--- CC.GMS(7) 138 Kb
--- Starting execution - empty program
```

*** Status: Normal completion

Note that %system.incline% has been replaced by 5 which is the line number where the string replacement was requested. Also note that %system.incname% has been substituted by the name of the file completed with the absolute path. Also note that the leading blanks on the second line of the example are ignored.

macro

GAMS includes the ability to define macros as of version 22.9. The definition takes the form **\$macro** name(arg1,arg2,arg3,...) body where name is the name of the macro which has to be unique, arg1,arg2,arg3,... are the arguments and body defines what the macro should do. Consider the following example,

```
scalar x1 /2/, x2 /3/, y;
  $macro oneoverit(y) 1/y
  y = oneoverit(x1);
  display y;
  $macro ratio(a,b) a/b
  y = ratio(x2,x1);
  display y;
The resulting listing file contains,
   ____
            5 PARAMETER y
                                                      0.500
            9 PARAMETER y
                                                      1.500
Macros can also be included within macros,
  $macro product(a,b) a*b
  $macro addup(i,x,z) sum(i,product(x(i),z))
when called with
  z = addup(j,a1,x1);
will expand into:
  z = sum(j,a1(j)*x1);
The recognition of macros and expansion of arguments can be more carefully controlled by the use of
ampersands (&) in the macro body. Ordinarily the macro will only substitute for full words thus the
macro group
  $macro f(i) sum(j, x(i,j))
  $macro equh(q) equation equ_q(i); equ_q(i).. q =e= 0;
  equh(f(i))
```

which would expand to become

equation equ_q(i); equ_q(i).. sum(j, x(i,j)) =e= 0;

Note this contains q in a number of other places. If one wished to replace some of them as well one could use

```
$macro f2(r,i) sum(j, r(i,j))
$macro equ2(z,d,q) equation equ2_&z&d; equ2_&z&d.. z*q =e= 0;
equ2(1,(i),f2(x,i))
equ2(2,(k),f2(r,k))
which would expand to become
equation equ2_1(i); equ2_1(i).. 1*sum(j, x(i,j)) =e= 0;
equation equ2_2(k); equ2_2(k).. 2*sum(j, r(k,j)) =e= 0;
```

where the &z and &d are replaced.

One can also include expressions with spaces, commas and unbalanced parentheses using && which includes an expression removing the outer set of quotes.

```
$macro d(q) display &&q;
$macro ss(q) &&q)
d('"hereit is" , i,k')
d('"(zz"')
z=ss('sum(j,a1(j)');
z=ss('prod(j,a1(j)');
```

Note that the d expressions contain quotes, spaces and commas and the ss expression has unbalanced parentheses within the quoted parts.

In turn these expand to become

```
display "hereit is" , i,k;
display "(zz";
z=sum(j,a1(j));
z=prod(j,a1(j));
```

Nested macro use can result in an expansion of infinite length. For example:

```
$macro a b,a
display a;
```

will expand into:

Another feature of macros is the implicit use of the .L suffix in report writing and other data manipulation statements. This allows using the same algebra in model definitions and assignment statements. The following code illustrates this feature,

```
$macro sumit(i,term) sum(i,term)
                  z =e= sumit((i,j), (c(i,j)*x(i,j)));
    cost ..
    supply(i) .. sumit(j, x(i,j)) =l= a(i) ;
    demand(j) .. sumit(i, x(i,j)) =g= b(j);
    Model transport /all/ ;
    Solve transport using lp minimizing z ;
  $onDotL
  parameter tsupply(i) total demand for report
            tdemand(j) total demand for report;
            tsupply(i)=sumit(j, x(i,j));
            tdemand(j)=sumit(i, x(i,j));
which will expand to:
                z =e= sum((i,j),(c(i,j)*x(i,j)));
  cost ..
  supply(i) .. sum(j,x(i,j)) =l= a(i) ;
  demand(j) .. sum(i,x(i,j)) = g = b(j);
```

```
Model transport /all/;
Solve transport using lp minimizing z ;
parameter tsupply(i) total demand for report
tdemand(j) total demand for report;
tsupply(i)=sum(j,x.L(i,j));
tdemand(j)=sum(i,x.L(i,j));
```

The **\$onDotL** enables the implicit .L suffix for variables. This feature was introduced to make macros more useful and is not limited to be used in macro bodies. The matching **\$offDotL** will disable this feature.

maxcol n (80000)

Sets the right margin for the input file. All valid data is before and including column n in the input file. All text after column n is treated as comment and ignored.

Consider the following example,

\$maxcol 30	
set i /vienna, rome/	set definition
scalar a $/2.3/$;	scalar definition

The text strings 'set definition' and 'scalar definition' are treated as comments and ignored since they begin on or after column 31.

Any changes in the margins via maxcol or mincol will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option \$mincol 20 maxcol 110 will trigger the message:

NEW MARGIN = 20-110

GAMS requires that the right margin set by *\$maxcol* is greater than 15.

GAMS requires that the right margin set by \$maxcol is greater than the left margin set by \$mincol.

maxGoto 100

Sets the maximum number of jumps to the same label. Once the maximum number is reached an error is triggered. Consider the following example,

```
scalar a / 1 /;
$maxgoto 5
$label label1
a = a+10;
display a ;
$goto label1
```

When '\$goto label1' is called for the fifth time an error is triggered.

mincol n (1)

Sets the left margin for the input file. All valid data is after and including column n in the input file. All text before column n is treated as comment and ignored.

Consider the following example,

\$mincol 30	
set definition	set i /vienna, rome/
scalar definition	scalar a $/2.3/$;

The text strings 'set definition' and 'scalar definition' are treated as comments and ignored since they begin before column 30.

Any changes in the margins via maxcol or mincol will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option \$mincol 20 maxcol 110 will trigger the message:

NEW MARGIN = 20-110

GAMS requires that the left margin set by \$mincol is smaller than the right margin set by \$maxcol.

[on|off]Delim (\$offDelim)

Controls whether data in table statements are in comma delimited format. Consider running the following slice of code,

```
SETS
    PLANT PLANT LOCATIONS /NEWYORK,CHICAGO,LOSANGLS /
    MARKET DEMANDS /MIAMI,HOUSTON, PORTLAND/
table dist(plant,market)
$ondelim
,MIAMI,HOUSTON,PORTLAND
NEWYORK,1300,1800,1100
CHICAG0,2200,1300,700
```

LOSANGLS,3700,2400,2500 \$offdelim

display dist;

The resulting listing file contains,

---- 12 PARAMETER dist

	MIAMI	HOUSTON	PORTLAND
NEWYORK	1300.000	1800.000	1100.000
CHICAGO	2200.000	1300.000	700.000
LOSANGLS	3700.000	2400.000	2500.000

[on|off]Digit (\$onDigit)

Controls the precision check on numbers. Computers work with different internal precision. Sometimes a GAMS problem has to be moved from a machine with higher precision to one with lower precision. Instead of changing numbers with too much precision the **\$offdigit** tells GAMS to use as much precision as possible and ignore the rest of the number. If the stated precision of a number exceeds the machine precision an error will be reported. For most machines, the precision is 16 digits.

Consider running the following slice of code, parameter y(*) / toolarge 12345678901234.5678 \$offdigit ignored 12345678901234.5678 / The resulting listing file contains, 1 parameter y(*) / toolarge 12345678901234.5678 ****

3 ignored 12345678901234.5678 / Error Messages 103 Too many digits in number (\$offdigit can be used to ignore trailing digits)

Note that the error occurs in the 17th significant digit of y("toolarge"). However, after the **\$offdigit** line, y("ignored") is accepted without any errors even though there are more than 16 significant digits.

[on|off]Dollar (\$offDollar)

This option controls the echoing of dollar control option lines in the listing file.

Consider running the following slice of code, \$hidden this line will not be displayed
\$ondollar
\$hidden this line will be displayed
\$offdollar
\$hidden this line will not be displayed

The listing file that results looks like,

2 \$ondollar

3 \$hidden this line will be displayed

Note that all lines between the **\$ondollar** and **\$offdollar** are echoed in the listing file. Also note that this action of this option is immediate, i.e. the **\$ondollar** line is itself echoed on the listing file, while the **\$offdollar** line is not.

[on|off]DotL (\$offDotL)

Activates or deactivates the automatic addition of .L to variables on the right hand side of calculations as described at \$macro.

[on|off]Echo

Sends multiple subsequent lines to an external file. These commands are used employing the syntax

```
$0necho > externalfile1
line 1 to send
line 2 to send
$0ffecho
Or
$0necho >> externalfile2
line 1 to send
line 2 to send
$0ffecho
```

In both cases the created file contains

line 1 to send line 2 to send

The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file.

There is also a variant **\$onEchoS** that permits parameter substitution as **\$onEcho** also does and another variant **\$onEchoV** that forbids parameter substitution. Consider the following example,

\$set it TEST \$OnechoS > externalfile1 send %it% to external file line 2 to send \$Offecho

```
The created file contains
send TEST to external file
line 2 to send
while the file created by
$set it TEST
$OnechoV > externalfile1
send %it% to external file
line 2 to send
$Offecho
contains
send %it% to external file
```

send %it% to external file line 2 to send

In contrast to the second case in the first case %it% is substituded by TEST. Note that when there is no path included the file by default will be placed in the working directory.

[on|off]Embedded (\$offEmbedded)

Enables or disables the use of embedded values in parameter and set data statements. For sets, the final text is concatenated with blank separators. For example, the element texts for the set i and j will be identical:

[on|off]Empty (\$offEmpty)

This option allows empty data statements for list or table formats. By default, data statements cannot be empty.

Consider running the following slice of code,

```
set i /1,2,3/;
set j(i) / /;
parameter x(i) empty parameter / /;
table y(i,i) headers only
        1 2 3
;
$onempty
set k(i) / /;
parameter xx(i) empty parameter / /;
table yy(i,i)
        1 2 3
```

```
The listing file that results looks like,
```

```
1 set i /1,2,3/ ;
   2 set j(i) / / ;
****
                 $460
   3 parameter x(i) empty parameter / /;
****
                                      $460
   4 table y(i,i) headers only
   5
          1 2
                 3
   6;
      $462
****
   8 set k(i) / /;
   9 parameter xx(i) empty parameter / / ;
  10 table yy(i,i)
          1 2 3
  11
  12
      ;
```

Error Messages

460 Empty data statements not allowed. You may want to use \$ON/OFFEMPTY 462 The row section in the previous table is missing

Note that empty data statements are not allowed for sets, parameters or tables. These are most likely to occur when data is being entered into the GAMS model by an external program. Using the **\$onempty** dollar control option allows one to overcome this problem.

The empty data statement can only be used with symbols, which have a known dimension. If the dimension is also derived from the data, the **\$phantom** dollar control option should be used to generate 'phantom' set elements.

[on|off]End (\$offEnd)

Offers alternative syntax for flow control statements. Endloop, endif, endfor, and endwhile are introduced as key-words with the use of the **\$onend** option that then serves the purpose of closing the **loop**, if, for, and while language constructs respectively.

The following example provides the alternate syntax for the four language constructs mentioned above (standard syntax as eolcomment).

```
set i/1*3/ ; scalar cond /0/;
parameter a(i)/1 1.23, 2 2.65, 3 1.34/;
$maxcol 40
$onend
loop i do
                                           loop (i.
 display a;
                                            display a;
endloop;
                                           );
if (cond) then
                                           if (cond,
  display a;
                                            display a;
else
                                           else
  a(i) = a(i)/2;
                                            a(i) = a(i)/2;
  display a;
                                            display a;
endif;
                                          );
for cond = 1 to 5 do
                                           for (cond = 1 to 5,
  a(i) = 2 * a(i);
                                            a(i) = 2 * a(i);
endfor:
                                           ):
while cond < 2 do
                                           while (cond < 2,
  a(i) = a(i) / 2;
                                            a(i) = a(i) / 2;
endwhile:
                                          ):
```

Note that the alternate syntax is more in line with syntax used in some of the popular programming languages.

Both forms of the syntax will never be valid simultaneously. Setting the **\$onend** option will make the alternate syntax valid, but makes the standard syntax invalid.

[on|off]EolCom (\$offEolCom)

Switch to control the use of end-of-line comments. By default, the end-of-line comment symbol is set to '!!' but the processing is disabled.

Consider running the following slice of code,

\$oneolcom
set i /1*2/ ; !! set declaration
parameter a(i) ; !! parameter declaration

Note that comments can now be entered on the same line as GAMS code.

🖙 \$eolcom automatically sets \$oneolcom.

Consider the following example,

\$eolcom ->
set i /1*2/ ; -> set declaration
parameter a(i) ; -> parameter declaration

The character set \rightarrow serves as the end-of-line-comment indicator.

[on|off]Eps (\$offEps)

This option is used to treat a zero as an EPS in a parameter or table data statement. This can be useful if one overloads the value zero with existence interpolation.

Consider running the following slice of code,

The result of the display statement in the listing file is as follows,

one EPS, three EPS

Note that only those entries specifically entered as 0 are treated as EPS.

[on|off]Expand (\$offExpand)

Changes the processing of macros appearing in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch **\$onExpand** enables the recognition and expansion of macros in the macro argument list and **\$offexpand** will restore the default behavior.

[on|off]Global (\$offGlobal)

When an include file is inserted, it inherits the dollar control options from the higher level file. However, the dollar control option settings specified in the include file do not affect the higher level file. This convention is common among most scripting languages or command processing shells. In some cases, it may be desirable to break this convention. This option allows an include file to change options of the parent file as well.

Consider running the following slice of code,

```
$include 'inc.inc'
$hidden after first call to include file
$onglobal
$include 'inc.inc'
$hidden after second call to include file
```

where the file inc.inc contains the lines,

\$ondollar
\$hidden text inside include file

The resulting listing file is as follows,

```
INCLUDE D:\GAMS\INC.INC
2 $ondollar
3 $hidden text inside include file
INCLUDE D:\GAMS\INC.INC
7 $ondollar
8 $hidden text inside include file
9 $hidden after second call to include file
```

Note that the effect of the **\$ondollar** dollar control option inside the include file does not affect the parent file until **\$onglobal** is turned on. The **\$hidden** text is then echoed to the listing file.

[on|off]Include (\$onInclude)

Controls the listing of the expanded include file name in the listing file.

Consider running the following slice of code,

```
$include 'inc.inc'
$offinclude
$include 'inc.inc'
where the file inc.inc contains the line,
$ondollar
$hidden text inside include file
```

The resulting listing file is as follows,

INCLUDE D:\GAMS\INC.INC

- 2 \$ondollar
- 3 \$hidden text inside include file
- 6 \$ondollar
 7 \$hidden text inside include file

Note that the include file name is echoed the first time the include file is used. However, the include file name is not echoed after **\$offinclude** is set.

[on|off]Inline (\$\$ (\$ offInline)

Switch to control the use of in-line comments. By default, the in-line comment symbols are set to the two character pairs '/*' and '*/' but the processing is disabled. These comments can span lines till the end-of-comment characters are encountered.

Consider running the following slice of code,

```
$oninline
/* the default comment symbols are now
   active. These comments can continue
  to additional lines till the closing
   comments are found */
```

🖙 \$inlinecom automatically sets \$oninline.

Consider running the following slice of code,

```
$inlinecom << >>
<< the in-line comment characters have been
  changed from the default. >>
```

Nested in-line comments are illegal unless \$onnestcom is set.

[on|off]Listing (\$onListing)

Controls the echoing of input lines to the listing file. Note that suppressed input lines do not generate entries in the symbol and reference sections appearing at the end of the compilation listing. Lines with errors will always be listed.

Consider running the following slice of code, set i /0234*0237/ j /a,b,c/ table x(i,j) very long table a b С 0234 1 2 3 \$offlisting 5 6 0235 4

The resulting listing file looks as follows,

```
1 set i /0234*0237/
2
       j /a,b,c/
3
  table x(i,j) very long table
4
           a b
                  с
5
    0234
               2
                   3
           1
    0237
10
           1
               1
                   1
```

Note that the lines in the source file between the **\$offlisting** and **\$onlisting** settings are not echoed to the listing file.

[on|off]Local (\$onLocal)

5

1

6 7

1 1

0236

\$onlisting 0237

\$onLocal allows unlimited use of .local on the same symbol in one control stack while **\$offLocal** limits the use to one. Consider the following example,

```
set i /1*3/; alias(i,j);
parameter xxx(i,j) / 1.1 1, 2.2 2, 3.3 3, 1.3 13, 3.1 31 /;
display xxx;
parameter g(i,i);
g(i.local-1,i.local) = xxx(i,i);
                                  display g;
$offlocal
g(i.local-1,i.local) = xxx(i,i)+1; display g;
```

The use of **\$offlocal** causes a compilation error in the following line because .local is used twice on the same symbol in one control stack.

on|off|Log (\$onLog)

Turns on/off line logging for information about the line number and memory consumption during compilation. This is scoped like the **\$on/offListing** applying only to included files and any subsequent included files but reverting to **\$onlog** in the parent files i.e. when file1 includes file 2 and file 2 contains \$offlog then subsequent lines in file 2 will not be logged but lines in file 1 will be.

[on|off]Macro (\$onMacro)

```
Enables or disables the expansion of macros. For example
$macro oneoverit(y) 1/y
$offmacro
y = oneoverit(x1);
display y;
```

causes an error because the macro oneoverit can not be expanded in line 3.

[on|off]Margin (\$offMargin)

Controls the margin marking. The margins are set with \$mincol and \$maxcol.

Consider running the following slice of code,

\$onmargin mincol 20	maxcol 45	
Now we have	set i plant /US, UK/	This defines I
turned on the	scalar x / 3.145 /	A scalar example.
margin marking.	parameter a, b;	Define some
		parameters.

\$offmargin

Any statements between columns 1 and 19, and anything beyond column 45 are treated as comments.

[on|off]Multi (\$offMulti)

Controls multiple data statements or tables. By default, GAMS does not allow data statements to be redefined. If this option is enabled, the second or subsequent data statements are merged with entries of the previous ones. Note that all multiple data statements are executed before any other statement is executed.

Consider running the following slice of code,

```
$eolcom !
set i / 1*10 /;
parameter x(i) / 1*3 1 / ! 1=1,2=1,3=1
$onmulti
parameter x(i) / 7*9 2 / ! 1=1,2=1,3=1,7=2,8=2,9=2
parameter x(i) / 2*6 3 / ! 1=1,2=3,3=3,4=3,5=3,6=3,7=2,8=2,9=2
parameter x(i) / 3*5 0 / ! 1=1,2=3,6=3,7=2,8=2,9=2
display x;
```

This would have been illegal without the presence of the **\$onmulti** option. The result of the display statement in the listing file is as follows,

---- 8 PARAMETER X

1 1.000, 2 3.000, 6 3.000, 7 2.000, 8 2.000, 9 2.000

Note that x("1") has a value of 1 after the first data statement since none of the subsequent data statements affect it. x("2") on the other hand is reset to 3 by the third data statement.

The two-pass processing of a GAMS file can lead to seemingly unexpected results. Both the dollar control options and the data initialization is done in the first pass, and assignments in the second, irrespective of their relative locations. This is an issue particularly with *\$onmulti* since it allows data initializations to be performed more than once.

Consider the following example,

```
scalar a /12/ ;
a=a+1;
$onmulti
scalar a /20/ ;
display a ;
```

The two scalar data initialization statements and the **\$onmulti** option are processed before the assignment statement a=a+1. In the order that it is processed, the example above is read by GAMS as,

```
* compilation step
scalar a /12/ ;
$onmulti
scalar a /20/ ;
* execution step
a=a+1;
display a ;
```

The example results in a taking a value of 21. The display statement in the resulting listing file is as follows,

---- 5 PARAMETER A = 21.000

[on|off]NestCom (\$offNestCom)

Controls nested in-line comments. Make sure that the open-comment and close-comment characters have to match.

Consider running the following slice of code,

\$inlinecom { } onnestcom

{ nesting is now possible in comments { braces
 have to match } }

[on|off]Put

Causes a block of text to be placed in a put file. These commands are used employing the syntax file putfile

```
put putfile
$onPut
Line 1 of text
Line 2 of text
Line 3 of text
$offPut
The resulting file putfile.put contains
Line 1 of text
Line 2 of text
```

Line 3 of text Line 4 of text

There is also a variant **\$onPutS** that permits parameter substitution and another variant **onPutV** that forbids parameter substitution as **onPut** also does. Consider the following example,

```
$set it TEST
  file putfile
  put putfile
   $onPutS
  Line 1 of text "%it%"
  Line 2 of text %it%
   $offPut
The created putfile contains
  Line 1 of text "TEST"
  Line 2 of text TEST
while the putfile created by
   $set it TEST
  file putfile
  put putfile
   $onPutV
  Line 1 of text "%it%"
  Line 2 of text %it%
   $offPu
contains
  Line 1 of text "%it%"
  Line 2 of text %it%
```

In contrast to the second case in the first case %it% is substituded by TEST.

on|off|Recurse \$offRecurse

Allows a file to include itself. For example a file called **file1.gms** can contain **\$onrecurse \$include file1**

Note that the maximum include nesting level is 40 and if it is exceeded an error is triggered.

[on|off]SymList (\$offSymList)

Controls the complete listing of all symbols that have been defined and their text, including pre-defined functions and symbols, in alphabetical order grouped by symbol type.

The symbol listing in the listing file generated by running [TRNSPORT] with **\$onsymlist** is as follows,

Symbol Listing SETS Ι canning plants J markets PARAMETERS capacity of plant i in cases Α В demand at market j in cases С transport cost in thousands of dollars per case D distance in thousands of miles F freight in dollars per case per thousand miles VARIABLES X shipment quantities in cases total transportation costs in thousands of dollars Ζ EQUATIONS COST define objective function DEMAND satisfy demand at market j SUPPLY observe supply limit at plant i MODELS TRANSPORT FILES FILE Current file name for FILE.xxx use PREDEFINED DTAG SAMEAS

This serves as a simple description of the symbols used in a model, and can be used in reports and other documentation.

[on|off]SymXRef (\$offSymXRef)

This option controls the following,

- \succ Collection of cross references for identifiers like sets, parameters, and variables.
- > Cross-reference report of all collected symbols in listing file
- ➤ Listing of all referenced symbols and their explanatory text by symbol type in listing file. This is also reported by using \$onsymlist.

Consider the following slice of code,

```
$onsymxref
set i / 1*6/,k;
$offsymxref
set j(i) will not show / 1*3 /
display i;
$onsymxref
k('1') = yes;
```

The resulting listing file will contain the following unique element reference reports,

SYMBOL TYPE REFERENCES

i k	SET SET	declared declared	2 2	defined assigned	2 7
SETS					
i k					

[on|off]Text

The **\$onText** - **\$offText** pair encloses comment lines. Line numbers in the compiler listing are suppressed to mark skipped lines.

Consider the following,

```
* standard comment line
$ontext
Everything here is a comment
until we encounter the closing $offtext
like the one below
$offtext
* another standard comment line
```

The resulting listing file is as follows,

```
1 * standard comment line
Everything here is a comment
until we encounter the closing $offtext
like the one below
7 * another standard comment line
```

GAMS requires that every **\$ontext** has a matching **\$offtext**, and vice versa.

[on|off]UElList (\$offUElList)

This option controls the complete listing of all set elements that have been entered, in the listing file.

The unique element listing in the listing file generated by running [TRNSPORT] with **\$onUElList** is as follows,

Unique Element Listing

Unique Elements in Entry Order 1 SEATTLE SAN-DIEGO NEW-YORK CHICAGO TOPEKA Unique Elements in Sorted Order 1 CHICAGO NEW-YORK SAN-DIEGO SEATTLE TOPEKA

Note that the sorted order is not the same as the entry order. This is explained in Section 12.2.

[on|off]UElXRef (\$offUElXRef)

This option controls the collection and listing (in the listing file) of cross references of set elements.

Consider the following slice of code,

```
$onuelxref
set i this is set declaration / one, two, three /, k(i)
$offuelxref
set j(i) will not show / two /;
$onuelxref
k('one') = yes;
```

The resulting listing file will contain the following unique element reference reports,

REFERE			
DECLARED	2	INDEX	6
DECLARED	2		
DECLARED	2		
	REFERE DECLARED DECLARED DECLARED	DECLARED 2 DECLARED 2 DECLARED 2 DECLARED 2	DECLARED 2 INDEX DECLARED 2 DECLARED 2 DECLARED 2

[on|off]UNDF (\$offUNDF)

Controls the use of the special value UNDF which indicates a result is undefined. A user may not use this in an assignment unless the **\$onUNDF** command has been used. The following slice of code

scalar x; \$onUNDF x=UNDF; display x;

4 PARAMETER x

creates the following part at the listing file

UNDF

Without the use of **\$onUndf** an error would be triggered. The use of **\$offUNDF** will disallow the usage of UNDF in assignments.

[on|off]Verbatim

These commands are used in conjunction with the GAMS parameter DUMPOPT to suppress the input preprocessing for input lines that are copied to the dmp file. This feature is mainly used to maintain different versions of related models in a central environment.

The Son/offVerbatim commands are only recognized for $\texttt{DUMPOPT} \ge 10$ and apply only to lines in the file the commands appeared.

The use of **\$goto** and **\$on/offVerbatim** are incompatible and may produce unexpected results. Consider the following example,

\$set f 123
\$log %f%
\$onVerbatim
\$log %f%
\$offverbatim
\$log %f%
The corresponding dmp file contains
\$log 123
\$onVerbatim
\$log %f%
\$offVerbatim
\$log %f%

[on|off]Warning (\$offWarning)

Switch for data domain checking. In some cases it may be useful to accept domain errors in data statements that are imported from other systems and report warnings instead of errors. Data will be accepted and stored, even though it is outside the domain.

This switch effects three types of domain errors usually referred to as error numbers 116, 170 and 171.

This can have serious side affects and one has to exercise great care when using this feature.

```
Consider the following slice of code,
set i / one,two,three /
$onwarning
    j(i) / four, five /
    k / zero /;
parameter x(i) Messed up Data / one 1.0, five 2.0 /;
x('six') = 6; x(j) = 10; x('two') = x('seven');
j(k) = yes;
$offwarning
```

Note that the set j, although specified as a subset of i, contains elements not belonging to its domain. Similarly, the parameter x contains data elements outside its domain. The skeleton listing file that results from running this code is as follows,

```
1 set i
             / one,two,three /
  3
         j(i) / four, five /
****
                    $170 $170
  4
               / zero /;
         k
  5
     parameter x(i) Messed up Data / one 1.0, five 2.0 /;
                                                 $170
****
  6 x('six') = 6; x(j) = 10; x('two') = x('seven');
****
           $170
                                                 $116.170
  7
     j(k) = yes;
****
        $171
  9 display i,j,x;
 Error Messages
116 Label is unknown
170 Domain violation for element
171 Domain violation for set
**** 0 ERROR(S)
                7 WARNING(S)
Execution
         9 SET i
____
one
         two ,
                   three
         9 SET j
four.
         five.
                 zero
         9 PARAMETER x Messed up Data
____
     1.000,
               four 10.000,
                               five 10.000,
                                                    6.000
one
                                               six
```

The domain violations are marked like normal compilation errors but are only treated as warnings and one can execute the code.

phantom id

Used to designate id as a phantom set element. Syntactically, a phantom element is handled like any other set element. Semantically, however, it is handled like it does not exist. This is sometimes used to specify a data template that initializes the phantom records to default values.

Consider the following example,

```
$phantom null
set i / null/
    j / a,b,null/;
display i,j;
The resulting section of the listing file is shown below,
---- 4 SET I
```

(EMPTY) ---- 4 SET J a, b

Note that null does not appear in the listing file.

Assignment statements on the phantom label are ignored.

```
Consider the following extension to the previous example,
Parameter p(j) / a 1, null 23 / ;
display p ;
The resulting section of the listing file is shown below.
```

The resulting section of the listing file is shown below,

```
--- 6 PARAMETER P
```

a 1.000

prefixPath

Augments the search path in the Windows path environment variable. The use of **\$prefixpath value** results in the text in **value** being appended to the beginning of the search path. Consider the following example,

```
display "%sysenv.PATH%";
$prefixpath C:\somewhereelse\anotherpath
display "%sysenv.PATH%";
```

If %sysenv.PATH% contains C:\some\path\somehwere in the beginning it will also contain

```
C:\somewhereelse\anotherpath after the second line and the resulting listing file contains
```

```
---- 1
C:\some\path\somehwere
---- 3
```

C:\somewhereelse\anotherpath;C:\some\path\somehwere

protect

Freezes all values of the named parameters not allowing modification but still allowing their use in model calculation (.. commands when models are set up) in a privacy setting. The syntax is \$protect item1 item2 ...

```
abiorect itemi ite
```

or

```
$protect all
```

where one can list multiple items to protect them. The word **all** causes protection of all items. These privacy restrictions can be removed with **\$expose** or **\$purge**.

purge

removes the items and all data associated in a privacy setting. The syntax is

\$purge item1 item2 ...

or

\$purge all

In the first case only the listed items are removed, in the second case all items are removed. One can set the corresponding privacy restrictions with **\$hide** or **\$protect**.

Note that a special license file is needed for this feature to work and that the removal only takes effect in the restart files.

remark

Adds a comment to the listing file with parameter substitution and suppressed line number. Consider the following example,

\$set it TEST
\$remark write %it% to the listing file

The resulting listing file contains write TEST to the listing file

set

Establishes or redefines contents of a control variable that is accessible in the code where the command appears and all code included therein. The syntax is **\$set varname expression** where **varname** is any user chosen variable name and **expression** is optional and can contain text or a number. These variables are destroyed using **\$drop varname**.

In contrast to **\$eval** the **\$set** command does not evaluate the expression at compile time.

For differences to **\$setGlobal** and **\$setLocal** consider the following example,

```
$setGlobal Anumber 3**2+2
   $setLocal Bnumber 44/2
   $set Cnumber min(33.34)
   $ife %Anumber%=11 display "Anumber equals 11"
  $ife %Bnumber%=22 display "Bnumber equals 22"
  $ife %Cnumber%=33 display "Cnumber equals 33"
  $include test2.gms
  $if %Dnumber%==44 display "Dnumber equals 44"
  $if NOT %Dnumber%==44 display "Dnumber does NOT equal 44"
  $if %Enumber%==55 display "Enumber equals 55"
   $if NOT %Enumber%==55 display "Enumber does NOT equal 55"
  $if %Fnumber%==66 display "Fnumber equals 66"
  $if NOT %Fnumber%==66 display "Fnumber does NOT equal 66"
where test2.gms is
  $ife %Anumber%==11 display "Anumber equals 11 in test2.gms"
  $ife NOT %Anumber%==11 display "Anumber does NOT equal 11 in test2.gms"
  $if %Bnumber%==22 display "Bnumber equals 22 in test2.gms"
  $if NOT %Bnumber%==22 display "Bnumber does NOT equal 22 in test2.gms"
  $ife %Cnumber%==33 display "Cnumber equals 33 in test2.gms"
  $ife NOT %Cnumber%==33 display "Cnumber does NOT equal 33 in test2.gms"
  $setGlobal Dnumber 44
  $setLocal Enumber 55
  $set Fnumber 66
The resulting listing file contains,
   ____
             4 Anumber equals 11
             5 Bnumber equals 22
   ____
             6 Cnumber equals 33
             9 Anumber equals 11 in test2.gms
            12 Bnumber does NOT equal 22 in test2.gms
   ____
   ____
            13 Cnumber equals 33 in test2.gms
   ____
            20 Dnumber equals 44
   ____
            23 Enumber does NOT equal 55
            25 Fnumber does NOT equal 66
```

Note that GAMS allows one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

\$setglobal notunique aa
\$setlocal notunique bb
\$set notunique cc
\$if "%notunique%" == "aa" display "it is aa";
\$if "%notunique%" == "bb" display "it is bb";

\$if "%notunique%" == "cc" display "it is cc";

The resulting listing file contains, ---- 5 it is bb

setArgs

Sets up substitutable parameters as GAMS control variable names. Consider the following example, scalar a /2/, b /4/, c /5/; \$batinclude test3 a b c

```
where test3.gms is,
    scalar x;
    x = %1 + %2 * %3 ;
    display x;
    $setargs aa bb cc
    x = %aa% - %bb% * %cc% ;
    display x;
    x = %1 + %2 * %3 ;
    display x;
```

The **\$setArgs** command that must appear in the Batincluded file allows this Batinclude file to use **%aa%** in place of **%1**, **%bb%** in place of **%2** and **%cc%** in place of **%3**. Note that the use of **%1**, **%2** etc. is still allowed. The resulting listing file contains,

 5 PARAMETER x	=	22.000
 8 PARAMETER x	=	-18.000
 10 PARAMETER x	=	22.000

setComps

Establishes or redefines control variables so they contain the components of a period delimited string. The syntax is **\$setcomps perioddelimstring v1 v2 v3 ...** where **perioddelimstring** is any period delimited string like the set specification of a multidimensional parameter and

v1 is the name of a control variable that will contain the name of the set element in the first position v2 is the name of a control variable that will contain the name of the set element in the second position v3 is the name of a control variable that will contain the name of the set element in the third position. The items may be recombined back into the original filename string by using %v1%.%v2%.%v3%. Consider the following example,

\$setcomps period.delim.string v1 v2 v3
display "v1 = %v1%";
display "v2 = %v2%";
display "v3 = %v3%";
\$set name %v1%.%v2%.%v3%
display "name = %name%";

The resulting listing file contains,

2 v1 = period

---- 3 v2 = delim ---- 4 v3 = string ---- 6 name = period.delim.string

setDDList

Causes GAMS to look for misspelled or undefined 'double dash' GAMS parameters. Consider the following example:

The following 'double dash' parameters are defined in the command line --one=11 --three=33 --four=44

and the corresponding gms file contains

```
display '%one%';
display '%two%';
$setDDList three
display '%three%';
display '%four%';
```

\$setDDList three checks if all 'double dash' parameters have been used so far except **three**. An error is triggered because **four** has not been used so far and the log file contains

```
*** 1 double dash variables not referenced
    --four=44
```

setEnv

Defines an operating system environment variable. The syntax is **\$setEnv varname value** where **varname** is a user chosen environment variable name and **value** can contain text or a number. Environment variables are destroyed with **\$dropenv varname**. Consider the following example,

```
$ondollar
$set env this is very silly
$log %env%
$setenv verysilly %env%
$log %sysenv.verysilly%
$if NOT "%env%" == "%sysenv.verysilly%" $error setenv did not work
```

\$dropenv verysilly
\$if setenv verysilly \$error should not be true

The following output is echoed to the log file,

```
--- Starting compilation
this is very silly
this is very silly
```

setGlobal

Establishes or redefines contents of a control variable that is accessible in the code where the command appears and all code included therein. The syntax is **\$setglobal varname expression** where **varname** is any user chosen variable name and **expression** is optional and can contain text or a number. These variables are destroyed using **\$dropglobal varname**.

In contrast to **\$evalGlobal** the **\$setGlobal** command does not evaluate the expression at compile time. For differences to **\$set** and **\$setLocal** check the example in the desription of **\$set**.

Note that GAMS allows one to define scoped, local and global variables with the same name but treats them as different under some cases and prioritizes them when using **\$ife** or **\$if**.

Consider the following example,

```
$setglobal notunique aa
$setlocal notunique bb
$set notunique cc
$if "%notunique%" == "aa" display "it is aa";
$if "%notunique%" == "bb" display "it is bb";
$if "%notunique%" == "cc" display "it is cc";
The resulting listing file contains,
```

---- 5 it is bb

setLocal

Establishes or redefines contents of a control variable that is accessible only in the code module where defined. The syntax is **\$setLocal varname expression** where **varname** is any user chosen variable name and **expression** is optional and can contain text or a number. These variables are destroyed using **\$droplocal varname**.

In contrast to **\$evalLocal** the **\$setLocal** command does not evaluate the expression at compile time. For differences to **\$set** and **\$setGlobal** check the example in the desription of **\$set**.

Note that GAMS allows one to define scoped, local and global variables with the same name but treats them as different under some cases and prioritizes them when using **\$ife** or **\$if**.

Consider the following example,

```
$setglobal notunique aa
$setlocal notunique bb
$set notunique cc
$if "%notunique%" == "aa" display "it is aa";
$if "%notunique%" == "bb" display "it is bb";
$if "%notunique%" == "cc" display "it is cc";
The resulting listing file contains,
---- 5 it is bb
```

setNames

Establishes or redefines three control variables so they contain the drive subdirectory, filename and extension of a file named with full path. The syntax is setnames FILE filepath filename

fileextension where FILE is any filename, filepath is the name of a control variable that will contain the name of the subdirectory where the file is located, filename is the name of a control variable that will contain the root name of the file and fileextension is the name of a control variable that will contain the extension of the file. Consider the following example,

\$setnames d:\gams\xxx.txt filepath filename fileextension \$setglobal name %filepath%%filename%%fileextension% \$log %name%

FILE is seperated into its three components placing d:\gams into filepath, xxx into filename and .txt into fileextension. The three items can be recombined back into the original filename by using %filepath%%filename%%fileextension% as shown in the example.

\mathbf{shift}

The **\$shift** option is similar to the DOS batch shift operator. It shifts the order of all parameters passed once to the *`left'*. This effectively drops the lowest numbered parameter in the list.

Consider the following example,

```
scalar a, b, c ; a = 1 ;
$batinclude inc.inc a b c
display a, b, c ;
```

where the batch include file inc.inc is as follows,

```
%2 = %1 + 1 ;
$shift
%2 = %1 + 1 ;
he could in a listing file coutsing
```

The resulting listing file contains,

```
1 scalar a, b, c ; a = 1 ;
BATINCLUDE C:\PROGRAM FILES\GAMSIDE\INC.INC
3 b = a + 1 ;
5 c = b + 1 ;
6 display a, b, c ;
```

In the first statement in the include file, %1 is the first argument in the **\$batinclude** call and is interpreted in this case as a. %2 is the second argument in the **\$batinclude** call and is interpreted as b. This leads to the overall assignment being interpreted as b=a+1.

The shift option shifts the arguments to the left. So now, %1 is interpreted as b, and %2 is interpreted as c. This leads to the second assignment being interpreted as c=b+1.

The result of the display statement in the input file is therefore,

 6 PARAMETER	Α	=	1.000
PARAMETER	В	=	2.000
PARAMETER	С	=	3.000

show

Shows current values of the control variables plus a list of the macros. Consider the following example,

Text

1

what

GLOBAL

GLOBAL

\$set it 1 \$setlocal yy \$setglobal gg what \$include includ \$show where indlud.gms is \$set inincs \$setlocal inincsl \$setglobal inincsg \$show The resulting listing file contains, Level SetVal Туре 1 inincsl LOCAL 1 inincs SCOPED 0 yy LOCAL 0 it SCOPED

0 gg

1 inincsg

Level	SetVal	Туре	Text
0	yy it	LOCAL SCOPED	1
0 1	gg inincsg	GLOBAL GLOBAL	what

Note that only the item defined as **\$setglobal** in the included file carries over.

single

The lines following a **\$single** option will be echoed single spaced on the compiler listing. This option is the default, and is only useful as a switch to turn off the **\$double** option.

Consider the following example,

```
set i /1*2/ ;
scalar a /1/ ;
$double
set j /10*15/ ;
scalar b /2/ ;
$single
set k /5*10/ ;
scalar c /3/ ;
```

The resulting listing file looks as follows,

```
1 set i /1*2/;
2 scalar a /1/;
4 set j /10*15/;
5 scalar b /2/;
7 set k /5*10/;
8 scalar c /3/;
```

Note that lines between the **\$double** and **\$single** options are listed double spaced, while the lines after the **\$single** option revert back to being listed singly spaced.

stars (****)

This option is used to redefine the '****' marker in the GAMS listing file. By default, important lines like those denote errors, and the solver/model status are prefixed with '****'.

Consider the following example,

\$stars #### garbage The resulting listing file looks as follows, 2 garbage #### \$140 ##### \$36,299 UNEXPECTED END OF FILE (1) Error Messages 36 '=' or '..' or ':=' or '\$=' operator expected rest of statement ignored 140 Unknown symbol 299 Unexpected end of file

stitle

This option sets the subtitle in the page header of the listing file to 'text' which follows immediately the keyword stitle. The next output line will appear on a new page in the listing file.

Consider the following example,

\$stitle data tables for input/output

stop

Stops program compilation without creating an error. But there is a difference to **\$exit**. If you have only one input file **\$stop** and **\$exit** will do the same thing. If you are in an include file, **\$exit** acts like an end-of file on the include file. However, if you encounter a **\$stop** in an include file, GAMS will stop reading all input.

sysInclude

Equivalent to **\$batInclude**:

 $\mathbf{239}$

\$sysinclude file arg1 arg2 ...

However, if an incomplete path is given, the file name is completed using the system include directory. By default, the system include directory is set to the GAMS system directory. The default directory can be reset with the **sdir** command line parameter.

Consider the following example,

\$sysinclude abc x y

This call first looks for the include file [GAMS System Directory]/abc, and if this file does not exist, looks for [GAMS System Directory]/abc.gms. The arguments x and y are passed on to the include file to interpret as explained for the \$batinclude option.

Consider the following example,

\$sysinclude c:\abc\myinc.inc x y

This call first looks specifically for the include file c:\abc\myfile.inc.

terminate

\$terminate terminates compilation and execution immediately.

title

This option sets the title in the page header of the listing file to 'text' which follows immediately the keyword title. The next output line will appear on a new page in the listing file.

Consider the following example, \$title Production Planning Model
\$stitle Set Definitions

unload

This dollar command unloads specified items to a GDX file. It is employed using the syntax **\$unload** item1 item2 ... but must be used in conjunction with the command **\$gdxOut**.

\$unload must be proceeded and succeeded by a \$gdxOut. The proceeding \$gdxOut specifies the GDX file name and opens the file. The succeeding \$gdxOut closes the file. More than one \$unload can appear in between. Consider the following slice of code,

Sets

```
i
           canning plants
                           / seattle, san-diego /
       j
          markets
                            / new-york, chicago, topeka / ;
 Parameters
      a(i) capacity of plant i in cases
         1
              seattle
                          350
              san-diego
                          600 /
      b(j) demand at market j in cases
              new-york
                          325
                          300
              chicago
              topeka
                          275 / ;
 Table d(i,j) distance in thousands of miles
                    new-york
                                   chicago
                                                topeka
     seattle
                       2.5
                                    1.7
                                                  1.8
     san-diego
                       2.5
                                     1.8
                                                  1.4 ;
$gdxout tran
$unload i j
$unload b=dem a=sup
$unload d
$gdxout
```

This will create a file tran.gdx containing i, j, d and parameters a and b which are now declared as dem and sup.

use205

This option sets the GAMS syntax to that of Release 2.05. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow one to run such models.

Consider the following example, \$use205 set if /1.2.3/; scalar x ;

The word **if** is a keyword in GAMS introduced with the first version of Release 2.25. The setting of the **\$use205** option allows if to be used as an identifier since it was not a keyword in Release 2.05.

use225

This option sets the GAMS syntax to that of first version of Release 2.25. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since the first version of Release 2.25. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow one to run such models.

Consider the following example,

\$use225

set for /1.2.3/; scalar \boldsymbol{x} ;

The word for is a keyword in GAMS introduced with the later versions of Release 2.25. The setting of the **\$use225** option allows for to be used as an identifier since it was not a keyword in the first version of Release 2.25.

use999

This option sets the GAMS syntax to that of the latest version of the compiler. This option is the default.

Consider the following example, \$use225 set for /1.2.3/; scalar x ; \$use999 for (x=1 to 3, display x) ;

The word for is used as a set identifier by setting the option suse225, and later the keyword for is used as a looping construct by setting the language syntax to that of the latest version by setting suse2999.

version

\$version nnn issues a compilation error if nnn is greater than the current GAMS version.

warning

\$warning issues a compilation warning but continues compilation and execution.
\mathbf{E}

The Option Statement

E.1 Introduction

The option statement is used to set various global system parameters that control output detail, solution process and the layout of displays. They are processed at execution time unlike the dollar control options discussed in Appendix D. They are provided to give flexibility to the user who would like to change the way GAMS would normally do things. GAMS does provide default values that are adequate for the most purposes, but there are always cases when the user would like to maintain control of aspects of the run.

E.1.1 The Syntax

The general form of an option statement is

option 'keyword1' [= 'valuel'] { ,|EOL 'keyword2' [= 'value2'] } ;

where the 'keyword1' and 'keyword2' are recognized option names (but not reserved words) and the 'value1' and 'value2' are valid values for each of the respective options. Note that commas or end-of-line characters are both legal separators between options.

© Option names are not reserved words and therefore do not conflict with other uses of their name.

There are five possible formats:

- 1. a display specifier.
- 2. a recognized name, number following = sign or value
- 3. a recognized name, number following an = sign, then an unsigned integer value.
- 4. a recognized name, number following an = sign, then an unsigned real number.
- 5. a recognized name, number following an = sign, then either of two recognized words.
- An option statement is executed by GAMS in sequence with other instructions. Therefore, if an option statement comes between two solve statements, the new values are assigned between the solves and thus apply only to the second one.
- The values associated with an option can be changed as often as necessary, with the new value replacing the older one each time.

An example of a list of option statements is shown below,

```
option profit:0:3:2;
option eject
        iterlim = 100 , solprint = off ;
solve mymodel using lp maximizing profit ;
display profit.1 ;
input("val1") = 5.3 ;
option iterlim = 50 ;
solve mymodel using lp maximizing profit ;
```

The option statement in the second line affects the display format of the identifier profit. More details on this option can be found under the heading <identifier> in the following section. The option on the second line has no value associated with it, and serves to advance the output in the listing file to the next page. The third line contains two options - iterlim, and solprint. The values associated with the two options on the fourth line are of different types - iterlim has an integer value while solprint requires a character string as a value. Note also that the end of line and the comma serve as legal separators between two options.

The option iterlim serves to limit the number of iterations taken by the solver while attempting to solve the lp model mymodel. After mymodel is solved for the first time, some of the input data is changed and the model is solved again. However, before the second solve statement, the option iterlim is changed to 50. The effect of the sequence above is to limit the first solve to less than 100 iterations and the second to less than 50.

E.2 List of Options

The options available through the option statement are grouped into the following functional categories affecting

output detail	solver specific parameters
nput program control	choice of solver

Table E.1 briefly describes the various options in each of the categories. Section E.3 contains a reference list of all options available through the option statement in alphabetical order with detailed description for each.

Options controlling output detail:				
<identifier> decimals eject limcol limrow</identifier>	controls print format global control of print format advances output to next page number of columns listed number of rows listed	profile profiletol solprint solslack sysout	lists program execution profile sets tolerance for execution profile controls printing of solution controls type of equation information controls printing of solver status file	
Options contr	olling solver specific parameters	5:		
bratio domlim iterlim	use of advanced basis limits number of domain errors limits number of solver iterations	optca optcr reslim	sets absolute optimality tolerance sets relative optimality tolerance limits amount of solver time	
Options controlling choice of solver:				
cns dnlp lp mcp minlp	sets solver for cns model type sets solver for dnlp model type sets solver for lp model type sets solver for mcp model type sets solver for minlp model type	mip mpec nlp rminlp rmip	sets solver for mip model type sets solver for mpec model type sets solver for nlp model type sets solver for rminlp model type sets solver for rmip model type	
Options affect	ing input program control:			
seed	resets seed for pseudo random num controls return of solution values to	ber generator GAMS		

Table E.1: GAMS options

E.3 Detailed Description of Options

This section describes each of the options in detail. The options are listed in alphabetical order for easy reference. In each of the following options, the default value, if available, is bracketed.

<identifier>

Display specifier: identifier:d, identifier:d:r:c Defines print formats for identifier when used in a display statement. d is the number of decimal places, r is the number of index positions printed as row labels, c is the number of index positions printed as column labels; the remaining index positions (if any) will be used to index the planes (index order: plane, row, column); if r is zero list format will be used. The default setting is described in Section 14.4.

bratio (0.25)

Certain solution procedures can restart from an advanced basis that is constructed automatically. This option is used to specify whether or not basis information (probably from an earlier solve) is used. The use of this basis is rejected if the number of basic variables is smaller than bratio times the size of the basis. Setting bratio to 1 will cause all existing basis information to be discarded, which is sometimes needed with nonlinear problems. A bratio of 0 accepts any basis, and a bratio of 1 always rejects the basis. Setting bratio to 0 forces GAMS to construct a basis using whatever information is available. If bratio has been set to 0 and there was no previous solve, an 'all slack' (sometimes called 'all logical') basis will be provided. This option is not useful for MIP solvers.

Range:

0-1

cns (default)

The default cns solver is set during installation. The user can change this default by setting this option to the required solver. The list of cns solvers available with your system can be obtained by reading the gamscomp.txt file that is present in the GAMS system directory. A value of default will change the cns solver back to the default one as specified in gamscomp.txt.

decimals (3)

Number of decimals printed for symbols not having a specific print format attached.

Range:

0-8

dnlp (default)

This option controls the solver used to solve dnlp models. For details cf. option cns.

domlim (0)

This controls the number of domain errors (undefined operations like division by zero) a nonlinear solver will perform, while calculating function and derivative values, before it terminates the run. Nonlinear solvers have difficulty recovering after attempting an undefined operation.

eject

Advances output in the listing file to the next page.

iterlim (1000)

This option will cause the solver to interrupt the solution process after iterlim iterations and return the current solution values to GAMS.

limcol (3)

This controls the number of columns that are listed for each variable in the COLUMN LISTING section of the listing file. Specify zero to suppress the COLUMN LISTING altogether.

limrow (3)

This controls the number of rows that are listed for each equation in the EQUATION LISTING section of the listing file. Specify zero to suppress the EQUATION LISTING altogether.

lp (default)

This option controls the solver used to solve 1p models. For details cf. option cns.

mcp (default)

This option controls the solver used to solve mcp models. For details cf. option cns.

minlp (default)

This option controls the solver used to solve minlp models. For details cf. option cns.

mip (default)

This option controls the solver used to solve mip models. For details cf. option cns.

miqcp (default)

This option controls the solver used to solve miqcp models. For details cf. option cns.

nlp (default)

This option controls the solver used to solve nlp models. For details cf. option cns.

optca (0.0)

This option is only used with problems containing discrete variables (i.e. the GAMS model type mip). General mixed integer problems are often extremely difficult to solve, and proving that a solution found is the best possible can use enormous amounts of resources. This option sets an absolute termination tolerance, which means that the solver will stop and report on the first solution found whose objective value is within optca of the best possible solution.

optcr (0.1)

This option sets a *relative termination tolerance* for problems containing discrete variables, which means that the solver will stop and report on the first solution found whose objective value is within 100*optcr of the best possible solution.

profile (0)

This option is used to generate more information on program execution profiles. This option is equivalent in function to the profile command line parameter.

- 0 No execution profile is generated in listing file
- 1 The listing file reports execution times for each statement and the number of set elements over which the particular statement is executed.
- 2 Specific times for statements inside control structures like loops.

profiletol (0.0)

This option sets profile tolerance in seconds. All statements that take less time to execute than this tolerance are not reported in the listing file.

qcp (default)

This option controls the solver used to solve qcp models. For details cf. option cns.

reslim (1000)

This option causes the solver to terminate the solution process after reslim units of processor time have been used, and the current solution values are returned to GAMS. The units are seconds on the 'wall clock' for PCs, or CPU seconds for larger machines. The SOLUTION SUMMARY part of the listing file shows the limit was used.

rmip (default)

This option controls the solver used to solve rmip models. For details cf. option cns.

rminlp (default)

This option controls the solver used to solve rminlp models. For details cf. option cns.

seed (3141)

This option resets the seed for the pseudo random number generator.

solprint (on)

This option controls the printing of the model solution in the listing file. Using this specification suppresses the list of the solution following a solve.

- on The solution is printed one line per row and column in the listing file.
- off Solution details are not printed. Although this saves paper, we do not recommend it unless you understand your model very well and solve it often.

solslack (0)

This option causes the equation output in the listing file to contain slack variable values instead of level values.

- 0 Equation output in listing file contains level values between lower and upper bound values
- 1 Equation output in listing file contains slack values between lower and upper bound values

solveopt (merge)

Controls the way solution values from a solve are returned to GAMS.

- replace All equations appearing in the model list will be completely replaced by the new model results. Variables are only replaced if they appear in the final model
- merge The new model results are merged into the existing structures.
- clear Similar to the replace option; in addition, variables appearing in the symbolic equations but squeezed out in the final model, are removed.

sysout (off)

This option controls the printing of the solver status file as part of the listing file. The contents of the solver status file are useful if you are interested in the behavior of the solver. If the solver crashes or encounters any difficulty, the contents of the solver status file will be automatically sent to the listing file.

on Prints the system output file of the solver

off No subsystem output appears on output file unless a subsystem error has occurred.

 \mathbf{F}

The Save and Restart Feature

F.1 Introduction

GAMS saves the information provided in the input files in intermediate, mostly binary, files. These files are referred to as *work files* or *scratch files*. Some of these files are used to exchange information between GAMS and the various solvers. Just before a GAMS run is complete, these files are usually deleted.

Input files can be processed in parts through the use of these intermediate files. This is an extremely powerful feature that can reduce the time needed when several runs of the same model are being made with different data.

It may be clearer if the process is described in a different way. Imagine taking a large GAMS program and running it, producing one output file. Then think of splitting the program into two pieces. The first piece is run and the resulting work file is saved along with the resulting listing file. Then the second piece is run after reading in the data from the work file saved previously. A new listing file is generated for the second piece. The content of the output that results is the same, though slightly rearranged, as the case when the large file was run. Splitting the files allows one to interrupt a GAMS task and restart it later without loss of information. Furthermore, changes could be made or errors corrected to the later parts.

F.2 The Save and Restart Feature

Using the save and restart command line parameters provides a mechanism to break up the compilation of a large input file into many components and stages. Some of the reasons for using these features and running a model in pieces are explained in Section F.3. The next two sub-sections explain the save and restart mechanisms in GAMS. The save and restart command line parameters, described in Appendix C, are used for this purpose.

[TRNSPORT] is used for the purposes of illustration. Consider the following file, containing code extracted from this model called file1.gms,

```
Sets
  i
       "canning plants"
                           / seattle. san-diego /
       "markets"
                           / new-york, chicago, topeka / ;
   j
Parameters
   a(i)
         "capacity of plant i in cases"
           seattle
                       350
      /
           san-diego
                       600
                             1
   b(j) "demand at market j in cases"
           new-york
                        325
                        300
           chicago
           topeka
                       275
                            /;
              "distance in 1000 miles"
Table d(i,j)
                  new-york
                                  chicago
                                                topeka
   seattle
                    2.5
                                   1.7
                                                 1.8
```

```
2.5
                                               1.4 ;
   san-diego
                                  1.8
Scalar f "freight in dollars/case per 1000 miles" /90/ ;
Parameter c(i,j) "transport cost in $1000/case" ;
               c(i,j) = f * d(i,j) / 1000 ;
Variables
            "shipment quantities in cases"
   x(i,i)
    z
            "total transportation costs in 1000$";
Positive Variable x ;
Equations
    cost
                "define objective function"
    supply(i)
                "observe supply limit at plant i"
    demand(j)
                "satisfy demand at market j" ;
               z =e= sum((i,j), c(i,j)*x(i,j));
cost ..
supply(i)
               sum(j, x(i,j)) =l= a(i) ;
         ..
demand(j) ..
              sum(i, x(i,j)) =g= b(j);
Model transport /all/ ;
```

Consider the following file (say file2.gms),

Solve transport using lp minimizing z ; Display x.l, x.m ;

Note that [TRNSPORT] results from appending file2.gms at the end of file1.gms.

F.2.1 Saving The Work File

The information in file1.gms can be stored by using the following call to GAMS,

```
gams file1 s=trans
```

One work file called trans.g00 is created in the working directory.

- The Work file preserves all information (including declarations, values, option settings and compiler dollar directives) known to GAMS at the end of the run that created them.
- The work file is not machine specific, it is portable between platforms. For example, a work file generated on a PC running Windows can be re-used on a Sun machine running Solaris.

F.2.2 Restarting from the Work File

Consider the following call,

gams file2 r=trans

GAMS reads the work file named trans.g00 and regenerates the information stored in file1.gms. Then file2.gms is run and the result is as if the two files were concatenated.

A restarted run also requires a continuation input file. The restart does not alter work files. They can be used repeatedly to continue a particular run many times, possibly with many different continuation input files.

The most common mistake that occurs in using the save and restart feature is running GAMS on the same file twice, so all the data and equation definitions get repeated which causes compilation errors during restart. The following calls will cause errors: In general, definitions of data constructs should not be repeated either in the same file or across files used in the Save and Restart operation. GAMS works as if the two files are actually concatenated. In order to avoid any syntax problems, one needs to understand the GAMS syntax regarding data entry. By default GAMS requires that each data item be entered only once. Once the elements that form the set have been defined, the set cannot be redefined through the data statement. For example, the following set of statements are all invalid:

```
set i /seattle, san-diego / ;
set i /seattle, san-diego, portland / ;
```

Similar rules apply to Scalar, Parameter, and Table declarations. One can only use assignment statuents to change values of scalars, parameters and tables once they have been specified by the data statement. For example,

```
parameter a(i) /
seattle 20
san-diego 50 / ;
a("seattle") = 10 ;
a("san-diego") = 100 ;
```

One can, however, separate the definition of the data type from the actual data entry. For example, the following succession of statements is valid:

Set i ;
Set i /seattle, san-diego / ;

This is true with the other data types as well. This last feature is very useful in completely separating the model definition from the data, and leads to the development of a good runtime GAMS model.

- It is the responsibility of the modeler to ensure that the contents of the input file matches that of the work file, although the compiler will issue errors if it detects any inconsistencies, such as references to symbols not previously declared.
- \blacksquare A Work file can be used only by GAMS tasks requesting a restarted run.
- A Work file can be saved following a restarted run, thus producing another work file that reflects the state of the job following completion of the statements in the continuation file.

F.3 Ways in which a Work File is Useful

The basic function of a work file is to preserve information that has been expensive to produce. Several reasons for wanting to do this are described in this section.

F.3.1 Separation of Model and Data

The separation of model and data is one of the core principles of the GAMS modeling paradigm. The use of save and restart features helps to exploit this separation.

Let us re-arrange the contents of file1.gms and file2.gms to separate the model from the data. The modified version of file1.gms is shown below,

```
Sets i canning plants
j markets
Parameters a(i) "capacity of plant i in cases"
```

```
"demand at market j in cases"
            b(j)
            c(i,i)
                    "transport cost in 1000$/case"
            d(i,j)
                   "distance in 1000 miles"
Scalar f "freight in $/case per 1000 miles"
Variables
           x(i,j)
                   "shipment quantities in cases"
                    "total transportation costs in 1000$"
            z
Positive Variable x ;
                       "define objective function"
Equations cost
                       "observe supply limit at plant i"
           supply(i)
           demand(j)
                       "satisfy demand at market j" ;
               z =e= sum((i,j), c(i,j)*x(i,j));
cost ..
supply(i) ..
               sum(j, x(i,j)) =l= a(i) ;
demand(j) ..
              sum(i, x(i,j)) =g= b(j);
Model transport /all/ ;
```

Note that this representation does not contain any data, and is a purely algebraic representation of the transportation problem. Running this model and saving the resulting work file will allow the model to be used with the data stored in a separate file (file2.gms).

```
Sets
       i
           / seattle, san-diego /
           / new-york, chicago, topeka / ;
       i
                  / seattle
Parameters
            a(i)
                                 350
                                 600
                     san-diego
                                      /
            b(j)
                  / new-york
                                 325
                     chicago
                                 300
                                 275
                     topeka
                                      1
Table d(i,j)
                   new-york
                                   chicago
                                                topeka
    seattle
                      2.5
                                    1.7
                                                  1.8
    san-diego
                      2.5
                                    1.8
                                                  1.4
                                                       :
Scalar f / 90 / ;
c(i,j) = f * d(i,j) / 1000;
Solve transport using lp minimizing z ;
Display x.l, x.m ;
```

This file contains the data for the model and the solve statement.

F.3.2 Incremental Program Development

GAMS programs are often developed in stages. A typically style is to put the sets first, followed by tables and data manipulations, then equations, and finally the assignments used to generate reports. As each piece of the model is built, it should be run and checked for errors by inserting diagnostic display and abort statements. As confidence mounts in the correctness of what has been done, it is useful to save the completed parts in a work file. Then by restarting it is possible to work only on the piece under active development, thereby minimizing computer costs and the amount of output produced in each of the developmental runs. This approach is especially useful when entering the statements needed for reporting. The solution is much more expensive than the report, but the report is likely to involve many details of content and layout that have to be tried several times before they are satisfactory. The model can be solved and the result saved in a work file. One can then restart from the work file when developing the report. It is a great relief not to have to solve the model every time.

F.3.3 Tacking Sequences of Difficult Solves

In many cases where solves are known to be difficult and expensive, it may be too risky to ask GAMS to process a job containing many solve statements. The risk is that if one solve does not proceed to normal completion, then the following one will be started from a bad initial point, and much time and effort will be wasted.

An alternative is to request one solve at a time and save the work file. Then the output is carefully inspected before proceeding. If everything is normal, the job is restarted with the next solve requested. If not, the previous solve can be repeated, probably with a different initial point, or continued if the cause of the trouble was an iteration limit, for example.

This approach is common when doing repeated solves of a model that successively represent several consecutive time periods. It uses a work file in a sequential rather than a tree-structure way.

It also produces many files, which can be difficult to manage, if the solves are especially difficult, it is possible to lose track of exactly what was done. Great care is needed to avoid losing control of this process.

F.3.4 Multiple Scenarios

The majority of modeling exercises involves a *base case*, and the point of the study is to see how the system changes when circumstances change, either naturally or by design. This is often done by making many different changes to the base case and separately considering the effects; it is sometimes called *'what if'* analysis.

The point is that the base can be saved using a work file, and as many different scenarios as may be interesting can then be run separately by restarting. Each scenario probably involves only making a change in data or in bounds, solving the changed model (the base solution is automatically used as a starting point), and reporting. This procedure is an example of how a work file is used in a tree-structured way: one work file is used with many different (but probably very small) input files to produce many different output files. File handling is less likely to be a problem than in the sequential case above.

F.3.5 The GAMS Runtime License

We assume the model and the data have been completely separated as shown above, with file1.gms containing only the model, and file2.gms containing only the data.

The developer of the model can run the first model with the following command:

gams file1 s=trans

and then distribute the file trans.g00 file that result, along with the example file2.gms.

If the end-user has a run-time license for GAMS, they will not be able to see the model, nor change it by adding any new variables or equations. The end-user will only be able to change the data, and run the model developed during the save process. However, the end-user will have full control of the data, and will be able to manipulate the number of elements in the set, and the values of the various scalars, parameters, and tables.

The end user will run the model with the following command:

gams file2 r=trans

G

Secure Work Files

G.1 Introduction

When models are distributed to users other than the original developers or embedded in applications to be deployed by other developers, issues of privacy, security, data integrity and ownership arise. We may have to hide, protect or purge some parts of the model before it can be released. The information to be protected can be of numeric or symbolic nature. For example:

Privacy

A Social Accounting Matrix supplied by a Statistical Office is required in a general equilibrium model to be used by the Ministry of Finance. The data from the statistical office needs to be protected for obvious privacy reasons and the model experiments are used to evaluate policy options that are highly confidential. Most of the model structure is public, most of the data however is private and model results need to be transformed in such a way as to prohibit the discovery of the original data.

Security

Components of a model contain proprietary information that describes mathematically a chemical reaction. The associated algebra and some of the data are considered of strategic importance and need to be hidden completely. The final model however, will be used at different locations around the world.

Integrity

Data integrity safeguards are needed to assure the proper functioning of a model. Certain data and symbolic information needs to be protected from accidental changes that would compromise the operation of the model.

To address these issues, access control at a symbol level and secure restart files have been added to the GAMS system.

Access Control

The access to GAMS symbols like sets, variables, parameters and equations can be changed once with the compile time commands \$purge, \$hide, \$protect and \$expose. \$Purge will remove any information associated with this symbol. \$Hide will make the symbol and all its information invisible. \$Protect prevents changes to information. \$Expose will revert the symbol to its original state.

Secure Restart Files

The GAMS licensing mechanism can be used to save a secure model in a secure work file. A secure work file¹ behaves like any other work file but is locked to a specific users license file. A privacy license, the license file of the target users, is required to create a secure work file. The content of a secure work file is disguised and protected against unauthorized access via the GAMS license mechanism.

A special license is required to set the access controls and to create a corresponding secure work file. Reporting features have been added to allow audits and traces during generation and use of secure work files.

 $^{^{1}}$ Work files are used to save and restart the state of a GAMS program. Depending on the context, we refer to those files as work files, save files or restart files.

G.2 A First Example

The model TRNSPORT from the GAMS model library will be used to illustrate the creation and deployment of a secure work file. Assume we want to distribute this model but have concerns about proprietary formulations and data. In addition we would like to protect the user from making unintentional modifications to the model. We assume that the objective function and the supply constraints are to be hidden from other users and only the demand figures can be changed. Data that is not needed any more will be purged as well. This will be demonstrated below using the command line interface to GAMS². First we will copy the model from the GAMS model library, run the model and save a normal work file:

```
> gamslib trnsport
```

```
> gams trnsport s=t1
```

We continue to enter access control commands in a file called t2.gms and create a secure work file with the name t2.g00:

```
> type t2.gms
      $eolcom //
      $protect all
                               // make all symbol read only
      $purge
              d f
                               // remove items d and f
      $hide
                               // make objective invisible
               cost supply a
      $expose transport b
                               // allow changes to b
> gams t2 r=t1 s=t2 plicense=target
      GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
      Licensee: Source User Name
                Source Company Name
      *** Creating a Secure Restart File for:
      ***
                Target User Name
      ***
                Target Company Name
      --- Starting continued compilation
      --- T2.GMS(6) 1 Mb
      --- Starting execution
      *** Status: Normal completion
```

The access control commands are activated by the use of the privacy GAMS license option PLICENSE. This option specifies the name of the target user license file. The save/restart file t2.g00 can only be read with the target license file.

The three lines starting with '***' are a recap of the content of the target license file. From now on, the source and the target licensees are 'burned into' this file and all its descendants. We are ready to send the restart file to the target user or system.

The target user can now run the model with new data, add new GAMS statements, and make new save/restart files. The only restrictions are that some of the symbols are hidden and that this model can only be executed using the target license file. For example, the target user may want to half the demand and compare the original solution with the new one. We will call this program t3.gms and it will be executed on the target system:

```
> type t3.gms
parameter rep summary report;
rep(i,j,'base') = x.l(i,j);
b(j) = b(j)*0.5; solve transport minimizing z using lp;
rep(i,j,'half') = x.l(i,j);
display rep;
> gams t3 r=t2
```

²When using the GAMS/IDE interface, the GAMS parameters are entered and mainened in the text window just right to the *Run GAMS (F9)* button.

```
GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
Licensee: Target User Name
Target User Company
*** Restarting from a Secure Restart File created by:
*** Source User Name
*** Source Company Name
--- Starting continued compilation
--- T3.GMS(5) 1 Mb
...
```

Note that the originator/owner of the secure work file is mentioned by name on the log file. A similar message is contained in the listing file:

```
> type t3.lst
...
EXECUTION TIME = 0.000 SECONDS 1.1 Mb WIN201-124
**** Secure Save/Restart File Source:
    Source User Name
    Source Company Name
**** Secure Save/Restart File Target:
    Target User Name
    Target User Company
.
```

A more detailed inspection of the listing file will show that the hidden variables and equations do not appear in the usual equation/variable listings and the solution print. The hidden items can only be accessed via a public (exposed) model and a solve statement.

In the following two sections we will describe secure work files and the access control commands in more detail.

G.3 Secure Work Files

Secure Work Files control access to symbolic and numeric information and can only be read by a specific GAMS user. The initial creation or additions to access control requires a special GAMS license. Saving Secure Work Files without new access controls does not require a special GAMS license. The creation or addition of access control is signaled by the use of the GAMS parameter PLICENSE, which gives the name of a privacy license file. The shortcut 'PLICENSE=LICENSE' sets the privacy license to the current license file. This is convenient when experimenting with access controls.

When a secure work file is written the first time, the first and second lines of the current license file and the privacy license file are inserted into the work file. This information cannot be changed any more and the original source and the intended target users are locked into the work file.

A secure work file can be used just like any other work file and new work files can be derived from secure files. However, their use is restricted to the 'target' user specified with the PLICENSE parameter. The target user can, if licensed, add access controls to an existing secure file by using the PLICENSE=LICENSE parameter but cannot change the original information about source and target users.

Secure work files can be tested on any GAMS system by specifying a non-default license file with the LI-CENSE=target parameter.

G.4 Access Control Commands

There are four Access Control Commands (ACC) that are processed during the compilation phase. These commands can be inserted anywhere and are processed in chronological order and have the following syntax:

```
$acc ident1 ident2 ...
$acc ALL
```

Where *acc* is one of the four ACC's:

PURGEremove the objects and all data associatedHIDEhide the objects but allow them to be used in model calculationsPROTECTthe objects cannot be modified but used in model calculationsEXPOSEremoves all restrictions

The keyword ALL applies the ACC to all identifiers defined up to this point in the GAMS source code. ACC's can be changed and redefined within the same GAMS program. Identifiers inherited from a restart file cannot be changed, however.

G.5 Advanced Use of Access Control

We will again use the transport model to show how to hide input data and results from the target user. The target user is only allowed to view percentage changes from an unknown base case. In addition to the original model we will introduce a data initialization and a report model.

First we will define a new model to calculate input data. The previous parameter c is now the variable newc and the model getc does the calculations:

```
$include trnsport.gms
variable newc(i,j) new tansport data;
equation defnewc(i,j) definition of new transport data;
model getc compute new transport data / defnewc /;
defnewc(i,j).. newc(i,j) === f*d(i,j)/1000;
solve getc using cns;
```

Next, we change the objective function of the original model to a more complicated nonlinear function. Furthermore, we will compute a base case value to be used later in the reporting model. Note the reference to newc.l(i,j), since nexc is a variable we have to specify that we only want the level value:

```
scalar beta scale coefficient / 1.1 /;
equation newcost definition of new objective function;
model newtrans / newcost,supply,demand /;
newcost.. z =e= sum((i,j), newc.l(i,j)*x(i,j)**beta);
solve newtrans using nlp minimizing z;
parameter basex(i,j) base values of x;
basex(i,j) = x.l(i,j);
```

Finally we transform the result by using a third model:

```
variable delta(i,j) percentage change from base values;
equation defdelta(i,j) definition of delta;
model rep / defdelta /;
defdelta(i,j)$basex(i,j)..
    delta(i,j) == 100*(x.l(i,j)- basex(i,j))/basex(i,j);
solve rep using cns;
```

We will save the above GAMS code under the name pl.gms, execute and make a save/restart file with the name pl.g00 as follows:

> gams p1 s=p1

Now we are ready to make some test runs similar to those we expect to be defined by the target user. We will define three scenarios to be solved in a loop and name the file ul.gms:

```
parameter report summary report;
loop(s,
    beta = sbeta(s);
    f = sf(s);
    solve getc using cns;
    solve newtrans using nlp minmizing z;
    solve rep using cns;
    report(i,j,s) = delta.l(i,j);
    report(',','beta',s) = beta;
    report('','f',s) = f;
    report(''obj','z',s) = z.l );
display report;
```

When executing the above GAMS code together with the original transport model from the GAMS model library we will get the following results.

```
> gams u1 r=p1
   ____
           109 PARAMETER report summary report
                               one
                                            two
                                                      three
   seattle
            .new-york
                            -4.050
                                         -6.967
                                                      -8.083
   seattle
            .chicago
                           -18.797
                                        -27.202
                                                    -31.550
   seattle .topeka
                           233.958
                                        348.468
                                                    404.187
   san-diego.new-york
                             3.605
                                          6.201
                                                      7.194
   san-diego.chicago
                            28.138
                                        40.719
                                                     47.228
                                        -23.104
   san-diego.topeka
                           -15.512
                                                     -26.799
            .beta
                             1.250
                                         1.500
                                                      2.000
            .f
                            85.000
                                         75.000
                                                     50.000
   obj
                           526.912
                                       1652.963
                                                  13988.774
            .z
```

Note that all symbols are still completely exposed. We need to add access controls to the model p1.gms before we can ship it to the target client. The information to be protected is the original distance matrix and derived information. We start out by hiding everything and then give access to selected parts of the model. We collect the access control information in the file s1.gms shown below and save the secure work file under the name s1.g00. Since we are still testing, we use our own license as target user. This will allows us to test the system the same way the target user will use it:

```
$hide all
$expose getc newtrans rep
$expose i j z delta
$expose f beta a b
> gams s1 r=p1 s=s1 plicense=license
```

To test the new secure file, we run again the problem u1.gms. When doing so you will observe that equation, variable and solution listings related to the hidden variables are not shown any more. Any attempt to reference a hidden variable will case a compilation error.

```
> gams u1 r=s1
```

Before we can ship a secure work file we need a copy of the target user license file. We then will restart again from pl.gms, zip the resulting secure files and we are ready to distribute the model:

> gams s1 r=p1 plicense=target.txt s=target > zip target target.g00

G.6 Limitations and Future Requirements

One of the design goals for secure work files has been to minimize the impact on other components of the GAMS system. Solvers used out of a secure environment should work as if called out of a normal environment. This

implies that, in principle, certain information could be recovered, if one has knowledge of GAMS solvers internals and is willing to expand considerable programming effort. In this section we will discuss current limitations and possible extension to the security features.

The following limitations exist:

- ➤ Solvers are not security aware and it would be possible to write a special GAMS solver that extracts information about a specific model instance. Primal and duals values as well as first partial derivatives could be extracted and displayed.
- ➤ The names and explanatory text of all GAMS symbols are retained in the work file and could be accessed by a special GAMS solver.
- ➤ The source and target license files locked to the secure work file cannot be changed. If the target user upgrades the GAMS system and receives a new license file, the secure work file cannot be read any more.

Η

Compressed and Encrypted Input Files

H.1 Introduction

When models are distributed to users other than the original developers, issues of privacy, security, data integrity and ownership arise. Besides using secure work files, one can compress and encrypt GAMS input files. The compression and decompression of files is available to any GAMS user. The encryption follows the work file security model and requires special licensing. Three new Dollar Control Options have been introduced:

\$Encrypt	<source/> <target></target>	Encrypts into a GAMS system file
\$Compress	<source/> <target></target>	Compresses into a GAMS system file
\$Decompress	<source/> <target></target>	Decompresses a GAMS system file

Encryption is only available if a system is licensed for secure work files and usually requires a target license file which will contain the user or target encryption key. Once a file has been encrypted it cannot be decrypted any more.

The use of a PLICENSE parameter will specify the target or privacy license to be used as a user key for encrypting. Decompression and encrypting is done on the fly into memory when reading the GAMS system files. GAMS will recognize if a file is just plain text or compressed and/or encrypted and will validate and process the files accordingly.

Finally, all compressed and encrypted files are, of course, platform independent as any other GAMS input file.

H.2 A First Example

The model TRNSPORT from the GAMS model library will be used to illustrate the creation of a compressed input file. First we will copy the model from the GAMS model and create a compressed version. Spaces are recognized as separators between the source and target file names which means you have to use quotes (single or double) if the filenames contain spaces:

```
> gamslib trnsport
> echo $compress trnsport.gms t1.gms > t2.gms
> gams t2
...
--- Compress Source: C:\support\28Dec\trnsport.gms
--- Compress Target: C:\support\28Dec\t1.gms.gms
--- Compress Time : Omsec
...
```

Now we can treat the compressed input files like any other GAMS input file and the listing files will be identical because the decompressed input is echoed just like any normal input line:

```
> gams transport
> gams t1
```

We can decompress the file and compare it to the original file:

```
> echo $Decompress .t1.gms. .t3.org. > t4.gms
> gams t4
> diff t1.gms t3.org
```

Finally, we may want to encrypt the input file in a way to hide the equation definitions. To do this we just insert \$Offlisting and \$Onlisting around the blocks of GAMS code we want to hide. We now can encrypt the modified model file by using a privacy or target license file to lock the new encrypted file to this license key.

```
> echo $encrypt trnsport.gms t1.gms > t2.gms
> gams t2 plicense=target
```

This new version of t1.gms can only be used with the target license file. To simulate the target environment we can force the target license to be used instead of the installed one.

```
> gams t1 license=target dumpopt=19
```

Note the use of dumopt=19 which is sometimes used for debugging and maintenance, writes a clean copy of the input to the file t1.dmp (all include files and macros are expanded). In the file t1.dmp you will see that the input text between \$offlisting and \$onlisting is suppressed.

An attempt to \$Decompress and decrypt the file will fail as well. Once a file has been encrypted, it cannot be decrypted any more. For example, trying to decompress as in the example before will fail:

> gams t4
...
--- Decompress Source: C:\support\28Dec\t1.gms
--- Decompress Target: C:\support\28Dec\t3.org
--- Decompress Error : Security violation

H.3 The CEFILES Gamslib Model

\$title Compressed Input Files (CEFILES,SEQ=317)

The CEFILES model from the GAMS Model Library contains a more elaborate example that can be easily modified to test the use of compressed files. This example will also show how to use the PLICENSE=LICENSE parameter to test the creation and use without having a target license file available.

```
$ontext
This model demonstrates the use of compressed input files.
Remember, if the file names contain spaces you need
to use single or double quotes around the file names.
$offtext
* --- get model
$ondollar
$call gamslib -q trnsport
```

```
* --- compress and run model
$compress trnsport.gms t1.gms
$decompress t1.gms t1.org
$call diff trnsport.gms t1.org > %system.nullfile%
$if errorlevel 1 $abort files trsnport and t1 are different
```

* --- check to see if we get the same result

```
$call gams trnsport gdx=trnsport lo=%gams.lo%
$if errorlevel 1 $abort model trnsport failed
$call gams t1 gdx=t1 lo=%gams.lo%
$if errorlevel 1 $abort model t1 failed
$call gdxdiff trnsport t1 %system.redirlog%
$if errorlevel 1 $abort results for trnsport and t1 are not equal
* --- also works with include files
$echo $include t1.gms > t2.gms
$call gams t2 gdx=t2 lo=%gams.lo%
$if errorlevel 1 $abort model t2 failed
$call gdxdiff trnsport t2 %system.redirlog%
$if errorlevel 1 $abort results for trnsport and t2 are not equal
$terminate
```

H.4 The ENCRYPT GAMSLIB Model

The ENCRYPT model from the GAMS Model Library contains a more elaborate example of the use of encrypted files. Note the use of LICENSE=DEMO which overrides the currently installed license with a demo license which has the secure file option enabled.

\$Title Input file encryption demo (ENCRYPT,SEQ=318)

```
$ontext
Input files can be encrypted and use the save/privacy license
file mechanism for managing the user password. Similar to
compression, we offer an $encrypt utility to lock any file to a
specific target license file. Once a file has been encrypted it
can only be read by a gams program that has the matching license
file. There is no inverse operation possible: you cannot recover
the original GAMS file from the encrypted version.
```

To create an encrypted file, we need a license file which has the security option enabled. To allow easy testing and demonstration a special temporary demo license can be created internally and will be valid for a limited time only, usually one to two hours.

In the following example we will use the GAMS option license=DEMO to use a demo license with secure option instead of our own license file. Also note that we use the same demo license file to read the locked file by specifying the GAMS parameter plicence=LICENSE. \$offtext

```
* --- get model
$ondollar
$call gamslib -q trnsport
```

* --- encrypt and try to decrypt
\$call rm -f t1.gms
\$echo \$encrypt trnsport.gms t1.gms > s1.gms
\$call gams s1 license=DEMO plicense=LICENSE lo=%gams.lo%
\$if errorlevel 1 \$abort encryption failed

\$eolcom //
\$if NOT errorfree \$abort pending errors
\$decompress t1.gms t1.org // this has to fail
\$if errorfree \$abort decompress did not fail
\$clearerror

* --- execute original and encrypted model \$call gams trnsport gdx=trnsport lo=%gams.lo% \$if errorlevel 1 \$abort model trnsport failed * Although this reads license=DEMO this license file is the one * specified with plicense from the s1 call \$call gams t1 license=DEMO gdx=t1 lo=%gams.lo% \$if errorlevel 1 \$abort model t1 failed \$call gdxdiff trnsport t1 %system.redirlog%

```
$if errorlevel 1 $abort results for trnsport and t1 are not equal
```

```
* --- use the encrypted file as an include file
$onecho > t2.gms
$offlisting
* this is hidden
option limrow=0,limcol=0,solprint=off;
$include t1.gms
$onlisting
* this will show
$offecho
$call gams t2 license=DEMO lo=%gams.lo%
$if errorlevel 1 $abort model t2 failed
* --- protect against viewing
     now we will show how to protect parts of an input
*
*
     file from viewing and extracting original source
     via the gams DUMPOPT parameter. We just need to
*
*
     encrypt again
* --- encrypt new model
$call rm -f t3.gms
$echo $encrypt t2.gms t3.gms > s1.gms
$call gams s1 license=DEMO plicense=LICENSE lo=%gams.lo%
$if errorlevel 1 $abort encryption failed
$call gams t3 license=DEMO gdx=t3 dumpopt=19 lo=%gams.lo%
$if errorlevel 1 $abort model t3 failed
$call gdxdiff trnsport t3 %system.redirlog%
     errorlevel 1 $abort results for trnsport and t3 are not equal
$if
* --- check for hidden output
$call grep "this is hidden" t3.lst > %system.nullfile%
$if not errorlevel 1 $abort did not hide in listing
$call grep "this is hidden" t3.dmp > %system.nullfile%
```

```
$if not errorlevel 1 $abort did not hide in dump file
```

Ι

The GAMS Grid Computing Facility

I.1 Introduction

As systems with multiple CPUs and High Performance Computing Grids are becoming available more widely, the GAMS language has been extended to take advantage of these new environments. New language features facilitate the management of asynchronous submission and collection of model solution tasks in a platform independent fashion. A simple architecture, relying on existing operating system functionality allows for rapid introduction of new environments and provides for an open research architecture.

A typical application uses a coarse grain approach involving hundreds or thousands of model solutions tasks which can be carried out in parallel. For example:

- \succ Scenario Analysis
- \succ Monte Carlo Simulations
- \succ Lagrangian Relaxation
- \succ Decomposition Algorithms
- \succ Advanced Solution Approaches

The grid features work on all GAMS platforms and have been tailored to many different environments, like the Condor Resource Manager, a system for high throughput computing from the University of Wisconsin or the Sun Grid Engine. Researchers using Condor reported a delivery of 5000 CPU hours in 20 hours wall clock time.

Disclaimer. The use of the term grid computing may be offensive to some purists in the computer science world. We use it very loosely to refer to a collection of computing components that allow us to provide high throughput to certain applications. One may also think of it as a resurrection of the commercial service bureau concept of some 30 years ago.

Caution. Although these features have been tested on all platforms and are part of our standard release we may change the approach and introduce alternative mechanisms in the future.

Acknowledgments. Prof. Monique Guignard-Spielberg from the Wharton School at U Penn introduced us to parallel Lagrangian Relaxation on the SUN Grid Environment. Prof. Michael Ferris from the University of Wisconsin at Madison adopted our original GAMS grid approach to the high throughput system Condor and helped to make this approach a practical proposition.

I.2 Basic Concepts

The grid facility separates the solution into several steps which then can be controlled separately. We will first review what happens during the synchronous solution step and then introduce the asynchronous or parallel solution steps.

When GAMS encounters a solve statement during execution it proceeds in three basic steps:

- 1. **Generation.** The symbolic equations of the model are used to instantiate the model using the current state of the GAMS data base. This instance contains all information and services needed by a solution method to attempt a solution. This representation is independent of the solution subsystem and computing platform.
- 2. Solution. The model instance is handed over to a solution subsystem and GAMS will wait until the solver subsystem terminates.
- 3. Update. The detailed solution and statistics are used to update the GAMS data base.

In most cases, the time taken to generate the model and update the data base with the solution will be much smaller than the actual time spent in a specific solution subsystem. The model generation takes a few seconds, whereas the time to obtain an optimal solution may take a few minutes to several hours. If sequential model solutions do not depend on each other, we can solve in parallel and update the data base in random order. All we need is a facility to generate models, submit them for solution and continue. At a convenient point in our GAMS program we will then look for the completed solution and update the data base accordingly. We will term this first phase the submission loop and the subsequent phase the collection loop:

- **Submission Loop.** In this phase we will generate and submit models for solutions that can be solved independently.
- **Collection Loop.** The solutions of the previously submitted models are collected as soon a solution is available. It may be necessary to wait for some solutions to complete by putting the GAMS program to 'sleep'.

Note that we have assumed that there will be no errors in any of those steps. This, of course, will not always be the case and elaborate mechanisms are in place to make the operation fail-safe.

I.3 A First Example

The model QMEANVAR form the GAMS model library will be used to illustrate the use of the basic grid facility. This model traces an efficiency frontier for restructuring an investment portfolio. Each point on the frontier requires the solution of independent quadratic mixed integer models. The original solution loop is shown below:

```
Loop(p(pp),
  ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
  Solve minvar min var using miqcp ;
  xres(i,p) = x.l(i);
  report(p,i,'inc') = xi.l(i);
  report(p,i,'dec') = xd.l(i) );
```

This loop will save the solutions to the model MINVAR for different returns RET. Since the solutions do not depend on the order in which they are carried out, we can rewrite this loop to operate in parallel. The first step is to write the submit loop:

```
parameter h(pp) model handles;
minvar.solvelink=3;
Loop(p(pp),
  ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
  Solve minvar min var using miqcp;
  h(pp) = minvar.handle );
```

The model attribute **.solvelink** controls the behavior of the solve statement. A value of '3' tells GAMS to generate and submit the model for solution and continue without waiting for the completion of the solution step. There is a new model attribute **.handle** which provides a unique identification of the submitted solution request. We need to store those handle values, in this case in the parameter h, to be used later to collect the solutions once completed. This is then done with a collection loop:

```
loop(pp$handlecollect(h(pp)),
    xres(i,pp) = x.l(i);
    report(pp,i,'inc') = xi.l(i);
    report(pp,i,'dec') = xd.l(i) );
```

The function **handlecollect** interrogates the solution process. If the solution process has been completed the results will be retrieved and the function returns a value of 1. If the solution is not ready to be retrieved the value zero will be returned.

The above collection loop has one big flaw. If a solution was not ready it will not be retrieved. We need to call this loop several times until all solutions have been retrieved or we get tired of it and quit. We will use a repeat-until construct and the handle parameter h to control the loop to look only for the not yet loaded solutions as shown below:

Once we have extracted a solution we will set the handle parameter to zero. In addition, we want to remove the instance from the system by calling the function **handledelete** which returns zero if successful (see definition). No harm is done if it fails but we want to be notified via the conditional display statement. Before running the collection loop again, we may want to wait a while to give the system time to complete more solution steps. This is done with the sleep command that sleeps some time. The final wrinkle is to terminate after 100 seconds elapsed time, even if we did not get all solutions. This is important, because if one of the solution steps fails our program would never terminate. The last statement sets the results of the missed solves to NA to signal the failed solve. The parameter h will now contain the handles of the failed solvers for later analysis.

Alternatively, we could have used the function **handlestatus** and collect the solution which is stored in a GDX file. For example we could write:

```
loop(pp$(handlestatus(h(pp))=2),
  minvar.handle = h(pp);
  execute_loadhandle minvar;
  xres(i,pp) = x.l(i);
  report(pp,i,'inc') = xi.l(i);
  report(pp,i,'dec') = xd.l(i) );
```

The function **handlestatus** interrogates the solution process and returns '2' if the solution process has been completed and the results can be retrieved. The solution is stored in a GDX file which can be loaded in a way similar to other gdx solution points. First we need to tell GAMS what solution to retrieve by setting the minvar.handle to the appropriate value. Then we can use execute_loadhandle to load the solution for model minvar back into the GAMS data base. Using **handlestatus** and **loadhandle** instead of the simpler **handlecollect** adds one more layer of control to the final collection loop. We now need one additional if statement inside the above collection loop:

```
Repeat
loop(pp$h(pp),
    if(handlestatus(h(pp))=2,
        minvar.handle = h(pp);
        execute_loadhandle minvar;
        xres(i,pp) = x.l(i);
        report(pp,i,'inc') = xi.l(i);
        report(pp,i,'dec') = xd.l(i);
        display$handledelete(h(pp)) 'trouble deleting handles' ;
        h(pp) = 0 ) );
```

```
display$sleep(card(h)*0.2) 'sleep some time';
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;
```

Now we are ready to run the modified model. The execution log will contain some new information that may be useful on more advanced applications:

```
--- LOOPS pp = p1
--- 46 rows 37 columns 119 non-zeroes
--- 311 nl-code 7 nl-non-zeroes
--- 14 discrete-columns
--- Submitting model minvar with handle grid137000002
--- Executing after solve
...
--- GDXin=C:\answerv5\gams_srcdev\225j\grid137000003\gmsgrid.gdx
--- Removing handle grid137000003
```

The log will now contain some additional information about the submission, retrieval and removal of the solution instance. In the following sections we will make use of this additional information. You can find a complete example of a grid enabled transport model in the GAMS model library.

At a final note, we have made no assumptions about what kind of solvers and what kind of computing environment we will operate. The above example is completely platform and solver independent and it runs on your Windows laptop or on a massive grid network like the Condor system without any changes in the GAMS source code.

I.4 Advanced use of Grid Features

In this section we will describe a few special application requirements and show how this can be handled with the current system. Some of those applications may involve thousands of model instances with solution times of many hours each. Some may fail and require resubmission. More complex examples require communication and the use of GAMS facilities like the BCH (Branch&Cut&Heuristic) which submit other models from within a running solver.

I.4.1 Very Long Job Durations

Imagine a situation with thousands of model instances each taking between minutes and many hours to solve. We will break the master program into a submitting program, an inquire program and a final collection program. We will again use the previous example to demonstrate the principle. We will split the GAMS code of the modified QMEANVAR GAMS code into three components: qsubmit, qcheck and qreport.

The file qubmit.gms file will include everything up to and including the new submit loop. To save the instances we will need a unique Grid Directory and to restart the problem we will have to create a save file. The first job will then look a follows.

> gams qsubmit s=submit gdir=c:\test\grid

The solution of all the model instances may take hours. From time to time I can then run a quick inquiry job to learn about the stats. The following program qcheck.gms will list the current status:

```
parameter status(pp,*); scalar handle;
acronym BadHandle,Waiting,Ready;
loop(pp,
    handle := handlestatus(h(pp));
    if(handle=0,
        handle := BadHandle
    elseif handle=2,
        handle := Ready;
        minvar.handle = h(pp);
        execute_loadhandle minvar;
        status(pp,'solvestat') = minvar.solvestat;
```

```
status(pp,'modelstat') = minvar.modelstat;
status(pp,'seconds') = minvar.resusd;
else
handle := Waiting );
status(pp,'status') = handle );
display status;
```

To run the above program we will restart from the previous save file by using the restart or r parameter.

> gams qcheck r=submit gdir=c:\test\grid

The output may then look like:

```
173 PARAMETER status
    solvestat
                modelstat
                             seconds
                                          status
                                0.328
p1
        1.000
                     1.000
                                           Ready
p2
        1.000
                     1.000
                                0.171
                                           Ready
pЗ
                                         Waiting
p4
                                         Waiting
p5
        1.000
                     1.000
                                0.046
                                           Ready
```

You may want to do some more detailed analysis on one of the solved model instances. Then we may have a qanalyze.gms program that may look like and be called using the double dash option, which sets a GAMS environment variable:

> gams qanalyze r=submit gdir=c:\test\grid --instance=p4

Once all jobs are completed we can continue with the second part which will contain the collection loop, for simplicity without the repeat loop because we would not run the final collection program unless we are satisfied that we got most of what we wanted. Then the qreport.gms file could look like:

```
loop(pp$handlestatus(h(pp)),
  minvar.handle = h(pp);
  execute_loadhandle minvar;
  xres(i,pp) = x.l(i);
  report(pp,i,'inc') = xi.l(i);
  report(pp,i,'dec') = xd.l(i);
  display$handledelete(h(pp)) 'trouble deleting handles' ;
  h(pp) = 0 );
  xres(i,pp)$h(pp) = na;
  . . .
```

We would restart the above program from the save file that was created by the submitting job like:

```
> gams qreport r=submit gdir=c:\test\grid
```

Note that it would not be necessary to run the job out of the same directory we did the initial submission. We don't even have to run the same operating system.

I.5 Summary of Grid Features

To facilitate the asynchronous or parallel execution of the solve solution steps we have introduced three new functions, a new model attribute, a new gdx load procedure and a new GAMS option GridDir.

I.5.1 Grid Handle Functions

HandleCollect(handle) collects (loads) the solution if ready.

- 0 the model instance was not ready or could not be loaded
- 1 the model instance solution has been loaded

HandleStatus(handle) returns the status of the solve identified by handle. An execution error is triggered if GAMS cannot retrieve the status of the handle.

- 0 the model instance is not known to the system
- 1 the model instance exists but no solution process is complete
- 2 the solution process has terminated and the solution is ready for retrieval
- 3 the solution process signaled completion but the solution cannot be retrieved

HandleDelete(handle) returns the status of the deletion of the handle model instance. In case of a nonzero return an execution error is triggered.

0	the model instance has been removed
1	the argument is not a legal handle
2	the model instance is not known to the system
3	the deletion of the model instance encountered errors

HandleSubmit(handle) resubmits a previously created instance for solution. In case of a nonzero return an execution error is triggered.

0	the model instance has been resubmitted for solution
1	the argument is not a legal handle
2	the model instance is not known to the system
3	the completion signal could not be removed
4	the resubmit procedure could not be found
5	the resubmit process could not be started

In addition, GAMS will issue execution errors which will give additional information that may help to identify the source of problems. The property execerror can be used to get and set the number of execution errors.

I.5.2 Grid Model Attributes

mymodel.solvelink specifies the solver linking conventions

- 0 automatic save/restart, wait for completion, the default
- 1 start the solution via a shell and wait
- 2 start the solution via spawn and wait
- 3 start the solution and continue
- 4 start the solution and wait (same submission process as 3)
- 5 start the solution via shared library and wait

mymodel.handle specifies the current instance handle

This is used to identify a specific model instance and to provide additional information needed for the process signal management.

mymodel.number specifies the current instance number

Any time a solve is attempted for mymodel, the instance number is incremented by one and the handle is update accordingly. The instance number can be reset by the user which then resyncs the handle.

I.5.3 Grid Solution Retrieval

Execute_loadhandle *mymodel*;

This will update the GAMS data base with the status and solution for the current instance of *mymodel*. The underlying mechanism is a gdx file and operates otherwise like the execute_loadpoint procedure. Additional arguments can be used to retrieve information from the gdx solution file.

I.5.4 Grid Directory

The instantiated (generated) models and their corresponding solution are kept in unique directories, reachable from your submitting system. Each GAMS job can have only one Grid Directory. By default, the grid directory is assumed to be the scratch directory. This can be overwritten by using the GAMS parameter GridDir, or short GDir. For example:

>gams myprogram ... GDir=gridpath

If *gridpath* is not a fully qualified name, the name will be completed using the current directory. If the grid path does not exist, an error will be issued and the GAMS job will be terminated. A related GAMS parameter is the DcrDir (SDir for short). Recall the following default mechanism:

When a GAMS job starts, a unique process directory is created in the current (job submitting), directory. These directories are named 225a to 225z. When a GAMS job terminates it will remove the process directory at the completion of a GAMS job. Any file that has not been created by the GAMS core system will be flagged.

Using the program *gamskeep* instead of *gams* will call another exit script which (the default script) will do nothing and the process directory will not be removed.

If we do not specify a scratch directory, the scratch directory will be the same as the process directory. If we do not specify a grid directory, the grid directory will be the same as the scratch directory.

If there is a danger that some of the model instances may fail or we want to break the GAMS program into several pieces to run as separate jobs, we need to be careful not to remove the model instance we have not completely processed. In such cases, we have to use the GridDir option in order to be able to access previously created model instances.

I.6 Architecture and Customization

The current Grid Facility relies on very basic operating system features and does not attempt to offer real and direct job or process control. The file system is used to signal the completion of a submitted task and GAMS has currently no other way to interact with the submitted process directly, like forcing termination or change the priority of a submitted task. This approach has its obvious advantages and disadvantages. There are a number of attempts to use grid computing to provide value added commercial remote computing services, notably is SUN's recent commercial entry. Commercial services require transparent and reliable charge approaches and related accounting and billing features which are still missing or inadequate.

When GAMS executes a solve under solvelink=3 it will perform the following steps:

- 1. Create a subdirectory in the GridDir with the name gridnnn. Where nnn stands for the numeric value of the handle. The handle value is the internal symbol ID number x 1e6 + the model instance number. For example, in the QMEANVAR example the first grid subdirectory was grid137000002.
- 2. Remove the completion signal in case the file already exists. Currently the signal is a file called finished. For example, grid137000002/finished.
- 3. Create or replace a gdx file called **gmsgrid.gdx** which will contain a dummy solution with failed model and solver status. This file will be overwritten by the final step of the solution process and will be read when calling execute_loadhandle.
- 4. Place all standard GAMS solver interface files into the above instance directory.

5. Execute the submission wrapper called **gmsgrid.cmd** under Windows or **gmsgrid.run** under Unix. These submission scripts are usually located in the GAMS system directory but are found via the current path if not found in the GAMS system directory.

The grid **submission script gmsgrid.cmd** or **gmsgrid.run** is called with three arguments needed to make a standard GAMS solver call:

- 1. The solver executable file name
- 2. The solver control file name
- 3. The solver scratch directory

The submission script then does the final submission to the operating system. This final script will perform the following steps:

```
call the solver
call a utility that will create the final gdx file gmsgrid.gdx
set the completion signal finished
```

If we want to use the function handlesubmit() we also have to create the **gmsrerun.cmd** or **gmsrerun.run** script which could later be used to resubmit the job.

For example, the default submission script for Windows is shown below:

```
@echo off
: gams grid submission script
: arg1 solver executable
     2 control file
:
     3 scratch directory
: gmscr_nx.exe processes the solution and produces 'gmsgrid.gdx'
: note: %3 will be the short name because START cannot
        handle spaces and/or "...". We could use the original
:
        and use \sqrt[n]{s3} which will strip "..." and makes the name :
:
        short
:
: gmsrerun.cmd will resubmit runit.cmd
echo @echo off
                               > %3runit.cmd
echo %1 %2
                              >> %3runit.cmd
echo gmscr_nx.exe %2 >> %3runit.cmd
echo echo OK ^> %3finished
                              >> %3runit.cmd
echo exit
                              >> %3runit.cmd
echo @start /b %3runit.cmd ^> nul > %3gmsrerun.cmd
start /b %3runit.cmd > nul
exit
```

I.6.1 Grid Submission Testing

The grid submission process can be tested on any GAMS program without having to change the source text. The **solvelink=4** option instructs the solve statement to use the grid submission process and then wait until the results are available and then loads the solution into the GAMS data base. The **solvelink** option can be set via a GAMS command line parameter or via assignment to a the model attribute. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the reslim seconds allocated for this optimization job and report a failure if this limit has been exceed. After successful or failed retrieval of the solution gams will remove the grid directory, unless we have used gamskeep or have set the gams keep parameter.

I.7 Glossary and Definitions

BCH	Branch & Cut & Heuristic
Condor	High throughput computing system
GAMS	General Algebraic Modeling System
GDX	GAMS Data Exchange
HPC	High Performance Computing
SUN Grid Compute Utility	

J

Extrinsic Functions

J.1 Introduction

Functions play an important part in the GAMS language, especially for non-linear models. Similar to other programming languages, GAMS provides a number of built-in (intrinsic) functions. However, GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS Function Library Facility (6.3.3) provides the means for managing that trade-off. In this Appendix the extrisic function libraries which are included in the GAMS distribution are described.

In the tables that follow, the Model Function Type (first column) specifies in which models the function can legally appear with endogenous (non-constant) arguments. In order of least to most restrictive, the choices are any, NLP, DNLP or none.

The following conventions are used for the function arguments. Lower case indicates that an endogenous variable is allowed. Upper case indicates that a constant argument is required. The arguments in square brackets can be omitted and default values will be used.

J.2 Fitpack Library

FITPACK by Paul Dierckx¹ is a FORTRAN based library for one and two dimensional spline interpolation. This library has been repackaged to work with the GAMS Function Library Facility. As it can be seen in the GAMS Test Library model fitlib01 the function data needs to be stored in a GDX file fit.gdx containing a three dimensional parameter fitdata. The first argument of that parameter contains the function index, the second argument is the index of the supporting point and the last one needs to be one of w (weight), x (x-value), y (y-value) or z (z-value).

Function	Endogenous Classifica- tion	Description
<pre>fitFunc(FUNIND,x[,y])</pre>	DNLP	Evalute Spline
fitParam(FUNIND,PARAM[,VALUE])	none	Read or set parameters

Table J.1: Fitpack functions

¹Paul Dierckx, Curve and Surface Fitting with Splines, Oxford University Press, 1993, http://www.netlib.org/dierckx/

The function FitParam can be used to change certain parameters used for the evaluation:

- 1: Smoothing factor (S)
- 2: Degree of spline in direction x (Kx)
- 3: Degree of spline in direction y (Ky)
- 4: Lower bound of function in direction x (LOx)
- 5: Lower bound of function in direction y (LOy)
- 6: Upper bound of function in direction x (UPx)
- 7: Upper bound of function in direction y (UPy)

This library is made available by the following directive:

\$FuncLibIn <InternalLibName> fitfclib

J.3 Piecewise Polynomial Library

This library can be used to evaluate piecewise polynomial functions. The functions which should be evaluated need to be defined and stored in a GDX file like it is done in the GAMS Test Library model pwplib01:

```
* Define two piecewise polynomial functions
Table pwpdata(*,*,*) '1st index: function number, 2nd index: segment number, 3rd index: degree'
                 leftBound
                                  0
                                             1
                                                        2
       1.1
                 1
                                  2.4
                                             -2.7
                                                        0.3
       1.2
                 4
                                  5.6
                                            -4.3
                                                        0.5
       2.1
                 0
                                  0
                                             -6.3333
                                                        0
       2.2
                                  1.0370
                                           -12.5554
                                                        9.3333
                 0.3333
       2.3
                 0.6667
                                  9.7792
                                            -38.7791
                                                       29
* Write pwp data to gdx file read by external library
$gdxout pwp.gdx
$unload pwpdata
$gdxout
```

On each row of the Table pwpdata we have

FuncInd.SegInd leftBound Coef0 Coef1 Coef2 ...

FuncInd sets the function index. SegInd defines the index of the segment (or interval) which is decribed here. LeftBound gives the lower bound of the segment. The upper bound is the lower bound on the next row, or the upper bound for the variable if this is the last segment. CoefX defines the Xth degree coefficient of the polynomial corresponding to this segment.

This library is made available by the following directive:

\$FuncLibIn <InternalLibName> pwpcclib

Function	Endogenous Classifica- tion	Description
pwpFunc(FUNIND,x)	DNLP	Piecewise Polynomials

J.4 Stochastic Library

The stochastic library provides random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for certain distributions. This library is made available by the following directive:

\$FuncLibIn <InternalLibName> stodclib

Function	Description
SetSeed(SEED)	defines the seed for random number generator
Continous distributions	
beta(SHAPE_1,SHAPE_2)	Beta distribution with shape parameters SHAPE_1 and SHAPE_2, see MathWorld
cauchy(LOCATION,SCALE)	Cauchy distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld
ChiSquare(DF)	Chi-squared distribution with degrees of freedom DF, see MathWorld
exponential(LAMBDA)	Exponential distribution with rate of changes LAMBDA, see MathWorld
f(DF_1,DF_2)	F-distribution with degrees of freedom DF_1 and DF_2, see MathWorld
gamma(SHAPE,SCALE)	Gamma ditribution with shape parameter SHAPE and scale parameter SCALE, see MathWorld
gumbel(LOCATION,SCALE)	Gumbel distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld
invGaussian(MEAN,SHAPE)	Inverse Gaussian distribution with mean MEAN and scaling parameter SHAPE, see MathWorld
laplace(MEAN,SCALE)	Laplace distribution with mean MEAN and scale parameter SCALE, see MathWorld
logistic(LOCATION,SCALE)	Logistic distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld
logNormal(MEAN,STD_DEV)	Log Normal distribution with mean MEAN and standard deviation STD_DEV, see MathWorld
normal(MEAN,STD_DEV)	Normal distribution with mean MEAN and standard deviation STD_DEV, see MathWorld
pareto(SCALE,SHAPE)	Pareto distribution with scaling parameter SCALE and shape parameter SHAPE, see MathWorld
rayleigh(SIGMA)	Rayleigh distribution with parameter SIGMA, see MathWorld
<pre>studentT(DF)</pre>	Student's t-distribution with degrees of freedom DF, see MathWorld
triangular(LOW,MID,HIGH)	Triangular distribution between LOW and HIGH, MID is the most probable number, see $\underline{MathWorld}$
uniform(LOW,HIGH)	Uniform distribution between LOW and HIGH, see MathWorld
weibull(SHAPE,SCALE)	Weibull distribution with shape parameter SHAPE and scaling parameter SCALE, see MathWorld

Discrete distributions	
binomial(N,P)	Binomial distribution with number of trials N and success probability P in each trial, see MathWorld
geometric(P)	Geometric distribution with success probability P in each trial, see Math-World
hyperGeo(TOTAL,GOOD,TRIALS)	Hypergeometric distribution with total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS, see MathWorld
logarithmic(P-FACTOR)	Logarithmic distribution with parameter P-FACTOR, also called log-series distribution, see MathWorld
<pre>negBinomial(FAILURES,P)</pre>	Negative Binomial distribution with the number of failures until the experiment is stopped FAILURES and success probability P in each trial. The generated random number describes the number of successes until we reached the defined number of failures, see MathWorld
poisson(LAMBDA)	Poisson distribution with mean LAMBDA, see MathWorld
uniformInt(LOW,HIGH)	Integer Uniform distribution between LOW and HIGH, see MathWorld
	Table J.3: Random number generators

For each distribution in table J.3, the library offers four functions:

Function	Endogenous Classifica- tion	Description
d <distributionname></distributionname>	none	generates a random number
pdf <distributionname></distributionname>	DNLP (none for discrete distributions)	probability density function
cdf <distributionname></distributionname>	DNLP (none for discrete distributions)	cumulative distribution function
icdf <distributionname></distributionname>	DNLP (none for discrete distributions)	inverse cumulative distribution func- tion

Table J.4: Distribution functions

The function d<DistributionName> needs the arguments described in table J.3. The other functions get an additional argument at the first position: The point to evaluate. This parameter can be an endogenous variable. The following table shows all fours functions for the Normal distribution:
Function	Endogenous Classifica- tion	Description
dNormal(MEAN,STD_DEV)	none	generates a random number with Normal distribution
pdfNormal(x,MEAN,STD_DEV)	DNLP	probability density function for Normal distribution
cdfNormal(x,MEAN,STD_DEV)	DNLP	cumulative distribution function for Normal distribution
icdfNormal(x,MEAN,STD_DEV)	DNLP	inverse cumulative distribution function for Normal distribution

Table J.5: Normal distribution functions

J.5 LINDO Sampling Library

The LINDO Sampling Library provides samples of random numbers for certain distributions. It is made available by the following directive:

\$FuncLibIn <InternalLibName> lsadclib

The function sampleLS<DistributionName> creates a sample of the specified distribution according to the distribution parameters and returns a HANDLE that references the sample as illustrated in the example down below. The Parameter SAMSIZE must be specified and describes the size of the sample while VARRED is optional and provides the possibility to define a variance reduction method (0=none, 1=Latin Hyper Square, 2=Antithetic). If omitted Latin Hyper Square Sampling is used.

Function	Description
getSampleValues(HANDLE)	Retrieve sampling
induceCorrelation(CORTYPE)	Induce correlation that has to be set with setCorrelation before. CORTYPE describes the correlation type and must be one of 0 (PEARSON), 1 (KENDALL) or 2 (SPEARMAN).
<pre>setCorrelation(SAMPLE1,SAMPLE2,COR)</pre>	Define correlation between two samplings
setSeed(SEED)	Define the seed for random number generator
Continous distributions	
<pre>beta(SHAPE_1,SHAPE_2,SAMSIZE[,VARRED])</pre>	Beta distribution specified by two shape parameters
<pre>cauchy(LOCATION,SCALE,SAMSIZE[,VARRED])</pre>	Cauchy distribution specified by location and scale parameter
chisquare(DF,SAMSIZE[,VARRED])	Chi-Squared distribution specified by degrees of freedom
exponential(RATE,SAMSIZE[,VARRED])	Exponential distribution specified by rate of change
f(DF_1,DF_2,SAMSIZE[,VARRED])	F distribution specified by degrees of freedom
<pre>gamma(SHAPE,SCALE,SAMSIZE[,VARRED])</pre>	Gamma distribution specified by shape and scale parameter

<pre>gumbel(LOCATION,SCALE,SAMSIZE[,VARRED])</pre>	Gumbel distribution specified by location and scale parameter
<pre>laplace(LOCATION,SCALE,SAMSIZE[,VARRED])</pre>	Laplace distribution specified by location and scale parameter
<pre>logistic(LOCATION,SCALE,SAMSIZE[,VARRED])</pre>	Logistic distribution specified by location and scale parameter
<pre>lognormal(LOCATION,SCALE,SAMSIZE[,VARRED])</pre>	Log Normal distribution specified by location and scale parameter
normal(MEAN,STD_DEV,SAMSIZE[,VARRED])	Normal distribution specified by given mean and stan- dard deviation
<pre>pareto(SCALE,SHAPE,SAMSIZE[,VARRED])</pre>	Pareto distribution specified by shape and scale parameter
<pre>studentt(DF,SAMSIZE[,VARRED])</pre>	Student's t-distribution specified by degrees of freedom
<pre>triangular(LOW,HIGH,MID,SAMSIZE[,VARRED])</pre>	Triangular distribution specified by lower and upper limit and mid value
uniform(LOW,HIGH,SAMSIZE[,VARRED])	Uniform distribution specified by the given bounds
<pre>weibull(SCALE,SHAPE,SAMSIZE[,VARRED])</pre>	Weibull distribution specified by scale and shape parameter

Discrete distributions	
<pre>binomial(N,P,SAMSIZE[,VARRED])</pre>	Binomial distribution specified by number of trials N and success probability P in each trial
geometric(P,SAMSIZE[,VARRED])	Geometric distribution specified by success probability \mathtt{P}
hypergeo(TOTAL,GOOD,TRIALS,SAMSIZE[,VARRED])	Hypergeometric distribution specified by total number of elements, number of good elements and number of trials
<pre>logarithmic(P-FACTOR,SAMSIZE[,VARRED])</pre>	Logarithmic distribution specified by P-Factor
<pre>negbinomial(SUCC,P,SAMSIZE[,VARRED])</pre>	Negative Binomial distribution specified by the number of successes and the probability of success. The gener- ated random number describes the number of failures until we reached the defined number of successes.
<pre>poisson(MEAN,SAMSIZE[,VARRED])</pre>	Poisson distribution specified by mean

Table J.6: LINDO sampling functions

The following example illustrates the use of the sample generator and shows how the commands setCorrelation and induceCorrelation work:

\$funclibin lsalib lsadclib

```
function normalSample /lsalib.SampleLSnormal /
    getSampleVal /lsalib.getSampleValues /
    setCor /lsalib.setCorrelation /
    indCor /lsalib.induceCorrelation /;
scalar d,h,k;
h = normalSample(5,2,12);
k = normalSample(5,2,12);
```

```
set i /i01*i12/;
parameter sv(i);
loop(i,
 sv(i) = getSampleVal(k);
);
display sv;
loop(i,
  sv(i) = getSampleVal(h);
);
display sv;
d=setCor(h,k,-1);
d=indCor(1);
loop(i,
 sv(i) = getSampleVal(k);
);
display sv;
loop(i,
 sv(i) = getSampleVal(h);
);
display sv;
```

The resulting output shows that the values of sv are restructured according to the desired correlation:

18 F	ARAMETER sv				
i01 7.610,	i02 5.710,	i03 3.755,	i04 5.306,	i05 2.382,	i06 2.174
i07 6.537,	i08 4.975,	i09 8.260,	i10 3.067,	i11 6.216,	i12 4.349
22 F	ARAMETER sv				
i01 7.610,	i02 5.710,	i03 3.755,	i04 5.306,	i05 2.382,	i06 2.174
i07 6.537,	i08 4.975,	i09 8.260,	i10 3.067,	i11 6.216,	i12 4.349
28 F	ARAMETER sv				
i01 2.382,	i02 4.349,	i03 6.216,	i04 4.975,	i05 7.610,	i06 8.260
i07 3.067,	i08 5.306,	i09 2.174,	i10 6.537,	i11 3.755,	i12 5.710
32 F	PARAMETER sv				
i01 7.610,	i02 5.710,	i03 3.755,	i04 5.306,	i05 2.382,	i06 2.174
i07 6.537,	i08 4.975,	i09 8.260,	i10 3.067,	i11 6.216,	i12 4.349

J.6 Trigonometric Library

This library comes in three versions. They come with the GAMS Test Library models trilib01, trilib02 and trilib03 and can be found compiled and as source code written in C, Delphi and FORTRAN respectively.

Function	Endogenous Classifica- tion	Description
<pre>setMode(MODE)</pre>	none	sets mode globally, could still be overwritten by MODE at (Co)Sine call, possible values are 0=radians and 1=degree
<pre>cosine(x[,MODE])</pre>	NLP	returns the cosine of the argument x, default setting: $MODE = 0$
<pre>sine(x[,MODE])</pre>	NLP	returns the sine of the argument x, default setting: $MODE = 0$
pi	any	value of $\pi = 3.141593$

Table J.7: Trigonometric functions

 \mathbf{K}

Installation and System Notes

Windows

INSTALLATION

- 1. Run windows_x86_32.exe (Windows 32bit) or windows_x64_64.exe (Windows 64bit): Both files are either available on the GAMS web site or on the distribution DVD (in the directory windows). The 32 bit version works both on a 32bit and on a 64bit operating system. Please note that the installation may require administrative privileges on your machine. You have two options to run the installer: In default or advanced mode. In the default mode the installer will prompt you for the name of the directory in which to install GAMS. We call this directory the 'GAMS directory'. You may accept the default choice or pick another directory. Please remember: if you want to install two different versions of GAMS, they must be in separate directories. If you choose to use the advanced mode, the installer will also ask you for a name of a start menu folder, if GAMS should be installed for all users, if the GAMS directory should be added to the PATH environment variable and if a desktop icon should be created.
- 2. Copy the GAMS license file: You will be asked for the GAMS license file (gamslice.txt) during the installation. If you are not sure, if you have a license file, choose 'No license, demo only' when asked for the GAMS license options. You can always do this later. If no valid license file is found, GAMS will still function in the demonstration mode, but will only solve small problems. All demonstration and student systems do not include a license file. If you have a license file you wish to copy to the GAMS directory at this time, answer 'Copy license file'. You will now be given the opportunity to browse the file system and find the license file gamslice.txt. When you have found the correct file, choose 'open' to perform the copy. Instead of copying a license file you could also copy the content of that file to the clipboard. If you have done this, select 'Copy license text from clipboard'.
- 3. Create a project file: If this is the first installation of GAMS on your system, the installation program will create a default GAMS project in a subdirectory of your home folder. Otherwise your existing GAMS projects will be preserved.
- 4. Choose the default solvers: Run the GAMS IDE by double clicking gamside.exe from the GAMS directory. To view or edit the default solvers, choose File \rightarrow Options \rightarrow Solvers from the IDE. You can accept the existing defaults if you wish, but most users want to select new default solvers for each model type.
- 5. Run a few models to test the GAMS system: The on-line help for the IDE (Help \rightarrow GAMS IDE Help Topics \rightarrow Guided Tour) describes how to copy a model from the GAMS model library, run it and view the solution. To test your installation, run the following models from the GAMS model library:

LP: trnsport (objective value: 153.675) NLP: chenery (objective value: 1058.9) MIP: bid (optimal solution: 15210109.512) MINLP: procsel (optimal solution: 1.9231) MCP: scarfmcp (no objective function) MPSGE: scarfmge (no objective function)

COMMAND LINE INSTALLATION

Users wishing to use GAMS from the command line (aka the console mode) may want to perform the following steps after they have installed the system as described above. These steps are not necessary to run GAMS via the IDE.

- 1. Run the program gamsinst: gamsinst is a command line program used to install and configure GAMS. It prompts the user for default solvers to be used for each model type. If possible choose solvers you have licensed, since unlicensed solvers will only run in demonstration mode. The solver defaults can be changed by:
 - (a) rerunning gamsinst and resetting the default values
 - (b) setting a command line default, e.g. gams trnsport lp=bdmlp
 - (c) by an option statement in the GAMS model, e.g: option lp=bdmlp;

The system wide solver defaults are shared by the command line and the GAMS IDE, so you can also choose to set these defaults using the GAMS IDE.

- 2. Add the GAMS directory to your path. To avoid having to type in an absolute path name each time you run GAMS, we recommend adding the GAMS directory to your PATH when using the console mode (not the GAMS IDE) version of GAMS. In case more than one GAMS system is installed on the machine, separate paths have to be set before invoking each version. Under Windows XP/Vista the following procedure must be applied to add the GAMS directory to your path:
 - Open the System Properties under the Control Panel.
 - On the Advanced tab click on the Environment Variables button and select the existing variable Path. Click Edit.
 - In the Value Box add the GAMS directory to the path as the following example illustrates: c:\your\current\path\setting;C:\gams and click OK.

Unix

INSTALLATION

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure:

- 1. Choose a location where you want to create the GAMS system directory (the GAMS system directory is the directory where the GAMS system files should reside). At this location the GAMS installer will create a subdirectory with a name that indicates the distribution of GAMS you are installing. For example, if you are installing the 23.9 distribution, in /usr/gams, the installer will create the GAMS system directory /usr/gams/gams23.9_linux_x86_32_sfx. If the directory where you want to install GAMS is not below your home directory, you may need to have root privileges on the machine.
- 2. Make the directory to create the GAMS system directory in, for instance /usr/gams. Go to this directory. Make sure pwd returns the name of this directory correctly.
- 3. Transfer the distribution file into the new directory. This file is available from the GAMS DVD or via the web in one large self-extracting zip archive with a <code>_sfx.exe</code> file extension. You can run the archive (e.g. <code>linux_x86_32_sfx.exe</code> on a Linux 32bit system) directly from the DVD to extract the necessary folder to your directory. For example, you might execute the following commands:

- 4. To mount the GAMS DVD, you may need to be logged in as root. We assume you want to mount the DVD over the directory /dvd. If the directory you want to mount over does not exist, you must create it now. Once this directory is created, mount the DVD, using the appropriate command. The correct arguments for the mount command vary from machine to machine. After mounting the DVD, view the README.TXT file on it to find the subdirectory containing the GAMS system for your machine.
- 5. If you transferred the distribution file via the web, check that it has the execute permission set. If you are not sure how to do this, just type in the command, e.g. chmod 755 linux_x86_32_sfx.exe.
- 6. Check if the file gamslice.txt exists in the GAMS system directory. The license files is nowadays sent via email. If no license file is present, GAMS will still function in the demonstration mode but can only solve small problems. Student and demonstration systems do not include a license file. A license file can easily be added later, so if you cannot find a license file, you can safely proceed without one.
- 7. Run the program ./gamsinst. This will unpack files if necessary. It will also prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:
 - (a) rerunning ./gamsinst and resetting the default values
 - (b) setting a command line default, e.g. gams trnsport lp=bdmlp
 - (c) an option statement in the GAMS model, e.g. option lp=bdmlp;
- 8. Add the GAMS system directory to your path (see 'ACCESS TO GAMS' below).
- 9. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:

LP: trnsport (objective value: 153.675) NLP: chenery (objective value: 1058.9) MIP: bid (optimal solution: 15210109.512) MINLP: procsel (optimal solution: 1.9231) MCP: scarfmcp (no objective function) MPSGE: scarfmge (no objective function)

10. If you move the GAMS system to another directory, remember to rerun ./gamsinst. It is also good practice to rerun ./gamsinst when you add or change your license file if this has changed the set of licensed solvers.

ACCESS TO GAMS

To run GAMS you must be able to execute the GAMS programs located in the GAMS system directory. There are several ways to do this. Remember that the GAMS system directory in the examples below may not correspond to the directory where you have installed your GAMS system.

1. If you are using the C shell (csh) and its variants you can modify your .cshrc file by adding the second of the two lines given below:

```
set path = (/your/previous/path/setting )
set path = ( $path /usr/gams/gams23.9_linux_x86_32_sfx ) # new
```

2. Those of you using the Bourne (sh) or Korn (ksh) shells and their variants can modify their .profile file by adding the second of the three lines below:

```
PATH=/your/previous/path/setting
PATH=$PATH:/usr/gams/gams23.9_linux_x86_32_sfx # new
export PATH
```

You should log out and log in again after you have made any changes to your path.

3. You may prefer to use an alias for the names of the programs instead of modifying the path as described above. C shell users can use the following commands on the command line or in their .cshrc file:

```
alias gams /usr/gams/gams23.9_linux_x86_32_sfx/gams
alias gamslib /usr/gams/gams23.9_linux_x86_32_sfx/gamslib
```

The correct Bourne or Korn shell syntax (either command line or .profile) is:

alias gams=/usr/gams/gams23.9_linux_x86_32_sfx/gams alias gamslib=/usr/gams/gams23.9_linux_x86_32_sfx/gamslib

Again, you should log out and log in in order to put the alias settings in .cshrc or .profile into effect.

4. Casual users can always type the absolute path names of the GAMS programs, e.g.: /usr/gams/gams23.9_linux_x86_32_sfx/gams trnsport

Index

++, circular operator, 125 --, circular operator, 125 ..., after equation name, 72 /, cursor control, 142 Qn, cursor control, 142 #n, cursor control, 142 abort dollar control option, 204 abs, function, 55 acronym, 28, 167 action GAMS call parameter, 177 activity level (.1) or (.L), 67, 68, 85, 95 ALAN, example from GAMSLIB, 89, 91 algorithm, 167 Implementation of non-standard, 86 alias, 28, 167 statement, 38 all, defining a model, 96 ALUM, example from GAMSLIB, 39 and, relational operator, 108 ANDEAN, example from GAMSLIB, 54 appendexpand GAMS call parameter, 177 appendlog GAMS call parameter, 177 appendout GAMS call parameter, 178 arccos, function, 55 arcsin, function, 55 arctan, function, 55 arctan2, function, 55 arithmetic operations, 53, 63, 105 addition, 53 division, 53 exponentiation, 53multiplication, 53prod, 54smax, 54smin, 54subtraction, 53sum, 54 assigned, reference type, 92 assignment, 68, 167 conditional, 111 indexed, 51

issues with controlling indices, 52over subsets, 52scalar, 51 statement, 51 to dynamic sets, 118 using labels explicitly, 52 asterisk in set definitions, 37 marking errors, 96, 103 use in comments, 32basic, 167 batinclude dollar control option, 204 beta, distribution, 275, 277 Beta, function, 55 betaReg, function, 55 binary, operator, 109, 132 binding, 167 binomial, distribution, 276, 278 binomial, function, 55 bool_and, function, 58 bool_eqv, function, 58 bool_imp, function, 58 $bool_not(x)$, function, 58 bool_or, function, 58 bool_xor, function, 58 boolean, operations, 114 bounds, 167 on variables, 67 branching priority value (.prior), 67 bratio GAMS call parameter, 178 GAMS option, 243 call dollar control option, 205call.Async dollar control option, 206 card, operator on sets, 125 case GAMS call parameter. 178 cauchy, distribution, 275, 277

ceil, function, 55, 74 centropy, function, 55 cerr GAMS call parameter, 178 character set, valid, 35 charset GAMS call parameter, 178 CHENERY, example from GAMSLIB, 68, 74, 115 chiSquare, distribution, 275, 277 clear dollar control option, 206 clearError(s) dollar control option, 206 cns GAMS call parameter, 178 GAMS option, 243 CNS, model type, 16, 78 column, 167 listing, 96 comma in data lists, 32 in put statements, 138 comment dollar control option, 207 using \$eolcom, 33 using \$inlinecom, 33 compilation, 167 actions during ..., 177 errors. 103 errors at \dots time, 104output, 90 complement, a set operation, 121compress, 259 dollar control option, 207Compressed and Encrypted Input Files, 259 conditional expressions numerical values, 109 operator precedence, 109 using set membership, 108 with logical operators, 108 with numerical relationship operators, 107 constant set, 167constraint, 167 continuous, 167 control, reference type, 92 controlling index, 52 set, 47, 153 controlling sets, 167 \cos , function, 55 \cosh , function, 55 cosine, function, 280CRAZY, example from GAMSLIB, 63, 105 curdir GAMS call parameter, 178 cvPower, function, 55

data

entered as parameters, 44 entered as tables, 46 entry, 43

handling aspects of equations, 75 manipulations with parameters, 51 type, 29data types, 167 decimals GAMS option, 243 decimals, global option, 134 declaration, 167 of a model, 77 parameter, 44 scalar, 43 separation between - and definition of -, 28 statements. 28 table, 45decompress, 259 dollar control option, 207 default, 167 defined, a reference type, 92 definition, 168 of a model, 77 of data, 29 of equation, 72of scalars, 44 of symbols, 29 statements, 28 definition statements, 168 dformat GAMS call parameter, 178 difference, set operation, 120, 121 direction, 168 of optimization, 84, 98 discontinuous, 168 derivate, 74 functions, 78 discrete, 168 variables, 66, 78 display controls local, 134 example, 131 generating data in list format, 135 global controls, 134 introduction, 131 label order, 132 syntax, 131 distributions beta, 275, 277 binomial, 276, 278 cauchy, 275, 277 chiSquare, 275, 277 exponential, 275, 277 f, 275, 277 gamma, 275, 277 geometric, 276, 278 gumbel, 275, 277 hyperGeo, 276, 278 invGaussian, 275

laplace, 275, 277 logarithmic, 276, 278 logistic, 275, 277 logNormal, 275, 277 negBinomial, 276, 278 normal, 275, 277 pareto, 275, 277 poisson, 276, 278 rayleigh, 275 studentT, 275, 277 triangular), 275, 277 uniform, 275, 277 uniformInt, 276 weibull, 275, 277 div, function, 55 div0, function, 55 dnlp GAMS call parameter, 179 GAMS option, 243 DNLP, model type, 15, 78 dollar dollar control option, 207 dollar condition control over the domain of definition, 115 example, 110 in equations, 115, 120in indexed operations, 120nested, 110 on the left. 111 on the right, 112with dynamic sets, 119 within indexed operations, 114 within the algebra, 115dollar control option, 168 Introduction, 201 Syntax, 201 dollar operator, 110, 168 domain checking, 38, 168 domain definition, 168domain restriction condition, 168 domlim GAMS call parameter, 179 GAMS option, 243 option, 98, 105 dot in equation definitions, 72 in level and marginal listings, 102 in many to many mappings, 40 in parameter definition, 45 in set definition, 40in sets, 40in tables, 47 double dollar control option, 207 driving set, 168 drop

dollar control option, 208 dropEnv dollar control option, 208 dropGlobal dollar control option, 208 dropLocal dollar control option, 208 dual value (.m), 67dumpopt GAMS call parameter, 179 dumpparms GAMS call parameter, 181 dynamic set, 168 assigning membership membership to, 117 assignments over the domain of, 118dollar assignments, 119 equations defined over the domain of, 119 example, 117 in equations, 120indexed operations, 120 introduction, 117 syntax, 117 using dollar controls with, 119 with multiple indices, 118e-format, 134, 168 echo dollar control option, 208 echoN dollar control option, 208eDist, function, 55 eject dollar control option, 208 GAMS option, 243 else dollar control option, 209 elseIf dollar control option, 209 elseIfE dollar control option, 209 elseIfI dollar control option, 209 emp GAMS call parameter, 181 EMP, model type, 78 encrypt, 259 dollar control option, 209 end of line, 31, 46 endIf dollar control option, 209 endogenous, 168 arguments, 74 entropy, function, 55 eolcom dollar control option, 209 eolonly

GAMS call parameter, 181 eps a reserved word, 30 definition, 63 used with variables, 102eq, a relational operator, 107 equation, 168indexed, 73 listing, 95 scalar, 73 equation declaration, 71 example, 71, 72 syntax, 71 equation definition arithmetic operators, 74 functions, 74 preventing undefined operations, 74 syntax, 72 errmsg GAMS call parameter, 181 errnam GAMS call parameter, 182 error dollar control option, 209 GAMS call parameter, 182 handling, 63 no solution, 98other, 99 reporting, 102 reporting compilation, 103 reporting compilation time errors, 104 reporting execution errors, 105 reporting solve errors, 105 setup failure, 99 unknown, 98 errorf, function, 55 errorLevel, function, 59 errorlog GAMS call parameter, 182 escape dollar control option, 209 etlim GAMS call parameter, 182 eval dollar control option, 210 evalGlobal dollar control option, 211 evalLocal dollar control option, 211 evaluation error limit, 99 exception handling, 93 handling in equations, 115execerr GAMS call parameter, 182 execError, function, 59

execmode GAMS call parameter, 182 execSeed, function, 55 execution, 168 errors, 105 execution statements, 168 exit dollar control option, 211 exogenous, 168 exp, function, 55expand GAMS call parameter, 182 explanatory text, 66, 77, 168 exponent, 54, 168 exponential, distribution, 275, 277 expose dollar control option, 212 extended value, 102extended arithmetic, 168 **Extrinsic Functions** Fitpack Library, 273 Introduction, 273 LINDO Sampling Library, 277 Piecewise Polynomial Library, 274 Stochastic Library, 275 Trigonometric Library, 280 f. distribution, 275, 277 fact, function, 55 feasible, 168 feasible solution, 169 ferr GAMS call parameter, 183 FERTD, example from GAMSLIB, 112 FERTS, example from GAMSLIB, 116 file GAMS statement, 137 defining, 139 summary, 102 filecase GAMS call parameter, 183 fitFunc, function, 273 fitParam, function, 273 floor, function, 55 for example, 157 statement, 156 syntax, 157 forcework GAMS call parameter, 184 forlim GAMS call parameter, 184 frac, function, 55 fsave

GAMS call parameter, 184

funcLibIn dollar control option, 212functions abs. 55 $\arccos, 55$ $\arcsin, 55$ arctan, 55 arctan2, 55 Beta, 55 betaReg, 55 binomial, 55 bool_and, 58 bool_eqv, 58 bool_imp, 58 $bool_not(x), 58$ bool_or, 58 bool_xor, 58 ceil, 55 centropy, 55 $\cos, 55$ $\cosh, 55$ $\cosine, 280$ cvPower, 55 div, 55 div0, 55 eDist, 55entropy, 55 errorf, 55 errorLevel, 59 execError, 59 execSeed, 55 $\exp, 55$ fact, 55 fitFunc, 273 fitParam, 273 floor, 55frac, 55gamma, 55gammaReg, 55 gamsRelease, 59 gamsVersion, 59 gday, 59 gdow, 59ghour, 59 gleap, 59 gmillisec, 59 gminute, 59 gmonth, 59 gsecond, 59 gyear, 59 handleCollect, 59 handleDelete, 59 handleStatus, 59 handleSubmit, 59 heapFree, 59 heapLimit, 59

heapSize, 59 ifThen, 58 jdate, 59 jnow, 59 jobHandle, 59 jobKill, 59 jobStatus, 59 jobTerminate, 59 jstart, 59 jtime, 59 licenseLevel, 59 licenseStatus, 59 log, 55 $\log 10, 55$ $\log 2, 55$ logBeta, 55 $\log Gamma, 55$ mapVal, 55 max, 55 maxExecError, 59 min, 55mod, 55 ncpCM, 55 ncpF, 55ncpVUpow, 55 ncpVUsin, 55 normal, 55 pi, 55, 280 poly, 55 power, 55pwpFunc, 274 randBinomial, 55 randLinear, 55 randTriangle, 55 $rel_eq, 58$ rel_ge, 58 rel_gt, 58 rel_le, 58 rel_lt, 58 rel_ne, 58 round, 55 rPower, 55 setMode, 280 sigmoid, 55 sign, 55signPower, 55 $\sin, 55$ sine, 280 $\sinh, 55$ sleep, 59 slexp, 55sllog10, 55 slrec, 55sqexp, 55sqlog10, 55 sqr, 55

sqrec, 55 sqrt, 55 tan, 55 tanh, 55 timeClose, 59 timeElapsed, 59 timeEtart, 59 timeStart, 59 trunc, 55 uniformInt, 55 vcPower, 55

g205

GAMS call parameter, 184 gamma, distribution, 275, 277 gamma, function, 55 gammaReg, function, 55 GAMS call parameter action, 176 appendexpand, 176 appendlog, 176 appendout, 176 bratio, 176 case, 176 cerr, 176 charset, 176 cns. 176 curdir, 176 dformat, 176 dnlp, 176 domlim, 176 dumpopt, 176 dumpparms, 176 emp, 176 eolonly, 176 errmsg, 176 errnam, 176 error, 176errorlog, 176 etlim, 176 execerr, 176 execmode, 176 expand, 176 ferr, 176 filecase, 176 forcework, 176 forlim, 176 g205, 176 gdx, 176 gdxcompress, 176 gdxconvert, 176 griddir, 176gridscript, 176 heaplimit, 176

holdfixed, 176 ide, 176 input, 176 inputdir, 176 inputdir1, 176 iterlim, 176 jobtrace, 176 keep, 176 libincdir, 176 license, 176 logfile, 176 logline, 176 logoption, 176 lp, 176 maxprocdir, 176 mcp, 176 minlp, 176 mip. 176 miqcp, 176 mpec, 176 multipass, 176 nlp, 176 nodlim, 176on115, 176 optca, 176 optcr, 176optdir, 176 optfile, 176 output, 176 pagecontr, 176 pagesize, 176 pagewidth, 176 parmfile, 176 plincense, 176 procdir, 176 profile, 176 profilefile, 176 profiletol, 176 putdir, 176 qcp, 176 reference, 176 reslim, 176 restart, 176 rminlp, 176 rmip, 176 rmiqcp, 176 rmpec, 176 save, 176 savepoint, 176 scrdir, 176 scrext, 176 scriptexit, 176 scriptfrst, 176 scriptnext, 176 scrnam, 176 seed, 176

solprint, 176 solvelink, 176 solvercntr, 176 solverdict, 176 solverinst, 176 solvermatr, 176 solversolu, 176 solverstat, 176 stepsum, 176 stringchk, 176 subsys, 176 suppress, 176 symbol, 176 sys11, 176 sysdir, 176 sysincdir, 176 sysout, 176 tabin, 176 tformat, 176 threads, 176 timer, 176 trace, 176 traceopt, 176 user1, 176 warnings, 176 workdir, 176 workfactor, 176 workspace, 176 xsave. 176 zerores, 176 zeroresrep, 176 GAMS coordinator, 169 GAMS execution output column listing, 96 equation listing, 95Model statistics, 96 solve summary, 97 GAMS output echo print, 90 example, 89 introduction, 89 report summary, 102 solution listing, 101symbol listing map, 93 symbol reference map, 91 GAMS call Introduction, 175 specifying options, 175 GAMS language items Characters, 30 GAMS language items, 29 Comments, 32 declarative statements, 28 Delimiters, 32 identifiers, 30 Labels, 31

Numbers, 32 reserved words, 30 text, 31 GAMSLIB, 173 gamsRelease, function, 59 gamsVersion, function, 59 gday, function, 59 gdow, function, 59 gdx GAMS call parameter, 184gdxcompress GAMS call parameter, 184 gdxconvert GAMS call parameter, 185 gdxIn dollar control option, 212 gdxOut dollar control option, 212 ge, relational operator, 30, 107 geometric, distribution, 276, 278 ghour, function, 59 gleap, function, 59 gmillisec, function, 59 gminute, function, 59 gmonth, function, 59 goto dollar control option, 212 grid computing, 263 griddir GAMS call parameter, 185 gridscript GAMS call parameter, 185 gsecond, function, 59 gt, relational operator, 30 GTM, example from GAMSLIB, 114, 115 gumbel, distribution, 275, 277 gyear, function, 59 handleCollect, function, 59 handleDelete, function, 59 handleStatus, function, 59 handleSubmit, function, 59 heapFree, function, 59 heaplimit GAMS call parameter, 185 heapLimit, function, 59 heapSize, function, 59 hidden dollar control option, 213 hiddenCall dollar control option, 213 hide dollar control option, 213 holdfixed GAMS call parameter, 185 hyperGeo, distribution, 276, 278

ide GAMS call parameter, 185 identifiers, 169 if dollar control option, 213 if-elseif-else a statement, 154 example, 155 syntax, 154 ifE dollar control option, 214 ifI dollar control option, 215 ifThen dollar control option, 215 ifThen, function, 58 ifThenE dollar control option, 216 ifThenI dollar control option, 216 impl-asn, reference type, 92 include dollar control option, 216 index position(s), 169 indices, controlling, 54 INDUS, example from GAMSLIB, 48 inequality constraint, 169 inf an extended range value, 105 as a variable bound, 75, 132 extended range value, 32, 63 variable bound, 66 infeasible, 96, 98, 102, 169 infes, solution marker, 102 initial values, 28, 68 initialization, 92, 169 of data, 43 of parameters, 44 inlinecom dollar control option, 217input GAMS call parameter, 185 inputdir GAMS call parameter, 185 inputdir1 to inputdir40 GAMS call parameter, 186 integer infeasible, 98 solution, 98 variable, 67, 78 integer1 to integer5 GAMS call parameter, 186 intermediate infeasible, 98 noninteger, 98 nonoptimal, 98

intersection, set operation, 121 invGaussian, distribution, 275 iteration default limit, 98 interrupt, 99 iterlim GAMS call parameter, 186 GAMS option, 243 option, 98, 99 jdate, function, 59 inow, function, 59 jobHandle, function, 59 jobKill, function, 59 jobStatus, function, 59 jobTerminate, function, 59 iobtrace GAMS call parameter, 186 jstart, function, 59 jtime, function, 59 keep GAMS call parameter, 186 kill dollar control option, 217 KORPET, example from GAMSLIB, 46 label, 29, 31, 40 dollar control option, 217 order, 135 order on displays, 132 quoted, 31, 36 row and column, 47 unquoted, 31using in equations, 73 laplace, distribution, 275, 277 le, relational operator, 30, 107 legal characters, 30level, 68 libincdir GAMS call parameter, 187 libinclude dollar control option, 217license GAMS call parameter, 187 licenseLevel, function, 59 licenseStatus, function, 59 limcol GAMS call parameter, 187 GAMS option, 243 limrow GAMS call parameter, 187 GAMS option, 243 lines dollar control option, 218 list, 169 of labels using Asterisks, 37

list format, 169 load dollar control option, 218 loadDC dollar control option, 219 loadDCM dollar control option, 219 loadDCR dollar control option, 219 loadM dollar control option, 219 loadR dollar control option, 219 locally infeasible, 98 optimal, 98 log dollar control option, 219 function, 74 log, function, 55log10, function, 55 log2, function, 55 logarithmic, distribution, 276, 278 logBeta, function, 55 logfile GAMS call parameter, 187 logGamma, function, 55 logistic, distribution, 275, 277 logline GAMS call parameter, 188 logNormal, distribution, 275, 277 logoption GAMS call parameter, 189 loop example, 154statement, 153 syntax, 153 lower bound, 67(.lo), 67 lower case, 30lp GAMS call parameter, 189 GAMS option, 243 LP, model type, 15, 78 lt, relational operator, 30, 107 macro dollar control option, 220 mapping sets, 39 maps symbol listing, 93 symbol reference, 91 mapVal, function, 55 MARCO, example from GAMSLIB, 85 marginal, 69, 102, 169

value (.m), 67

matrix element, 169 max, function, 55, 74 maxcol dollar control option, 221 maxExecError, function, 59 maxGoto dollar control option, 222 maximizing, 84 maxprocdir GAMS call parameter, 189 mcp GAMS call parameter, 189 GAMS option, 243 MCP, model type, 16, 78 MEXSS, example from GAMSLIB, 151 min, function, 55, 74 mincol dollar control option, 222 minimizing, 83, 84 minlp GAMS call parameter, 189 GAMS option, 244 MINLP, model type, 16, 78 mip GAMS call parameter, 189 GAMS option, 244 MIP, model type, 15, 78, 99 miqcp GAMS call parameter, 189 GAMS option, 244 MIQCP, model type, 15, 78 mod, function, 55 model library, 173 statistics, 96 status, 98 styles, 29 syntax of statement, 77 types, 78 model status error no solution, 98infeasible, 98 integer solution, 98 model attributes bRatio. 79 cheat, 79 cutoff, 79 dictFile, 79 domLim, 79 domUsd, 82 etAlg, 82 etSolve, 82 etSolver, 82 handle, 82 holdFixed, 79 integer1, 79

integer2, 79 integer3, 79 integer4, 79 integer5, 79 iterLim, 79 iterUsd, 82 limCol, 79 limRow, 79 line, 82 linkUsed, 82 modelStat, 82 nodLim, 79 nodUsd. 82 number, 82 numDepnd, 82 numDVar, 82 numEqu, 82 numInfes, 82 numNLIns, 82 numNLNZ, 82 numNOpt, 82 numNZ, 82 numRedef, 82 numVar, 82 numVarProj, 82 objEst, 82 objVal, 82 optCA, 79 optCR, 79 optFile, 79 priorOpt, 79 procUsed, 82 real1, 79 real2, 79 real3, 79 real4, 79 real5, 79 reform, 79 resGen, 82 resLim, 79 resUsd, 82 rObj, 82 savePoint, 79 scaleOpt, 79 solPrint. 79 solveLink, 79 solveOpt, 79 solveStat, 82sumInfes, 82 sysOut, 79 threads, 79 tolInfeas, 79 tolInfRep, 79 tolProj, 79 tryInt, 79 tryLinear, 79

workFactor, 79 workSpace, 79 model classification CNS. 78 **DNLP**, 78 EMP, 78 LP, 78 MCP, 78 MINLP, 78 MIP, 78 MIQCP, 78 MPEC, 78 NLP, 78 QCP, 78 RMINLP, 78 **RMIP**, 78 RMIQCP, 78 RMPEC, 78 model generation, 169 model list, 169 model status error unknown, 98 integer infeasible, 98 intermediate noninteger, 98 intermediate infeasible, 98intermediate nonoptimal, 98 locally optimal, 98 optimal, 98 unbounded, 98 mpec GAMS call parameter, 189 MPEC, model type, 16, 78 multipass GAMS call parameter, 189 multiple solves, 85na, extended range value, 32, 63 ncpCM, function, 55 ncpF, function, 55 ncpVUpow, function, 55 ncpVUsin, function, 55 ne, relational operator, 107 negBinomial, distribution, 276, 278 nlp GAMS call parameter, 190 GAMS option, 244 NLP, model type, 15, 78 nodlim GAMS call parameter, 190 nonbasic, 169 nonlinear equations, 95 programming, 67, 78 nonlinear nonzero, 169 nonoptimal, 169

nonsmooth, 169 nonzero element, 169 nopt, solution marker, 102 normal completion (a solver status), 99 normal, distribution, 275, 277 normal, function, 55 not, relational operator, 119 number of rows and columns in display, 133 objective row (or function), 169 objective value, 170 objective variable, 170 on/offDelim dollar control option, 222 on/offDigit dollar control option, 223 on/offDollar dollar control option, 223 on/offDotL dollar control option, 223 on/offEcho dollar control option, 223 on/offEmbedded dollar control option, 224 on/offEmpty dollar control option, 224on/offEnd dollar control option, 225 on/offEolCom dollar control option, 225 on/offEps dollar control option, 225 on/offExpand dollar control option, 226 on/offGlobal dollar control option, 226 on/offInclude dollar control option, 226 on/offInline dollar control option, 227 on/offListing dollar control option, 227 on/offLocal dollar control option, 227 on/offLog dollar control option, 227 on/offMacro dollar control option, 228 on/offMargin dollar control option, 228 on/offMulti dollar control option, 228 on/offNestCom dollar control option, 229 on/offPut

dollar control option, 229 on/offRecurse dollar control option, 229 on/offSymList dollar control option, 229 on/offSymXRef dollar control option, 230 on/offText dollar control option, 230on/offUElList dollar control option, 231 on/offUElXRef dollar control option, 231 on/offUNDF dollar control option, 231 on/offVerbatim dollar control option, 231 on/offWarning dollar control option, 232 on115 GAMS call parameter, 190 optca GAMS call parameter, 190 GAMS option, 244 optcr GAMS call parameter, 190 GAMS option, 244 optdir GAMS call parameter, 190 optfile GAMS call parameter, 190 optimal, 170 option, 170introduction, 241 syntax, 241 or, relational operator, 119 ORANI, example from GAMSLIB, 38 ordered set, 170 card operator, 125circular lag and lead operator, 127 introduction, 123 lags and leads in assignments, 125 lags and leads in equations, 127linear lag and lead operator, 126 ord operator, 124 output, 170 GAMS call parameter, 191 output file, 170 pagecontr GAMS call parameter, 191 pagesize GAMS call parameter, 191 pagewidth GAMS call parameter, 191 parameter, 170

examples, 44 higher dimensions, 45 statement, 44 svntax, 44 pareto, distribution, 275, 277 parmfile GAMS call parameter, 191 phantom dollar control option, 233 pi, function, 55, 280 plicense GAMS call parameter, 192 poisson, distribution, 276, 278 poly, function, 55 power, function, 54, 55 precision, fixed, 102prefixPath dollar control option, 233 priorities for branching example, 162 introduction, 162 problem type, 78, 170 procdir GAMS call parameter, 192 prod, operator, 54, 92 PRODSCH, example from GAMSLIB, 71 profile GAMS call parameter, 192 GAMS option, 244 profilefile GAMS call parameter, 193 profiletol GAMS call parameter, 193 GAMS option, 244 program, 170 PROLOG, example from GAMSLIB, 77, 85 protect dollar control option, 233 purge dollar control option, 233 put additional numeric control, 147 appending to a file, 140assigning files, 139 closing a file, 140cursor control, 148 database/database application, 151 defining files, 139 errors, 150 example, 138, 147 exception handling, 150global item formatting, 145 introduction, 137 local item formatting, 146 numeric items, 145 output items, 143

page format, 140 page sections, 141 paging, 142 paging control, 150 positioning the cursor on a page, 142 set value items, 145 syntax, 137 system suffices, 143 text items, 144 put current cursor control .cc, 148 .cr, 149 .hdcr. 149 .tlcr, 149 put cursor control .hdcc, 148 .tlcc, 148 put local item formatting, 146 put paging control .lp, 150 .ws, 150 putdir GAMS call parameter, 193 pwpFunc, function, 274 qcp GAMS call parameter, 193 GAMS option, 244 QCP, model type, 15, 78 quoted labels, 31 names of sets, 36text, 31 quotes, 31, 36

RAMSEY, example from GAMSLIB, 65, 74 randBinomial, function, 55 randLinear, function, 55 randTriangle, function, 55 range of numbers, 32rayleigh, distribution, 275 ref, reference type, 92 reference GAMS call parameter, 193 rel_eq, function, 58 rel_ge, function, 58 rel_gt, function, 58 rel_le, function, 58 rel_lt, function, 58 rel_ne, function, 58 relational operator, 170 remark dollar control option, 234 report summary, 102reporting, format, 104 reserved words, 30

reslim GAMS call parameter, 193 GAMS option, 244 resource interrupt, 99 restart GAMS call parameter, 193 right hand side, 170 rminlp GAMS call parameter, 194 GAMS option, 244 RMINLP, model type, 16, 78 rmip GAMS call parameter, 194 GAMS option, 244 RMIP, model type, 15, 78 rmiqcp GAMS call parameter, 194 RMIQCP, model type, 16, 78 rmpec GAMS call parameter, 194 RMPEC, model type, 78 round, function, 55 rPower, function, 55 rules constructing tables, 46formating tables, 46 save GAMS call parameter, 194 savepoint GAMS call parameter, 194 scalar, 170 equation, 73 example, 44statement, 43 syntax, 43 scale option, 162scaling models, 162of a variable, 163

option, 162 scaling models, 162 of a variable , 163 of an equation , 163 of derivate, 164 scenario analysis, 85 scrdir GAMS call parameter, 194 scrext GAMS call parameter, 194 scriptexit GAMS call parameter, 194 scriptfrst GAMS call parameter, 195 scriptnext GAMS call parameter, 195 scriptnext GAMS call parameter, 195

Secure Work Files, 253 seed GAMS call parameter, 195 GAMS option, 244 semi-continuous variables Definition, 161 Example, 161 semi-integer variables definition, 161 example, 161 semicolon, 31 set, 170associated text, 36declaration for multiple sets, 37 definition, 35 dollar control option, 234 dynamic, 117 elements, 36 multi-dimensional, 39 multi-dimensional many to many, 40 multi-dimensional one-to-one mapping, 39 names, 35 sequences as set elements, 37 simple, 35syntax, 35 set operations complement, 121 difference, 121 intersection. 121 union, 121 setArgs dollar control option, 235 setComps dollar control option, 235 setDDList dollar control option, 235 setEnv dollar control option, 236 setGlobal dollar control option, 236 setLocal dollar control option, 236 setMode, function, 280 setNames dollar control option, 236 SHALE, example from GAMSLIB, 36 shift dollar control option, 237 show dollar control option, 237sigmoid, function, 55 sign, function, 55, 74 signed number, 43, 44, 46 signPower, function, 55 simple assignment, 51 simplex method, 170

sin, function, 55, 74 sine, function, 280single dollar control option, 238 \sinh , function, 55 slack, 170 slack variable, 170 slash, delimiter, 32, 138 sleep, function, 59 slexp, function, 55sllog10, function, 55 slrec, function, 55 smax, operator, 54, 55 smin, operator, 54, 55 smooth, 170 functions, 74 solprint GAMS call parameter, 195 GAMS option, 244 solslack GAMS option, 245 solution listing, 101 solve errors, 105 errors messages, 104 statement, 77 solve statement actions triggered by, 84 requirements. 84 several in a program, 85several models, 85 syntax, 83 solve summary, 97 evaluation errors, 98 iteration count, 98 objective summary, 98 resource usage, 98 solver status, 98solvelink GAMS call parameter, 195 solveopt GAMS option, 245 solver, 170 solver status, 98evaluation error limit. 99 iteration interrupt, 99 normal completion, 99 other errors, 99 resource interrupt, 99 terminated by solver, 99 unknown error, 99 solvercntr GAMS call parameter, 195 solverdict GAMS call parameter, 195 solverinst

GAMS call parameter, 195 solvermatr GAMS call parameter, 195 solversolu GAMS call parameter, 195 solverstat GAMS call parameter, 196 special languages features, 159 special ordered sets introduction, 159 type 1 - example, 160type 1 - definition, 159type 2 - definition, 160 sqexp, function, 55sqlog10, function, 55 sqr, function, 55sqrec, function, 55 sqrt, function, 55 stars dollar control option, 238 statements, 170 static set, 118, 170 stepsum GAMS call parameter, 196 stitle dollar control option, 238 stop dollar control option, 238 stringchk GAMS call parameter, 196 studentT, distribution, 275, 277 subsets, 52 subsys GAMS call parameter, 196 suffix field width, 146 file, 145 numerical display control, 147 page control, 150 put-file, 140 system, 143 variable, 67, 75 sum, operator, 54superbasic, 102, 170 variable. 102 suppress GAMS call parameter, 196 symbol, 170 GAMS call parameter, 197 symbol reference map assigned, 92 control, 92 declared, 92 defined, 92 equ. 92 impl-asn, 92

model, 92param, 92 ref, 92set, 92 var, 92 sys11 GAMS call parameter, 197 sysdir GAMS call parameter, 197 sysincdir GAMS call parameter, 197 sysInclude dollar control option, 238 sysout, 101GAMS call parameter, 197 GAMS option, 245

tabin

GAMS call parameter, 197 table, 170 a statement, 45 condensing, 47 continued, 46example, 46long row labels, 48 more than two dimensions, 47statement, 45, 46 syntax, 45 tan. function. 55 tanh, function, 55 terminate dollar control option, 239 terminated by solver, 99 text, 170 tformat GAMS call parameter, 198 The GAMS Grid Computing Facility, 263 threads GAMS call parameter, 198 timeClose, function, 59 timeComp, function, 59 timeElapsed, function, 59 timeExec, function, 59 timer GAMS call parameter, 198 timeStart, function, 59 title dollar control option, 239 trace GAMS call parameter, 198 traceopt GAMS call parameter, 198 triangular, distribution, 275, 277 trunc, function, 55 type, 171 of discrete variables, 159

unbounded, 98, 102 undf, extended range value, 63 uniform, distribution, 275, 277 uniform, function, 55 uniformInt, distribution, 276 uniformInt, function, 55 union, of sets, 121 unique element, 171 UNIX Installation Notes, 282 unknown error, 99 unload dollar control option, 239 use205 dollar control option, 239 use225 dollar control option, 240 use999 dollar control option, 240user1 to user5 GAMS call parameter, 198 using, 84 variable binary, 66 free, 66 integer, 66 negative, 66 positive, 66 statement, 65, 66 styles for declaration, 66 suffix. 67 syntax of declaration, 65types, 66 variable attributes activity level (.1), 67branching priority value (.prior), 67 fixed value (.fx), 67 lower bound (.lo), 67marginal or dual value (.m), 67 scale value (.scale), 67 upper bound (.up), 67variable bounds activity level, 68 fixing, 67 variable type, 171 vcPower, function, 55 vector, 171version dollar control option, 240warning dollar control option, 240 warnings GAMS call parameter, 198 weibull, distribution, 275, 277 while

example, 156statement, 156 syntax, 156Windows, 281workdir GAMS call parameter, 198workfactor GAMS call parameter, 198workspace GAMS call parameter, 199xor, relational operator, 119xsave GAMS call parameter, 199zero default, 171zerores GAMS call parameter, 199 zeroresrep GAMS call parameter, 199

ZLOOF, example from GAMSLIB, 117