Ann's Copy
Don't Steal

MESA

Molecular Electronic Structure Applications

by

Paul Saxe,[a] Richard Martin,[a] Michael Page[b], and Byron H. Lengsfield III[c]

User's Manual

[a]Los Alamos National Laboratory
Los Alamos, New Mexico

[b]Naval Research Laboratory
Washington, D. C.

[c]Lawrence Livermore National Laboratory
Livermore, California

TABLE OF CONTENTS

		PAGE
I.	Introduction	2
II.	MESA Quick User's Guide	3
III.	Program Structure	16
IV.	Geometry Input	19
V.	Basis Set Input	24
VI.	Integral Program	26
VII.	Guess Molecular Orbitals	27
VIII.	SCF Program	29
IX.	IVO Program	30
X.	CI Programs	32
XI.	MCSCF Program	39
XII.	Molecular Properties	43
XIII.	Geometry Optimization	45
XIV.	Force Constants	47
XV.	MESA IO Library	48
XVI.	Print Options	53
XVII.	Links: One Line Description of the Programs	58
XVIII.	Sample Inputs	61

GUIDE: MESA User's Guide

Summary:

A quick user's guide to MESA

Program:

Defaults:

Description:

KEYWORD = *$ROUTE*

Option	Key	Default	Example
charge	q	0	q=0
multiplicity	2s+1	1	2s+1=1

Charge and Multiplicity are used by the SCF program to determine the number of alpha and beta electrons as well as the default coupling coefficients used to build the Fock matrices.

geometry	geom	Z-Matrix Angstroms degrees	geom=(coord,inau)

(suboptions)	*(default)*	*(explanation)*
coord	Z-Matrix	cartesian coordinates will be input
inau	Angstroms	distances will be given in atomic units
inrad	degrees	angles will be given in radians

basis sets	basis	dz	basis=(dzp)

Basis sets may be specified either in the $Route input or in the $Geom section. basis is used when the same basis is employed on all atoms

integrals	noints	false	noints

If noints is included in the $Route input , the integral evaluation and ordering steps are skipped. This option is used when the integral file has been saved from an earlier calculation

Option	Key	Default	Example
Starting Orbitals	guess	guess=core	guess=chk=mcscf
(suboptions)		*(explanation)*	
core		diagonalize the core hamiltonian	
chk		the orbitals are read from the CHK file (def. CHK=SCF)	
chk=mcscf			
chk=unconverged-MCSCF			
chk=no		CI Natural Orbitals	
chk=no=2		CI Natural Orbitals of Root 2	
rwf		SCF orbitals are read from the RWF file	
rdinp		orbitals are read from the input file. The format is: $Vectors label vector 1 label vector 2 : $End a free formatted read is used to process the vectors	
nocc		the number of orbitals that the RDINP section will read.(def=SCF occupied orbitals)	
rdchk		pick up SCF orbtials from the CHK file which were generated in a different AO basis and generate a starting guess in the current AO basis	
merge=(ind=(s_1,b_1,e_1,..., s_n,b_n,e_n))		The merge option is used to merge two sets of molecular orbitals. The ind vector is a set of triplets, the first entry, s_i, denotes the vector set, the second entry denotes the starting orbital and the final entry denotes the ending orbital. Thus the first e_1-b_1+1 orbitals come from set s_1 and so forth. The sets are defined by including SET1=set and SET2=set input lines in the $Route input.	

expand

The molecular orbitals from a smaller basis set calculation are used to construct a set of orbitals in a larger basis. Unit vectors are added to the end of the starting orbitals and then Schmidt orthogonalized to the preceding orbitals. The augmented set of orbitals are then rotated under a one-electron operator to produce a set of symmetry orbitals.

huckel

A huckel calculation is used to generate the starting orbitals.

alter

The alter option is used to reorder the starting orbitals. If the alter option is included in the $Route input, the input file is searched for a $ALTER input line. The $ALTER input consists of pairs of vectors to be flipped. The input

```
$Alter
 2,4
 5,7
$End
```

would reorder orbitals 2 and 4, as well as orbitals 5 and 7.

Option	Key	Default	Example
Wave Function	hf	scf	hf=tcscf

If hf is specified then a closed-shell or high-spin open-shell scf calculation is performed as dictated by 2s+1, the multiplicity. hf=tcscf performs a two-configuration scf calculation

Option	Key	Default	Example
wave-function	scf	Page-McIver	SCF=(Pulay,cycles=40)

if hf is specified, the hf input is read from the SCF keyword.

(suboptions)	*(default)*	*(explanation)*
Pulay	false	Pulay DIIS algorithm is used
Convergence	6	SCF convergence is $10^{-\text{convergence}}$
Cycles	30	Maximum number of SCF iterations
GSCF	false	

In the general SCF program the Fock operator has the following form:

$$\mathcal{F}_i = f_i \mathcal{h} + \sum_j^{\text{shells}} a_{ij} \mathcal{J}_j + b_{ij} \mathcal{K}_j$$

where $\mathfrak{h}$ is the sum of one-electron kinetic and potential energy operators and $\mathcal{J}_j$ and $\mathcal{K}_j$ are the coulomb and exchange operators respectively. The gscf energy expression is then:

$$E = \sum_{i=1}^{\text{nshells}} \sum_{j=1}^{\text{orbitals}(i)} \langle \varphi_j | f_i \mathfrak{h} + \mathcal{F}_i | \varphi_j \rangle + NR$$

where φ_j is an orbital in shell i and NR is the nuclear repulsion energy. The vector coupling coefficients, f_i, a_{ij} and b_{ij}, can be input in two ways.

The may be input directly

$f = (f_1, f_2 \cdots, f_{nshells})$

alpha = $(a_{11}, a_{21}, a_{22}, a_{31}, \cdots, a_{nshells,nshells})$

beta = $(b_{11}, b_{21}, b_{22}, b_{31}, \cdots, b_{nshells,nshells})$

where alpha and beta are lower triangular arrays. These arrays may also be defined by dividing two sets of numbers

$f_i = fn_i/fd_i$

$a_{ij} = an_{ij}/ad_{ij}$ and $b_{ij} = bn_{ij}/bd_{ij}$

and the numerators and denominators are input as follows

$fn = (fn_1, fn_2, \cdots, fn_{nshells})$

$fd = (fd_1, fd_2, \cdots, fd_{nshells})$

$an = (an_{11}, an_{21}, an_{22}, \cdots, an_{nshells,nshells})$

$ad = (ad_{11}, \cdots \qquad)$

$bn = (bn_{11}, \cdots \qquad)$

$bd = (bd_{11}, \cdots \qquad)$

We must also input the number of shells in the calculation and the number of orbitals in each shell.

nshells = n

shlnbf=$(n_1, n_2, \cdots, n_n)$

<u>Example</u> Open shell singlet with 7 doubly occupied orbitals

```
SCF = (GSCF, nshells=3,
       shlnbf = (7,1,1),
       f = (1,.5,.5),
       alpha = (2,1.,0.,1.,.5,0.), beta = (-1,-.5.,0.,-.5,.5,0.))
```

Option	Key	Default	Example
Wave Function	mcscf	general mcscf	mcscf=(cas)

both the type of MCSCF wave function and the input options are specified by this keyword. If "CAS" is not specified then a general MCSCF wavefunction is assumed and the user must specify the invariant-orbital sets if any are present. For every MCSCF run the number of core orbitals and a drt (or a drtmc entry in a multireference CI calculation) must be provided.

(suboptions)	*(default)*	*(explanation)*
cas	false	the drt input defines a full-CI calculation in the active orbital space
number-of-core-orbitals	none	the number of core or doubly occupied orbitals in the calculation. This input must be specified if core orbitals are present.
convergence	12	the square of the change in the orbitals which is approximately the convergence of the energy
cycles	10	the number of MCSCF iterations
invariant-orbital-sets=(.,.,.)	none	this is a vector whose length is the number of active orbitals. The MCSCF energy is invariant with respect to rotation of orbitals with the same label. For example, `invariant-orbitals-sets=(1,1,2,2,2,2)` would be the proper input for six active orbitals where the CI expansion was generated by all single and double excitations from two doubly occupied orbitals into a virtual orbital space of four orbitals.
nonautomatic	false	use the method vector to determine if a first-order or second-order algorithm will be used on each iteration
method=(.,.,.)	none	`method=(1,1,..,2,2,..)` an entry of 1 directs the program to use a first-order algorithm and a 2 will invoke a second-order step. One entry is needed for each iteration
noabort	false	if the MCSCF does not converge, do not abort the job and save the unconverged orbitals. The next run of the MCSCF will use these orbitals if `guess=chk=unconverged-mcscf` is specified

linear-equation=cycles	30	maximum number of iterations allowed in the solution to the Second-Order Linear Equations
linear-equation=convergence	1.e-4	convergence criteria in Second-Order Linear Equations
ci=nroots ci=tolerance ci=cycles	1 7 40	input parameters for CI step see option CI for more details

Option	Key	Default	Example
DRT	drt	----	DRT=(vexcite=1,NSYM=2)

(suboptions)	*(default)*	*(explanation)*
NRefs	1	The number of reference configurations
NSym	1	The dimension of the symmetry group. The symmetry of the state is determined from the symmetry of the first reference CSF.
NBF	the number of atomic orbitals in the basis set	The number of basis functions (MOs) in the CI calculation
Ref=SymLimit	false	Limit the symmetry of the reference CSFs to the symmetry of the final state
NoInt	---	Turn off the interacting space option false if NRefs=1 true if NRefs > 1
Vexcite	2	The maximum number of excitations from the reference CSFs allowed
RefExcite	0	The maximum number of excitations from the reference CSF in the Valence space. Valence orbitals are labeled with the % key. Only valence electrons are excited. This option is used to define a space of reference CSFs as in a First or Second Order CI.
No34X	false	This flag tells the program only single excitations into the virtual space are present.
BlkSize	100000	This blocksize used in writing GUGA integrals to disk
NGroups	none	This option is used in M802 to define the number of orbital groups. If this option is present, the code searches for a $Groups input string on the input file and uses the "Group" orbital codes in the $DRT input.

NOpen = Max	unlimited	Restrict the number of open shell electrons allowed in the problem.
Nopen=Min	0	

Option	Key	Default	Example
CI	ci	----	CI=(nroots=2, cycles=15)

(suboptions)	*(default)*	*(explanation)*
NRoots	1	The number of eigenvalues and eigenvectors to be determined
Cycles	15	The number of cycles allowed for each root in the Davidson diagonalization
Tolerance	5	Convergence criteria on the norm of the error vector, (H-E)C. The criteria is $10^{-tolerance}$
NGuess	NRoots	The number of initial guess vectors
NRoots-At-A-Time	NRoots	The number of eigenvectors to be determined during each pass
Threshold	5	The threshold for discarding a vector during a multiroot calculation. The threshold $10^{-threshold}$ is the criteria used to decide if an expansion vector is to be retained.

Option	Key	Default	Example
Properties	properties	don't compute	Properties=e1

(suboptions)	*(explanation)*
V0	Electrostatic potential at the nuclei
V1	Electric field of the nuclei
E0	Electric monopole term
E1	Electric dipole term
E2	Electric quadrupole term
E3	Electric octupole term
E4	Electric hexadecapole term
Transition=(i,j,k)	Transition properties between roots i and j, i and k as well as j and k.
Transition=(1,*)	Transition properties between the ground state and all available excited states.

Option	Key	Default	Example
Symmetry	Sym	Sym=On	Sym=Off

Sym=Off must be specified in a geometry optimization or a finite-difference force constant calculation. In almost all applications Sym=Off should be included in the $Route input..

Option	Key	Default	Example
optimization	opt	none	opt=berny

(suboptions)	*(default)*	*(explanation)*
berny		Berny Schlegel optimization method
ms		Murtagh-Sargent optimization method
fp		Fletcher-Powell optimization method
ts		transition state search
negeig		number of negative eigenvalues characterizing the stable point
convf	3.0e-4	convergence criteria on gradient
dxmaxt	2.0e-1	maximum stepsize
fmaxt	1.0	maximum allowable gradient element
srcd2e	berny	srcd2e=rdchk read force constants from the checkpoint file
force		Analytic force constants
force-constants		Finite difference force constants

Option	Key	Default	Example
Frequencies	frequencies	false	frequencies

If this option is present then harmonic frequencies are computed

Option	Key	Default	Example
Force Constants	force-constants		force-constants=cartesian

(suboptions)	*(default)*	*(explanantion)*
two-point	one-point	two-point derivative expression
cartesian	internal	use cartesian coordinates for finite-difference calculations

Option	Key	Default	Example
IVO	ivo	false	ivo=(singlet,homo=4)

(suboptions)	*(default)*	*(explanantion)*
homo	SCF HOMO	Redefines the open-shell core orbital
singlet	triplet	Singlet couple the open-shell electrons
CHK	RWF	Read the orbitals from the CHK file

KEYWORD = *$DRT*

The Reference Configurations used in the DRT codes follow the $DRT input string. There are two formats that are employed. In a multireference CI calculation link M801 must currently be used to generate the DRT. In M801 a reference configuration is of the form

```
$DRT
        2COR1  2DOC1  1ALP1  6UOC1  2VIR1
               3DOC1         4UOC2
$End
```

All of the orbitals appear in the first reference configuration. All of the orbital codes used to define a single reference need not be input on the same line.

Orbital codes with the following structure

(repeat count) (key) code symmetry

are used to build the reference configurations. The "repeat count" and "key" parameters are optional. The following orbital codes are available

	CODE	EXPLANATION
1)	fzc	frozen core
2)	fzv	frozen virtual
3)	cor	restricted core
4)	vir	restricted virtual
5)	doc	doubly occupied
6)	uoc	virtual
7)	alp	alpha occupancy (couple up)
8)	bet	beta occupancy (couple down)

keys:

% labels the active orbital space for First Order, Second Order and POL CI calculations

/ labels active orbitals when one is inputting a list reference configurations. These orbitals have different codes associated with them in some of the reference configurations.

If more than one reference configuration is being employed, only the active orbitals with the slash, "/", key are defined in the additional references.

If link M802 is being used, which is always the case in an MCSCF calculation, and "ngroups" is specified in the drt input line in $Route, the following format is used to define the reference configuration;

(repeat count) "typ" number ";" symmetry

for example

```
$DRT
        2typ1  4typ2;2  6typ3;1
        na=2   n6=1     ns=1
$End
```

where

$2 \cdot na + nb$	=	number of electrons
$nb + 1$	=	spin multiplicity
ns	=	symmetry of the wavefunction

Only one reference is defined in the $DRT section if orbital groups are being used. The distribution of the electrons among the orbital groups are defined in the $Groups input which is described in the next section.

KEYWORD = *$Groups*

The $Groups input is used to distribute electrons among the orbital groups. The "nrefs" option of the drt input line in $Route determines the number of distributions that will be processed. A distribution is defined by a "numel" vector, where each element of the vector determines the number of electrons in that group.

```
$Groups
        numel = (3,2,1)   spin = (1,1,0)
$End
```

Each entry to the "spin" vector is $2S_i$ (where the multiplicity would be $2S_i$ +1) and this vector defines a genealogical coupling scheme. In this example the first three electrons would be coupled to a doublet, the next two electrons would be coupled to the preceding three electrons to produce a doublet and the final electron is coupled to this doublet to produce a singled.

KEYWORD = *$Geom*

MESA supports cartesian and Z-Matrix geometry input. In addition, ghost centers (centers with a basis set and zero nuclear charge), dummy centers (centers with neither a basis set or a nuclear charge) and point charges are supported. A detailed input description is given in the Geometry section of this manual.

Cartesian Input Format:

```
$Geom
        atomic-symbol (number) [Basis=basis-name]  X Y Z
        ⋮
        atomic-symbol (number) [Basis=basis-name]  X Y Z
$End
```

Both the "number" associated with an "atomic-symbol" and the basis set specification are optional. Coordinates can be specified in Angstroms, the default, or in atomic units. There can be no blank lines in the cartesian input.

Example:

```
$Route
        basis=dzp
        geom=(coord,inau)
$End
$Geom
O                   0.0    0.0    0.0
H1(basis=tz)        0.0    1.0   -1.1
H2(basis=tz)        0.0    1.0    1.1
$End
```

The oxygén basis is defaulted to DZP.

Z-Matrix Input Format:

$Geom

as_1

$as_2 \quad as_1 \quad r_{12}$

$as_3 \quad as_i \quad r_{3i} \quad a_{sj} \quad ang_{3ij}$

$as_4 \quad as_k \quad r_{4k} \quad as_\ell \quad ang_{4k\ell} \quad as_m \quad dih_{4k\ell m}$

$as_s \quad as_n \quad r_{sn} \quad as_o \quad ang_{5no} \quad as_p \quad dih_{5nop}$

$\vdots$

$r_{12} = 2.0$

$r_{3i} = 3.0$

$\vdots$

$r_{5n} = 1.0$

$ang_{3ij} = 120.$

$\vdots$

$ang_{5no} = 125.$

$dih_{4k\ell n} = 0.0$

$\vdots$

$dih_{6qrs} = 180.$

$End

where r_{ij} is a variable which represents the distance between centers i and j, ang_{ijk} is a variable which represents the angle between centers i, j and k , dih_{mnop} is a variable which represents

the dihedral angle formed by centers m, n, o and p. The symbol as_i denotes an atomic symbol of the form

atomic-symbol (number) [Basis=basis-name].

The "basis-name" option is only input in the first column of the Z-Matrix input. Once the symbolic Z-Matrix is built, then the variables are defined after a blank line.

<u>Example</u>: H_2O_2

```
$Geom
      H1
      O1  H1  r1
      O2  O1  r2  H1  A1
      H2  O2  r1  O1  A2  H1  D1

      r1 =.95
      r2 = 1.2
      A1 = 107.
      A2 = 107.
      D1 = 30.0
$End
```

The OH bonds are constrained to be equal in this example but the OOH angles may differ in a geometry optimization calculation.

PROGRAM STRUCTURE:

MESA consists of a number of independent program units which communicate primarily through a master file called the read-write file or RWF. The first step in calculation is the generation of a Route or list of programs that will be executed. MESA first scans the input deck for $Route and then interprets the character strings following this keyword and builds a Route. MESA does not recognize the case of any of the characters in the $Route input string.

A Route has the following appearance

1//1,2;
2//2;
3//2,12,30;
4//1;
5//1;
19//51;
20//1;

which is the Route generated for a SCF calculation. The programs M0, M1, M101, M102, M202, M302, M312, M330, M401, M501, M1951 and M2001 are executed in sequence. These programs are not overlaid, after one program successfully ends it invokes the next program in the Route. On UNICOS and CTSS/LTSS systems this process is controlled by a script. The program which has just completed writes the name of the next program in the Route to the standard output "device". The script picks up this name and invokes the next program. In a geometry optimization, statements are included in the Route to define a loop structure.

The information contained in the $Route input string is condensed and assembled in a character string of the form

" A / B / C / D "

where A determines if a single point calculation or geometry optimization is to be performed, B determines the calculation used to obtain the molecular orbital, C defines the type of CI, if any, that will be performed and D determines if molecular properties are going to be completed. A single reference CI calculation which included the evaluation of molecular properties would be invoked by the following character string,

" SP / HF / CI / FULL ".

Once the program assembles this string a data file, MESADAT, is searched for this string and the Route is read from this data file. Standard basis sets and the list of the data files that will be saved at the completion of the calculation are contained in this file. This is a formatted file, so new Routes or basis sets can be added at any time.

If the Route is not in MESADAT then a "Nonstandard" Route can be run by including a separate keyword, $NONSTD, followed by the Route in the input. Several examples of calculations which employ "Nonstandard" Routes have been included in the Sample Input section of this manual. All of the sample IVO calculations currently use "Nonstandard" Routes.

As the programs specified in the Route are invoked they may read input from either the input option included in the $Route character string or they may search the input for other keywords, such as $GEOM which preceeds Z-Matrix or cartesian geometry input, and read the data which follows these keywords. The data following keywords other than $Route can be modified at any time during a calculation. The data contained in the $Route character string is read by the initialization program which determines the Routine and stores this data on the RWF file. In order to change input data in $Route a utility program, M103, is run . M103 reads the data in $Route and writes it to the RWF file.

All of the programs in MESA make use of a standard set of IO and math routines contained in a library, MESALIB. MESALIB is broken down into four subunits, an IO library, an utility library, a linear algebra library and a set of machine dependent library routines. All unformatted files used by MESA are word addressable and blocks of data are stored and retrieved by simply specifying the character string associated with that data. Thus, the MCSCF molecular orbitals are read from a file by calling an IO routine with the character string "MCSCF VECTORS". The IO library routines are described in detail in the IO section of this manual. The utility library contains standard matrix multiplication, printing and character manipulation routines. The linear algebra library contains fortran routines to solve linear equations and diagonalize matrices. The machine dependent library is designed to hold memory allocation and low-level IO routines which are system dependent.

In the current version of MESA there are six principle files. There are formatted input and output files. There is a standard data file, MESADAT, and there are three word addressable library files, RWF, CHK and INT. The RWF file was described earlier so we only need to discuss the INT file, which contains the two-electron integrals, and the CHK file, which contains the information which is saved at the end of a calculation. The CHK file is also read by programs, such as M401, to obtain a guess set of molecular orbitals or other information, such as force constants, saved from earlier calculations. Currently, raw, ordered and transformed integrals are stored on the INT file. This is burdensome for calculations employing large basis sets and revisions of the MESA code are available which store the different integral files on their own units.

At the end of a calculation program M2001 copies the information that is to be retained from the RWF to the CHK file and then prints out timing information.

MESA has error handling routines which provide some traceback information on most operating systems. When these error routines are called the binary files controlled by the IO library are closed and the calculation can be restarted with these files. If MESA abnormally terminates with a system error then these files may have been corrupted and MESA will not allow you to continue with these files. Scripts are available which will save the RWF and INT files so earlier steps in the calculation need not be repeated.

GEOM: Geometry Input Code

Summary:

M101 generates a molecular geometry from either Z-Matrix or coordinate input

Program: M101

Defaults:

See discussion below

Description:

The $Geom section is read by M101 and is always required, except in the case of a restart, a Geom=RdChk or Geom=CHK directive. M101 reads the molecular geometry and basis sets. The input is free-field: the several items on a card may be separated by one or more blanks. The first set set of cards specifies the nuclei. There are two formats which you may want to use at one time or another. They are discussed in turn below.

Z-Matrix Format:

Element(Basis=basis-name) N1 Length N2 Alpha N3 Beta J

The Z-Matrix format is the default in MESA. A series of cards, one for each center, is read in the format above.

Element specifies the chemical nature of the nucleus. It may consist of simply the chemical symbol, such as 'H' or 'Pt', or it may begin with the chemical symbol followed immediately by a numeric identifier, e.g. 'Pt4' to designate the fourth platinum atom. Ghost centers (centers with a basis but zero charge) are denoted by 'ZQ'. Dummy nuclei are denoted by the symbols 'X' or '-'. A symbol is required for every center. The remainder of this field is optional and specifies the basis set to be used on the center. If unspecified, the default basis from the route card is supplied. If you wish to specify the basis here, enter it's name (16 characters maximum) in parentheses. A list of the basis sets known to MESA is available in MESADAT.

N1 specifies the (previously defined) nucleus for which the internuclear distance from center N1 to center N will be given. This item may be either an integer (with N1<N) giving the number of the card defining the other center or the alphanumeric label of the other center.

Length is the internuclear distance R(N1,N). It may be either a positive floating point number in Angstroms(unless modified by the InAU directive in $Route) or an alphanumeric string of up to 16 characters. In the latter case, the distance is represented by a parameter, for

which a value will be supplied in a later section of $Geom. If a length is to be varied in an optimization run, it must be specified as a parameter. Note that a given parameter may be used to specify two distinct lengths, in which case they are constrained to be equal. The items N1 and Length are required for all nuclei after the first. For the second nucleus, only Element, N1, and Length are required. The nucleus is placed on the Z-axis.

N2 specifies the center for which the angle Alpha(N,N1,N2) is given. Again, it may be either an integer or an alphanumeric string which matches a previous Element entry. Note that N1 and N2 must represent different nuclei.

Alpha is the internuclear angle. This may be a floating point number giving the angle in degrees (unless modified by the InRad directive in $Route) or an alphanumeric string representing a parameter. The numerical value of Alpha must lie in the range 0<Alpha<180 degrees. N2 and Alpha are required for all centers after the second. For the third center, only Element,N1,Length,N2,and Alpha are required. This center is placed in the xz-plane.

N3 differs in meaning,as does Beta, depending on the value of the last item, J. If J=0 or is omitted, N3 specifies the center for which the internuclear dihedral angle Beta(N,N1,N2,N3) will be given. As with N1 and N2, this may be either an integer or an alphanumeric string matching a previous Element entry. This center must be distinct from those specified by N1 and N2.

Beta is the internuclear dihedral angle (if J=0,see below). This may be a floating point number giving the angle in degrees (unless modified by $Route), or a (signed) alphanumeric string representing a parameter. The dihedral angle is defined as the angle between the planes defined by (N,N1,N2) and (N1,N2,N3). Beta must lie in the range 0<Beta<180 degrees. The sign is positive if the movement of the directed vector N1-->N2 towards the directed vector N2-->N3 involves a right-handed screw motion.

J allows you some freedom over the specification of the dihedral angle. Although it is always possible to specify the center by a bond length, a bond angle and a dihedral angle (which is what you must do if J=0 or is absent), it is sometimes more convenient to use a bond length and two bond angles. This is called for by using J=1. In this case, the internuclear angle Beta(N,N1,N3) is specified. As before, this may be either a floating point value in degrees or an alphanumeric string representing a parameter. In the event that you specify two bond angles, there will be two possible positions for the center N. This is fixed by the sign of J. Thus J=+1 if the triple vector product N1-N(N1-N2 X N1-N3) is positive and J=-1 if the product is negative.

If you have not specified any parameters, this completes the $Geom section. If you have then you must specify one last set of information, the initial values of the parameters.

This set of data must be separated from the Z-Matrix by a blank card, after which one card for each parameter is read in the form:

Parameter=Value D2E=Value Type

Parameter is the alphanumeric string defined by the preceeding section and Value is a floating point number representing the initial value (in Angstroms and degrees) unless modified by $Route directives.

D2E is an optional field. If specified, it denotes how MESA obtains initial guesses at the force constants to be used in the optimization. If omitted, internally stored guesses are provided. The D2E value may be a floating point number, which then specifies the initial diagonal second derivative associated with this parameter(in Hartrees/sq-bohr, or Hartrees/sq-radian). The D2E specification is ignored for Opt=MS and Opt=FP runs.

Type denotes the status of the parameter. It can be either 'constant', in which case the parameter is not varied in an optimization, or 'variable', the default type. This is included for convenience only. A 'constant' could have been just as easily entered directly in the Z-Matrix as a floating point number.

Coord format:

Element(Basis=basis-name) X Y Z

This case is specified by the Coord directive in $Route. The input section consists of a number of cards, one for each center, which specify the element, basis set, and cartesian coordinates of the element.

Element is either a string (as above), or an integer specifying the atomic number of the center.

X is a floating point number giving the x-coordinate of the center in Angstroms (unless modified by a directive in $Route).
Y is a floating point number giving the y-coordinate of the center in Angstroms (unless modified by a directive in $Route).
Z is a floating point number giving the z-coordinate of the center in Angstroms (unless modified by a directive in $Route).

Example:

```
$Geom
o
h1  o  0.96
h2  o  0.96     h1 104.5
$End
```

Explanation: This specifies the geometry for the water molecule. The bond distance is 0.96 Angstroms and the angle is 104.5 degrees.

Example:

```
$Geom
o(basis=6-31g)
h1(basis=sto-3g)  1  roh
h2(basis=sto-3g)  1  roh     h1 theta

roh=0.96     constant
theta=104.5
$End
```

Explanation: This specifies a geometry for the water cation. The hydrogens are 0.96 Angstroms from the oxygen and the distance will not be varied in the optimization. The bond angle, theta, is initially 104.5 degrees, and will be optimized. A double zeta basis (6-31g) will be used on the oxygen, the hydrogens will use a minimal (sto-3g) basis.

Example:

```
$Geom
o   0.0     0.0   0.0
h   0.0     0.0  -0.96
h   0.0     0.0   0.96
$End
```

Explanation: This specifies yet another geometry for the water molecule. The hydrogens are 0.96 Angstroms from the oxygen along the z-axis, and the bond angle is 180.0 degrees.

```
$Geom
N
C N RCN
X C 1.0      N Half
O C RCO      X Half      N 180.0
H O ROH      C COH       X 0.0

RCN=1.20
RCO=1.3
ROH=1.0
COH=105.0
$End
```

Explanation: In this section, Half represents half of the NCO angle which is expected to be nearly linear. Note that a value of Half less the 90 degrees corresponds to a cis arrangement.

BASIS: Basis Set Input

Summary:

Input description of the program which determines the gaussian basis sets used in a MESA run.

Program: M102

Defaults:

Double Zeta basis

Description:

The gaussian basis employed in a calculation can be specified by the "BASIS" input line in the $Route input string,

```
$Route
        basis=dzp
$End
```

or a different basis set can be used on each atom by including a "BASIS=basis-name" line the $GEOM input .

```
$Geom
o(basis=tzp)
h1(basis=dzp) o r1
h2(basis=dzp) o r2 h1 angle

r1=.9
r2=.9
angle=102.
$End
```

Once the basis set is specified, MESA searches the input file for an input line of the form

```
$basis-name  atomic-symbol
```

where the "atomic-symbol" matches the atomic symbol used in the $GEOM input. Note that the "atomic-symbol" does not include the "atom-number" used to distinquish like atoms. Thus in this example the code would search the input file for the string "$dzp h" to obtain the hydrogen basis. If this string is not found in the input file then the data file, MESADAT, is searched.

Once this string is located, a free-format input statement is used to read the orbital exponents and contraction coefficients. The basis set input has the following structure,

```
$dzp h
/arbitary hydrogen basis
type=s
1000.        .05
500.         .86
120.         .68
type=s
.5           1.0
type=p
.75          1.0
$end
```

The slash character, "/", is used for comment lines.

Ghosts centers (a center with a basis but no charge) are specified by using "zq" as the atomic symbol.

A number of standard basis sets are included in MESADAT as well as Effective Core Potentials (ECPs) and the basis sets that were designed to be used with the ECPs.

INTS: Integral Programs

Summary:

MESA employs a suite of gaussian integral programs based on the Rhys polynomial method. First and second derivatives of the integrals with respect to nuclear coordinates are also determined by these codes.

Program: M302,M312,M330,M303,M313,M323,M333,M702,M712,M730

Defaults:

Description:

The integral codes require no input other than the specification of the basis set and this input is described elsewhere. There are several versions of the derivative integral programs. In some versions the primitive derivative integrals are traced with density matrix elements and in others the derivative integrals over contracted basis functions are constructed and written to disk.

The one-electron integrals needed to determine a nonrelativistic energy are generated by M302 and stored on the RWF file. The two-electron integrals are generated by M312 and ordered by M330. These integrals are stored on the INT file. The INT file can be retained and used in subsequent calculations. To avoid recomputing integrals a "NOINTS" line is inserted in the $Route input. This avoids the necessity of constructing a nonstandard Route when the two-electron integral file has been saved.

Transformed two-electron integrals and in some cases derivative integrals are also stored on the INT file. When large basis sets are in use, particularly when CI calculations are being conducted, it is often necessary to store the ordered and transformed integrals on separate files and a version of the MESA codes is available which uses separate integral files in the more commonly used Routes.

The two options which are available in these codes are PreEx and CutOff which are read from the $Route input. These parameters are defaulted to PreEx=15 and CutOff=10 in M312. These integers determine the criteria for computing integrals based on a preexponential factor, 10^{-preex}, and the criteria for retaining an integral after it has been computed, $10^{-cutoff}$. These parameters are also used in the computation of derivative integrals, link M323. The input format and defaults are M323=(PreEx=13,CutOff=9)

GUESS: Starting Set of Molecular Orbitals

Summary: This section describes the techniques used by MESA to construct a starting set of molecular orbitals.

Program: M401

Defaults: The default is to diagonalize the core Hamiltonian to generate the starting orbitals

Description:

A starting set of orbitals can be calculated in a number of ways in MESA. They can be obtained from an earlier calculation or a one-electron Hamiltonian will be constructed and diagonalized to generate the starting orbitals. The orbitals from two previous calculations can also be merged to produce a set of starting orbitals. In this context merged means groups of orbitals from the input sets can be selected as the user sees fit to produce a final set of the proper dimension.

One can also use molecular orbitals constructed in a different basis set to generate a set of starting orbitals. If the current basis set was obtained by augmenting a smaller basis (here augment requires that the additional basis functions are added to the end of the basis set) then an "expand" option is available to generate the new orbital set. If guess=rdchk is specified, then a corresponding orbital transformation is used to generate the orbitals in the new basis.

The guess=merge option requires that the set1 = set and set2 = set input lines be included in the $Route. These two sets are CHK files from previous calculations. One of these files must currently be renamed as SAV. A set of triplets are used to specify how the orbitals are selected from each set. The format of this input is as follows,

$$IND = (s_1, b_1, e_1, s_2, b_2, e_2 \ldots, s_n, b_n, e_n).$$

s_i = the orbital set

b_i = the starting orbital

e_i = the ending orbital

where the first $e_1 - b_1 + 1$ orbitals were taken from set s_1 and the next $e_2 - b_2 + 1$ orbitals are taken from set s_2, and so forth. For a DZ H_2O calculation the input might look like,

```
$Route
        guess = (merge,ind=(1,2,3, 2,1,1, 1,4,5, 2,5,13))
        set1 = sav              set2 = chk
$End
```

The first two orbitals come from set1 which is the SAV file. The next orbital comes from set2 and is the first orbital in that set. All of the orbitals are specified in this way until the fourteen orbitals comprising a DZ H_2O set of orbitals are defined.

The other input options are described in sufficient detail in the Quick User's Guide so that further explanations at this point are not needed.

SCF: Self-Consistent-Field Program

Summary:

General SCF Program

Program: M501

Defaults:

Defaults exist for closed-shell and high spin open-shell wavefunctions.

Description:

The SCF program utilizes either Pulay's DIIS algorithm or the Page-McIver method. Default input exists for closed-shell or high spin open-shell calculations. Once the spin multiplicity is specified in the $Route input string, no other input to the SCF code is required for these wavefunctions. The "HF" line in the $Route input results in the SCF program being included in the Route used by MESA.

The definition of the coupling coefficients used to construct the Fock operators for a General SCF calculation are given in the Input Summary portion of this manual. In a General SCF calculation the DIIS algorithm is used and options exist to display the diagonal elements of the Fock operators and to introduce damping factors if convergence difficulties are encountered. A sample input for a General SCF run is included in the Sample Input section of this manual.

At the conclusion of a SCF run, the SCF vectors are written to the RWF file and a character string is also written to this file to tell the integral transformation program to use SCF vectors in the transformation.

IVO: Improved Virtual Orbitals

Summary:

This code is used after a SCF calculation to generate a set of virtual orbitals which see a positive ion core.

Program: M502

Defaults:

1) The highest occupied molecular orbital in the closed shell SCF calculation is singly occupied in the IVO calculation.

2) The open shell electrons are triplet coupled.

Description:

If the default input options are acceptable then only a nonstandard route need be prepared where link 502 is executed after the SCF calculation (link 501). Two Fock operators are constructed during an IVO calculation. A closed shell Fock operator defined as

$$F_c = \mathfrak{h} + 2\mathfrak{J}_c - \mathfrak{K}_c$$

where $\mathfrak{J}_c$ and $\mathfrak{K}_c$ are the coulomb and exchange matrices constructed from the doubly occupied electron density matrix defined by the orbitals in shell "1". $\mathfrak{h}$ is the matrix of one-electron integrals. The IVO Fock operator is diagonalized to determine the virtual orbitals and is defined as

$$F_{IVO} = \sum_{i}^{nshell-1} f_i \mathfrak{h} + alpha_i \mathfrak{J}_i + beta_i \mathfrak{K}_i$$

The energy of the m'th excited state in the IVO approximation is

$$E_m = E_{closed} + E_{open} + e_m + E_{nuc}$$

where E_{closed} is the closed shell energy.

$$E_{closed} = \sum_{ij}^{atomic\ orbitals} \mathfrak{d}^c_{ij}(\mathfrak{h}_{ij} + \mathfrak{F}^c_{ij})$$

where $\mathfrak{d}_c$ is the density matrix constructed from the closed shell orbitals, shell "1".

$$E_{open} = \sum_{ij} \mathfrak{d}^o_{ij} \mathfrak{F}^c_{ij}$$

where $\mathbb{d}^o$ is the density constructed from the open shell orbitals, shell "2'. e_m is the m'th eigenvalue of the IVO Fock operator and E_{nuc} is the nuclear repulsion energy. The options in an IVO calculation include the ability to redefine the orbital which is to be singly occupied in the core and the ability to specify singlet or triplet coupling of the open shell electrons. Some flexibility is also available in determining the origin of the input molecular orbitals. These options are described in the Input Summary.

CI: Configuration Interaction Codes

Summary:

There are three CI codes which can be used in MESA. A Full CI Code (FCI) and a general CI Code (GCI) are also used by the MCSCF program. These two codes have a practical limitation of perhaps 10,000 CSFs. There is also a multireference CI code (MRCI) which can handle much larger CI expansions of perhaps 1,500,000 CSFs, but is more limited than the GCI code in the choice of CSFs which can be handled. There is also a version of the MRCI code, FMRCI, which employs a formula tape during a portion of the calculation. The formula tape version of the MRCI code is much faster than its predecessor but can require a modest amount of IO to read the internal formulas. All of these codes are based on a Graphical Unitary Group (GUGA) formulation of the CI problem

Defaults:

See discussion below and the Input Summary

Programs:

M801, M802, M805, M901, M902, M903, M914, M915, M916

Description:

In MESA, a CI calculation is invoked by including a "CI" input line in the $Route input. The CI configuration list is generated by running a Distinct Row Table (DRT) program. There are three such programs:

1) M801 is used by the multireference CI code, M901. The DRT option is read in $Route character string and then the input is searched for a $DRT card. The reference configurations are input after the $DRT keyword. In the $Route character string, input options such as the number of reference configurations, the excitation levels that are to be allowed into each class of orbitals and the molecular point group are determined. These options are explained in the Quick User's Guide.

The reference configurations and orbital classes are defined by orbital occupation codes. There are eight such codes which are used:

	CODE	EXPLANATION
1)	fzc	frozen core
2)	fzv	frozen virtual
3)	cor	restricted core
4)	vir	restricted virtual
5)	doc	doubly occupied
6)	uoc	virtual
7)	alp	alpha occupancy (couple up)
8)	bet	beta occupancy (couple down)

A complete occupation code has the form: (repeat count)(key) code symmetry. The "repeat count" and "key" portions of the occupation code are optional. The "symmetry" input is a number which refers to an irreducible representation. For D_{2h} symmetry, one must use Cotton's convention when determining the irreducible representation of an orbital. There are two orbital occupation keys:

keys:

% labels the active orbital space for First Order, Second Order and POL CI calculations

/ labels active orbitals when one is inputting a list of reference configurations. These orbitals have different codes associated with them in some of the reference configurations.

No excitations are performed into or out of frozen or restricted orbitals. Frozen orbitals are not included in the transformation of the integrals from the AO basis to the MO basis. So a Fock operator is constructed with frozen core orbitals and added to the one electron integrals. The frozen virtual orbitals are truncated in the CI problem. Restricted orbitals are included in the integral transformation step. This extra bit of work of transforming the virtual orbitals is needed when a CI gradient is being performed since the full Lagrangian, which includes all of the orbitals, is needed.

Based on these orbital classes, one can define a zero'th order, a first order and second order wavefunction by the following electron distributions. In this example there are m electrons in the inactive space and n electrons in the active space in the reference or zero'th order wavefunctions. Each successive order includes the configuration in the lower order wavefunction.

INACTIVE	ACTIVE	VIRTUAL	WAVEFUNCTION
m	n	0	Zero'th Order
m-1	n+1	0	
m-1	n	1	First Order
m	n-1	1	
m-2	n+2	0	
m-2	n	2	Second Order
m	n	2	
m-1	n-1	2	

Example: Second order CI for H_20 in C_S Symmetry

```
$Route
        DRT = (nsym=2,refexcite=99,vexcite=2,noint)
$End
$DRT
        2cor1  2doc1  2%doc1  2%uoc1  12uoc1 1vir1
               1doc2                  5uoc2
$End
```

Explanation: In a First or Second Order CI one defines the following orbital classes; inactive (orbitals which are doubly occupied in the reference configurations), active orbitals (orbitals whose occupation may differ in each reference configuration) and virtual orbitals (orbitals which are unoccupied in the reference configurations). The active orbitals define an orbital space in which a Full CI is performed to generate the reference configurations.

Example: multireference CI DZP H_2O in C_1 Symmetry all, single and double excitations from five references

```
$Route
        DRT = (nrefs=5)
        CI
$End
$DRT
        1cor1  2doc1  2/doc1  2/uoc1  18uoc1  1vir1
                2alp1  2bet1
                2alp1   bet1   alp1   bet1
                doc1    uoc1   doc1   uoc1
                uoc1    doc1   uoc1   doc1
$End
```

Explanation: In each reference after the first, only the active orbitals are defined. Note that each reference refers to a particular configuration state function or spin coupling and not simply to an electron occupation. By using the "alp" and "bet" orbital keys a genealogical spin coupling scheme is being evoked. The reference **alp1 alp1 bet1 bet1** means that the first two electrons are coupled to a triplet, then the next open shell electron is coupled to the triplet to produce a doublet and the final open shell electron is coupled to the doublet to produce a singlet state. The reference **alp1 bet1 alp1 bet1** means the first two open shell electrons are singlet coupled, the next open shell electron is coupled to the singlet to produce a doublet and the final open shell electron is coupled to the doublet to produce a singlet state. This exhausts the number of ways four open shell electrons can be coupled to a singlet so we must invoke a convention to eliminate redundant couplings. The convention that is adopted by the code is that we can not beta couple (couple down) from a singlet. We then exclude the following references

```
bet1   alp1   bet1 . . . etc.
alp1   bet1   bet1 . . . etc.
```

In this example we are stretching one of the OH bonds so this bonding orbital and its antibonding partner would be orbitals five and six, counting from the left, in the first reference configuration. The other OH bonding orbital would be orbital four and its antibonding partner would be orbital seven.

2) M802 is used by the general CI code, M902 and by the full CI code, M903. M802 is also used in an MCSCF calculation. M802 can use the same orbital codes that are employed in M801. This is often the most convenient way to set up a CAS calculation. In M802 the configuration list is generated by distributing electrons in orbital groups. The orbital code is of the form

(repeat count) "typ" type-number ";" symmetry

so if three orbital groups and two symmetries were present we might find the reference

2typ1;1 2typ1;2 4typ2;1 2typ3;2

Note that an orbital group can span a symmetry.

The $DRT input consists only of two lines, one which defines the orbital groups and the second line which contains three input parameters, na, nb and ns. These parameters determine the number of electrons in the problem, the spin multiplicity and the spatial symmetry of the wavefunctions.

2na + nb	= the number of electrons
nb+1	= the spin multiplicity
ns	= the symmetry number

The "nrefs" parameter in the drt input line of $Route is used to determine the number of electron distributions that will be defined. Electron distributions are defined by specifying the number of electrons in each orbital group. As an option one can also specify intermediate spin coupling for each orbital group. The number of electrons in each orbital group are input via the "numel" vector and the intermediate spin coupling is determined from the "spin" vector. There is one entry to these vectors for each orbital group and one set of vectors, "numel" and optionally "spin", determine one "reference" electron distribution. The entry to the "spin" vector is the multiplicity less one or 2S for the intermediate spin coupling. Once again, the spin coupling in a GUGA program is genealogical, so if spin = (S_1, S_2, S_3, . . .) the electrons in group 1 are coupled so all of the configuration state functions have spin symmetry, $S = S_1/2$, or a spin multiplicity of S_1+1. The electrons in group 2 are coupled to the electrons in group 1 to yield an intermediate spin symmetry of $S=S_2/2$ or a spin multiplicity of S_2+1.

When the "spin" vector is not present, all possible intermediate couplings, consistent with the overall spin multiplicity, nb+1, are present.

Example:

```
$Route
        DRT = (nsym=2,nrefs=5,ngroups=4)
$End
$DRT
        2typ1;1  3type2;1  2typ3;1  3typ3;2  10typ4;1  12typ4;2
        na = 4  nb = 1  ns = 1
$End
$Groups
        numel = (4,3,0,0)
        numel = (4,2,1,0)
        numel = (4,1,2,0)
        numel = (4,1,1,1)
        numel = (4,2,0,1)
$End
```

Explanation: There are two symmetry types so the point group is isomorphic with C_S. The configuration state functions transform like the first irreducible representation in the point group, ns = 1. There are nine electrons in the problem, 2na + nb = 9. The spin multiplicity is doublet, nb + 1 = 2. The first group of orbitals is doubly occupied in all configurations. In the first reference distribution, three electrons are distributed in all possible ways, consistent with the overall spin and spatial symmetry defined by nb and ns, among the three orbitals in group two. In the second reference distribution two electrons are distributed in all possible ways among the three orbitals of group two and one electron is distributed in all possible ways among the five orbitals of group three. In reference two, the doubly occupied orbital must be singlet coupled, the open shell electrons in group two are then forced to be triplet coupled. The final electron in group three is then coupled to the triplet to produce doublet configuration state functions.

3) M805 is a version of M802 used by the MCSCF code. In a multireference CI calculation, one distinguishes the drt input to the CI from the drt input to the MCSCF code. M805 does this by searching for drtmc in the $Route input in place of drt. M805 also searches for $DRTMC and $GroupMC in place of $DRT and $Groups, respectively. In all

other aspects, the input to M805 is identical to the input to M802. The input to the Davidson diagonalization step in a CI calculation is described in the Input Summary portion of this manual. The input parameters need no further explanation at this point, I only note that the MCSCF code has its own input parameters to the diagonalization procedure and these are also explained in the Input Summary.

MCSCF: Multiconfiguration SCF Code

Summary:

A general second-order MCSCF program. Both CAS and general CI expansions are supported. First and Second derivatives of MCSCF wavefunctions are available. This program is also used in Quadratic SCF calculations.

Program: M551

Defaults:

See Input Summary and the discussion below

Description:

The MCSCF program uses either the full CI code, M903, or the general CI code, M902, in the CI portions of the program. Both MCSCF and DRT (or DRTMC in the case of a multireference CI calculation) input lines are needed in the $Route input. When running a MCSCF calculation, the number of active orbitals is defined by the "nbf" option in the DRT input line. The number of doubly occupied core (inactive) orbitals are specified by the number-of-core-orbitals option in the MCSCF input line. Both of these input options are necessary for the code to function properly. The number of virtual orbitals is determined by reading the number of basis functions from the RWF file and subtracting the occupied orbitals (the number of inactive plus active orbitals) from this number. The minimum input needed to run a MCSCF calculation, in addition to the usual geometry and basis set input, is DRT input, with "nbf" specified, to define the CI expansion and a MCSCF input line with the number of core orbitals (number-of-core-orbitals) specified.

If a general CI expansion, as opposed to a CAS or full CI expansion, is employed and the energy is invariant to the rotations among a group of active orbitals then the input line invariant-orbital-sets, which is a vector whose length is the number of active orbitals, must be specified for the second-order portion of the code to work correctly. For example, if we were stretching the OH bonds in the H_2O molecule we might define a MCSCF calculation where two electrons of one OH bond were distributed in all possible ways among a bonding and antibonding pair of orbitals and the same treatment might be applied to the other OH bond. This wavefunction is the direct produce of two full CI wavefunctions defined in subsets of the active orbitals. In such a wavefunction, the energy is invariant to rotating the orbitals which belong to the same subset of active orbitals. This is a property of a full CI performed in an orbital subset. The energy would not be invariant to the rotation of orbitals between subsets. The input to such a calculation is presented below.

Example: DZP H_2O 2in2 X 2in2 MCSCF C_1 Symmetry

```
$Route
        drt = (ngroups=2,nrefs=1,nbf=4)
        mcscf = (number-of-core-orbitals=3,invariant-orbital-sets=(1,2,2,1))
$End
$DRT
        1typ1;1  2typ2;1  1typ1;1
        na = 2  nb = 0  ns = 1
$End
$Groups
        numel = (2,2)
$End
```

Explanation: The first three orbitals are doubly occupied and the next four orbitals comprise the active orbital space. The first two active orbitals are the OH bonding orbitals. Active orbitals two and three comprise one OH bond-antibond pair and active orbitals one and four comprise the other. The DRT input is described in the Input Summary and in the DRT section of the Program Description portion of this manual so only a brief descriptions will be given here.

The DRT input defines two orbital groups, typ1 and typ2. There are four electrons in the CI portion of the problem, 2 * na = 4, and the spin multiplicity is singlet, nb + 1 =1. Since "nsym" was not specified in the drt input line nsym = 1 or C_1 symmetry is the default . The ns = 1 input, which determines the spatial symmetry of the CSFs, is only included in this example to complete the list of drt input parameters. The nrefs = 1 input string indicates that one electron distribution is to be processed in the $Group input section. The "numel" vector specifies that there are two electrons distributed in all possible ways in group one and a like number of electrons distributed in group two. Active orbitals with the same set number, as defined by the "invariant-orbital-sets" input vector, are not rotated at first order during the MCSCF calculation.

A CASSCF input deck for the same problem would be:

Example: 4in4 CAS for H_2O C_1 symmetry

```
$Route
        DRT = (vexcite=99,nbf=4)
        MCSCF = (CAS,number-of-core-orbitals=3)
$End
$DRT
 2doc1  2uoc1
$End
```

Explanation: The DRT codes M802 and M805 recognize two types of input. The group input exemplified in the preceding general MCSCF example and the orbital key input used by M801 to defined doubly occupied (doc) orbitals and unoccupied orbitals (uoc) in the reference as well as alpha (alp) and beta (bet), spin coupling of the open shell electrons. In a CAS calculation it is most convenient to employ this orbital key input scheme to define the CI expansion. In this example four electrons would be distributed in all possible ways among the four active orbitals. The spin and spatial symmetry are determined from the symmetry of the reference configuration in this input scheme. The vexcite=99 input option specifies that a full CI will be performed in the active orbital space. We will consider one more CAS input example before we discuss the selection of starting MCSCF orbitals.

Example: 6in6 CAS for triplet CH_2 C_1 symmetry

```
$Route
        DRT = (nbf=6,vexcite=99)
        MCSCF = (number-of-core-orbitals=1,CAS)
$End
$DRT
        2doc1  2alp1  2uoc1
$End
```

Explanation: Note that the order of the input options in the DRT and MCSCF input lines is arbitrary.

SCF virtual orbitals at equilibrium geometries are often not an appropriate starting point in a MCSCF calculation. There are several options offered in MESA to provide a better set starting orbitals. One can use Natural Orbitals from a single reference CI calculation to start the MCSCF procedure and this is probably the best choice of starting orbitals if the MCSCF active space is larger than the molecule's valence space. MESA also has the ability to generate IVO's and these orbitals may be used to initate an MCSCF calculation. Alternatively, if one is concerned with dissociating one or more bonds, one can stretch these bonds and run an SCF calculation at this point on the surface. When the bonds are stretched, the highest occupied MOs are the stretched bonds and the lowest virtual orbitals are the antibonding partners of these bonding orbitals. These orbitals can then be used near equilibrium to start the MCSCF calculation. Once one set of MCSCF orbitals is obtained, then these orbitals will be used to start subsequent MCSCF calculations either at different points on the potential energy surface or in excited state calculations. There are other attractive ways to generate starting MCSCF orbitals via localization and corresponding orbital schemes and these will be present in future versions of the code.

PROPERTIES:

Summary: A description of the codes used to evaluate molecular properties.

Programs: M921, M1902, M1951

Defaults:

Description:

Molecular property integrals are evaluated in link M1902. One-electron density matrices are computed in the SCF and MCSCF codes. For CI wavefunctions, density and transition density matrices are constructed in link M921. The density matrices are traced with the property integrals in link M1951. Properties are currently determined as an expectation value for CI wavefunctions.

To evaluate a molecular property a **property=prop** input line is included in the $Route input section. Multiple properties are evaluated by including the line **property=(prop$_1$, prop$_2$, . . ., prop$_n$)**. To determine a transition property, the input line **properties=(transition=(1,2,3),el)** or **properties=(transition=(1,*), el)** is used. In the first case transition properties between roots 1 and 2, 1 and 3, as well as 2 and 3 are computed. The string **el** indicates dipole properties will be evaluated. In the second example, transition properties between the ground state and all available excited states are computed. Expectation values for the following properties are supported.

Key	Property
V0	Electrostatic potential at the nuclei
V1	Electrostatic field at the nuclei
E0	Electric monopole
E1	Electric dipole
E2	Electric quadrupole
E3	Electric octupole
E4	Electric hexadecapole
Populations	Full population analysis
Populations=Orbital	Mulliken Gross population analysis
Populations=Bond	Bond-order analysis
Populations=Charge	Evaluate atomic charges

Link M1902 also will evaluate mass velocity, momentum and Darwin integrals. The appropriate flags are:

Key	Integral
Del	Momentum
MV	Mass velocity
Fermi	Fermi contact on Darwin term

OPT: Geometry Optimization Package

Summary:

This section describes the geometry optimization driver in MESA.

Program: M201

Defaults:

1) Convergence Criteria
2) Maximum Stepsize
3) Number of Cycles
4) Gradient Tolerance (if this tolerance is exceeded the program shuts down)
5) Optimization Method

Description:

A geometry optimization calculation in MESA requires Z-Matrix (internal coordinate) geometry input. Geometry optimization is invoked with an OPT option in the $Route input. The optimization codes require very little input. An OPT=BERNY line in $Route is the extent of the additional input required in most calculations. Transition State searches are invoked by an OPT=TS input line. The initial force constant matrix in a Transition State search is either obtained by running an analytic or finite-difference force constant calculation or by specifying a negative diagonal element of the force constant matrix. This last option is invoked by using the D2E option in the $Geom input . Force constants from previous calculations are used in the optimization if the SRCD2E=RDCHK option is included in the OPT input line,

```
$Route
opt=(berny,ts,srcd2e=rdchk)
$End
```

A search for the transition state in the cis-trans isomerization of HONO provides an example where the D2E option can be used to determine the initial force constant matrix.

Example: Z-Matrix input for HONO

```
$Route
opt=(berny,ts)
$End
$Geom
o1
n  o1  r1
o2 n   r2  o1 ang1
h  o1  r3  n   ang2  o1  dih

r1=1.3          constant
r2=1.4
r3=0.9
ang1=110.
ang2=120.
dih=80.         d2e=-1.
$end
```

Explanation: In this example we are searching for the transition state in the cis-trans isomerization of HONO. The dihedral angle, dih, associated with the rotation of the hydrogen atom out of the plane formed by the O-N-O atoms is expected to be the internal coordinate which dominates the description of the Reaction Path so this is the diagonal element of the initial force constant matrix which is given a negative value. One also has the ability to freeze internal coordinates during an optimization procedure by declaring them to be held constant during the calculation. This option was also employed in the HONO input as the O1-N bond distance was not allowed to vary. These options are discussed in more detail in Geometry Input portion of this manual.

SCF and MCSCF wavefunctions can be used in geometry optimization runs without regard to the size of the gaussian basis sets employed in the calculation. CI optimizations are limited to sixty to eighty basis functions in this version of MESA as two-electron derivative integrals are currently stored on disk.

In CI optimizations, two additional input lines are needed in $Route. CIOPT and TRANSFORMATION=SCF are needed in a single reference CI optimization. In a multireference CI optimization MRCIOPT is needed in place of CIOPT and the TRANSFORMATION=SCF line is also required.

FC: Force Constant Calculations

Summary:

This section describes the finite-difference and analytic methods used in MESA to determine force constant matrices and harmonic vibrational frequencies.

Program: M203 M204 M205

Defaults:

1) Finite Difference
2) One Point Derivative Evaluation
3) Internal Coordinates

Description:

Finite difference force constants and harmonic frequencies are available for SCF, MCSCF and CI wavefunctions. This option is invoked by including OPT=FORCE-CONSTANTS in the $Route input. Frequencies are obtained by including FREQUENCIES as a separate input line in $Route. If dummy atoms are used in the Z-Matrix input then the finite difference procedure must vary cartesian coordinates rather than internal coordinates. A two point derivative expression can also be used in a finite difference calculation. A FORCE-CONSTANTS input line is used to turn on these options.

```
$Route
opt=(force-constants) frequencies
force-constants=(cartesian,two-point)
$End
```

One usually runs a force constant calculation after MESA has completed a geometry optimization run. It is then convenient to read the optimized geometry from the previous CHK file. This is accomplished with a GEOM=RDCHK line in $Route.

One should also note that NDERIV=2 must be included in $Route in an analytic force constant calculation. The other input options are identical to those required in a geometry optimization run with the same wavefunction.

IO: The MESA IO Library

Summary: Word Addressable IO Library where character strings are used to designate blocks of data

Program: All of the codes in MESA are linked to these IO routines

Description:

IOSys is an attempt at a flexible, user-friendly IO package which hides the machine dependent aspects of IO. All IO in the MESA system other than Fortran formatted input and output should be performed using IOSys. The form of a call to IOSys is:

```
call IOSys(directive,arg1,arg2,arg3,arg4)
```

where "directive" is a character string giving (in English?) the action to be taken, and the remaining three arguments depend on what
is being done.

In the following description of the directives <file> is an OSys internal file name; <unit> is an IOSys internal unit name; parenthesis ( ) indicate a choice of words, one of which must be used; and brackets [ ] indicate an optional keyword or phrase. Also, all (hopefully) non-essential words such as "to", "on" and "of" may be omitted wherever they appear; however, their inclusion is suggested as an aid to readability.

The possible directives are:

"length of <file> on <unit>"
"get length of <file> on <unit>"
"get maximum length of <file> on <unit>"
"get read pointer of <file> on <unit>"
"get write pointer of <file> on <unit>"
"does <file> exist on <unit>"
"write (character integer real) <file> (to,on) <unit>
[without rewinding] [asynchronously]"
"read (character integer real) <file> from <unit>
[without rewinding] [asynchronously]"
"end <file> on <unit>"
"endfile <file> on <unit>"
"create (character integer real) <file> on <unit>"

Note: An "endless" file does not appear to require closing if it is a scratch unit, but an file on a permanent unit must be closed!!

"open <unit> [as (new,old,unknown,scratch)] [on ssd]"
"dump"
"destroy [<unit>]"
"close (all,<unit>)"
"wait for (all,<unit>)"
"rewind (all,<file>) on (all,<unit>)
[(read,write,read-and-write)]"
"set alignment (for,of) <unit>"
"reset alignment (for,of) <unit>"

IOSys manages collections of "files" on "units", which are actually physical disc files. IOSys uses a directory to keep track of its logical "files" on each of its logical "units". The first step in using IOSys is to open one or more logical units using the "open" directive. This is the only stage where the user must be cognizant of the physical file name associated with the logical units. After a unit is opened, it is always referred to by its logical name embedded in a directive string.

There are two ways to create a logical file on a unit: the first is simply by writing data to a new file name using the "write" directive, the second is with an explicit "create" directive. The first method will create a file exactly the length of the data being written; whereas, a "create" directive allows the user to specify the length of the file. As a user, you will only need to explicitly create files only if all the data cannot be written to the file in one write. There is also a variant of the explicit create invoked by giving a length of -1. In this case, an "endless" file is created, on which an unspecified amount of data may be written. An "endless" file is active until an "end" or "endfile" directive is issued, at which point the "endless" file is fixed to the current length and may not be further extended. There are restrictions on the use of "endless" files: first, only one endless file can be active on a given unit at one time; and second, no new files may be created implicitly or explicitly on a unit with an active endless file.

Logical file and unit names may have up to 16 characters; any beyond 16 are truncated. If the name contains embedded blanks, it must be enclosed in double quotes, e.g. "my file". The quotes are treated as part of the name, so a name surrounded with quotes is not the same as the name without quotes. IOSys can currently handle 20 units open simultaneously with a total of 1000 files. These limits are set in the parameter statement and are readily modified.

The length of a file or of a transfer (read or write) are in terms of the type of data on the file. Integer and real files are in terms of integer or real words respectively, while character data is by character. Also note that for character data, the length need not be given as it can be assumed to be the length of the character string passed in.

"Alignment" has to do with the physical storage of data. If alignment is set, new files created will be positioned starting at physical disc sectors. This will speed up IO operations, but will also make the physical file larger since gaps will be left between logical files. Another ramification of alignment is that the "maximum length" of files can be larger than their "length" since IOSys will allow a file to expand in to the gap left by alignment. Alignment is turned off with a "reset alignment" directive. By default, units are not aligned. Finally, the alignment of a unit can be changed as often as desired since it only affects new files being created.

This section will describe briefly each directive and the other arguments associated with it. A "u" will denote that the argument is not referenced when using a directive, and the user should use a "0" or any variable in the calling sequence.

MANIPULATING UNITS:

call iosys("open <unit> [as (new,old,unknown)] [on ssd]",size, u,u,name)

"name" is the hollerith (or character) name or a unit number of the physical disc file to use. It must have 7 or less characters.

"size" is the size of family members on CTSS systems.

The default for an open is "unknown", which means that if the file exists, it will be opened, if not, it will be created. "Old" specifies that the file must exist; otherwise, there is an error. "New" guarantees a new file; if it already exists, it will be deleted and then recreated. "scratch" can be used to create a unique temporary file. In this case, no physical file name is needed as IOSys will create one, and when the unit is closed, the physical file will be deleted. "on ssd" specifies that the file should be put on a solid-state disc if possible. If there is not room, or no SSD, the file is created on a normal disc. The SSD should only be used for heavily used, temporary files that only exist as scratch in one program link.

call iosys("close (all,<unit>)",u,u,u,u)

This directive closes either a specific unit or all open units. When a unit is closed, the directory information is saved on the unit so that a subsequent open will restore the unit to the identical state as when it was closed — including the read and write pointers for files.

call iosys("dump",u,u,u,u)"

This will dump a listing of the file names, pointers, etc for all units to the output file.

call iosys("destroy [<unit>]",name,u,u,u)

This directive works in two fashions: if an IOSys unit name is given in the directive, that IOSys unit is closed and the physical file is destroyed. In this case, the argument "name" is not referenced. Otherwise, the physical file "name" is destroyed, regardless of whether it is an IOSys file or not.

call iosys("set alignment on <unit>",u,u,u,u)
call iosys("reset alignment on <unit>",u,u,u,u)

These calls turn on and off the alignment of new files to physical sector boundaries.

MANIPULATING FILES:

call iosys("create (character integer real) <file> on <unit>", length,u,u,u)

This call creates a file of length "length" data elements on an IOSys unit. If "length" is -1, an "endless" file is created (see above for a description of endless files)

call iosys("write (character integer real) <file> (to,on) <unit>
[without rewinding] [asynchronously]",nwords,array,offset, c)

Nwords" is the number of data elements to transfer.
"Array" is the array to transfer from.
"Offset" is added to the current write pointer before writing.
"C" is the character string if writing characters.

If "nwords" is -1, IOSys will use the current size of the file and transfer that number of data elements, which is a nice shortcut for rewriting a file. The default is to rewind the file before writing. If this is not overidden with "without rewinding", then "offset" specifies a location in the file to begin writing (starting at 0). Otherwise, "offset" is added to the current write location before writing. Thus, repeated calls "without rewinding" and an "offset" of 0 will sequentially write a file. Note that the read and write pointers are separate and do not affect each other. For character data, the length need not be given, in which case the length of "C" will be used.

call iosys("read (character integer real) <file> from <unit>
[without rewinding] [asynchronously]",nwords,array,offset,c)

Reading is exactly the inverse of writing, and all options and arguments are identical to the above.

call iosys("end <file> on <unit>",u,u,u,u)
call iosys("endfile <file> on <unit>,u,u,u,u)

Use this to terminate an "endless" file. The length of the file is fixed at the length at the time of this call.

call iosys("wait for <unit>",u,u,u,u)

Used to wait for the completion of an asynchronous IO request. An array being written asynchronously cannot safely be modified until a "wait" call has been made. Conversely, data being read asynchronously cannot be used before a call to "wait".

call iosys("rewind (all,<file>) on (all,<unit>) [(read,write,
read-and-write)]",u,u,u,u)

Rewinds the read and/or write pointers of one or all of the files on one or all units.

INQUIRIES ABOUT FILES:

call iosys("length of <file> on <unit>",length,u,u,u)
call iosys("get length of <file> on <unit>",length,u,u,u)

Returns the length of the data written to a file. If the file does not exist, -1 is returned.

call iosys("get maximum length of <file> on <unit>",length,u,u,u)

The maximum amount of data which can be stored on a file. If the file does not exist, a -1 is returned.

call iosys("get read pointer of <file> on <unit>",pointer,u,u,u)
call iosys("get write pointer of <file> on <unit>",pointer,u,u,u)

These are the addresses in a file (counting from 0) where the next read or write would occur if the file is not rewound. If the file does not exist, a -1 is returned.

call iosys("does <file> exist on <unit>",u,u,u,answer)
"Answer" returns as "no" or "yes"

PRT: Print Options

Summary:

This sections contains a list of all the print options available in MESA. The codes which begin with "src", such as src902.f, are called by several programs. They are usually used by the MCSCF code and a main program with the same numeric designation, i.e. m902.f calls src902.f.

Program:

Defaults:

Description:

```
m102.f:    if(logkey(Ops,'PRINT=BASIS',.False.,' ')) then
m102.f:    if(logkey(Ops,'PRINT=ECP',.False.,' ')) then
m102.f:    if(logkey(ops,'PRINT=GRADIENT=POINTER-ARRAYS',.false.,' '))
m202.f:    prtdis=logkey(Ops,'PRINT=DIS',.True.,' ')
m202.f:    prtang=logkey(Ops,'PRINT=ANG',.True.,' ')
m202.f:    prtz=logkey(Ops,'PRINT=Z',.False.,' ')
m202.f:    prtc=logkey(Ops,'PRINT=C',.False.,' ')
m202.f:    prtsym=logkey(Ops,'PRINT=SYM',.False.,' ')
m302.f:    if(logkey(Ops,'PRINT=INT=S',.False.,RefOps)) then
m302.f:    if(logkey(Ops,'PRINT=INT=T',.False.,RefOps)) then
m302.f:    if(logkey(Ops,'PRINT=INT=V',.False.,RefOps)) then
m302.f:    if(logkey(Ops,'PRINT=INT=LP',.False.,RefOps)) then
m401.f:    if(logkey(Ops,'PRINT=EGUESS',.False.,' ')) then
m401.f:    if(logkey(Ops,'PRINT=GUESS',.False.,' ')) then
m401.f:    if(logkey(Ops,'PRINT=GUESS=ALL',.False.,' ')) then
m501.f:    if(logkey(ops,'PRINT=SCF=LAGRANGIAN=MO',.false.,' ')) then
m501.f:    if(logkey(ops,'PRINT=SCF=LAGRANGIAN=AO',.false.,' ')) then
m501.f:    if(logkey(ops,'SCF=PRINT=PSEUDOLAGRANGIAN',.false.,' ')) then
m501.f:    if(logkey(ops,'PRINT=SCF=CORE-LAGRANGIAN',.false.,' ')) then
m501.f:    printj=logkey(ops,'SCF=PRINT=J',.false.,' ')
m501.f:    printk=logkey(ops,'SCF=PRINT=K',.false.,' ')
m501.f:    if(logkey(ops,'SCF=PRINT=KINETIC',.false.,' ')) then
m501.f:    if(logkey(ops,'SCF=PRINT=POTENTIAL',.false.,' ')) then
```

```
m501.f:     if(logkey(ops,'SCF=PRINT=ONE-ELECTRON',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=GUESS',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=DENSITY',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=Q-MATRIX',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=DIIS-ERROR',.false.,' ' )) then
m501.f:     if(prnt.and.logkey(ops,'PRINT=SCF=ENERGY',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=AO-DIIS-ERROR',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=MO-DIIS-ERROR',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=DIIS-MATRIX',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=MO_FOCK_MATRIX',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=DIAGONAL_FOCK',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=ROTATION-MATRIX',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=VITERS',.false.,' ' )) then
m501.f:     if(logkey(ops,'SCF=PRINT=COEFFICIENTS',.false.,' ' )) then
m501.f:     if(prnt.and..not.logkey(ops,'PRINT=SCF=ENERGY',.false.,' ' )) then
m501.f:     if(logkey(ops,'PRINT=SCF=VECTOR',.false.,' ' )) then
m501.f:     if(chrkey(ops,'PRINT=SCF=VECTOR',' ',' ').eq.'ALL') then
m501.f:     if(logkey(ops,'PRINT=SCF=VECTOR',.false.,' ' )) then
m501.f:     if(logkey(ops,'PRINT=SCF=CORE-LAGRANGIAN',.false.,' ' )) then
m502.f:     if(logkey(Ops,'PRINT=IVO',.False.,' ' )) then
m502.f:     if(logkey(ops,'PRINT=IVO',.false.,' ' )) then
m551.f:     if(logkey(ops,'PRINT=SCF=DENSITY',.false.,' ' )) then
m551.f:     if(logkey(ops,'PRINT=SCF=DENSITY',.false.,' ' )) then
m551.f:     (logkey(ops,'PRINT=MCSCF=DENSITY-MATRIX-ENERGY',.false.,' ' ))
m551.f:     if(logkey(ops,'PRINT=MCSCF=HESSIAN',.false.,' ' )) nphes=1
m551.f:     if(logkey(ops,'PRINT=MCSCF=GRADIEN',.false.,' ' )) iprtg=1
m551.f:     if(logkey(ops,'PRINT=MCSCF=VECTOR',.false.,' ' )) iprtmo=1
m551.f:     if(logkey(ops,'PRINT=MCSCF=LAGRANGIAN',.false.,' ' )) iprtlag=1
m551.f:     if(logkey(ops,'PRINT=MCSCF=CI',.false.,' ' )) iprtci=1
m551.f:     if(logkey(mcinp,'PRINT=MCSCF=DENSITY-MATRIX-ENERGY',
m551.f:     if(logkey(mcinp,'PRINT=MCSCF=HESSIAN',.false.,' ' )) nphes=1
m551.f:     if(logkey(mcinp,'PRINT=MCSCF=GRADIENT',.false.,' ' )) iprtg=1
m551.f:     if(logkey(mcinp,'PRINT=M551=VECTOR',.false.,' ' )) iprtmo=1
m551.f:     if(logkey(mcinp,'PRINT=MCSCF=LAGRANGIAN',.false.,' ' ))
m551.f:     if(logkey(mcinp,'PRINT=MCSCF=CI',.false.,' ' )) iprtci=1
m551.f:     if(logkey(ops,'PRINT=SCF=LAGRANGIAN=MO',.false.,' ' )) then
```

```
m551.f:    if(logkey(ops,'PRINT=SCF=LAGRANGIAN=AO',.false.,' ' )) then
m551.f:    if(logkey(ops,'PRINT=MCSCF=LAGRANGIAN=MO',.false.,' ' )) then
m551.f:    if(logkey(ops,'PRINT=MCSCF=LAGRANGIAN=AO',.false.,' ' )) then
m551.f:    if(logkey(ops,'PRINT=MCSCF=ACTIVE-DENSITY=MO',.false.,' ' ))
m551.f:    if(logkey(ops,'PRINT=MCSCF=ACTIVE-DENSITY=AO',.false.,' ' ))
m701.f:    if(logkey(ops,'M701=PRINT=PRIMITIVE-TO-AO-MATRIX',.false.,' ' ))
m701.f:    if(logkey(ops,'M701=PRINT=AO-DENSITY-MATRIX',.false.,' ' ))
m701.f:    if(logkey(ops,]M701=PRINT=PRIMITIVE-DENSITY-MATRIX',
m701.f:    if(logkey(ops,'M701=PRINT=AO-DENSITY-MATRIX=MCSCF',
m701.f:    (logkey(ops,'M701=PRINT=PRIMITIVE-DENSITY-MATRIX=MCSCF',
m701.f:    if(logkey(ops,'M701=PRINT=AO-DENSITY-MATRIX=MCSCF',
m701.f:    i(logkey(ops,'M701=PRINT=PRIMITIVE-DENSITY-MATRIX=MCSCF',
m702.f:    (logkey(ops,'PRINT=GRADIENT=NUCLEAR-REPULSION',.false.,' ' ))
m702.f:    if(logkey(ops,'PRINT=GRADIENT=ONE-ELECTRON',.false.,' ' )) then
m702.f:    if(logkey(ops,'PRINT=GRADIENT=OVERLAP',.false.,' ' )) then
m702.f:    if(logkey(ops,'PRINT=GRADIENT=KINETIC',.false.,' ' )) then
m702.f:    if(logkey(ops,'PRINT=GRADIENT=POTENTIAL',.false.,' ' )) then
m702.f:    if(logkey(ops,'PRINT=GRADIENT=ECP',.false.,' ' )) then
m712.f:    if(logkey(ops,'M712=PRINT=LOCAL-DENSITY-MATRICES',
m712.f:    if(logkey(ops,'PRINT=GRADIENT=ONE-ELECTRON',.false.,' ' )) then
m712.f:    if(ci.and.logkey(ops,'PRINT=GRADIENT=TWO-ELECTRON',.false.,' ' )) then
m712.f:    if(logkey(ops,'PRINT=GRADIENT=INTEGRAL',.false.,' ' )) then
m731.f:    if(logkey(Ops,'PRINT=GRADIENT=TOTAL',.false.,' ' )) then
m801.f:    if(logkey(Ops,'PRINT=FORCE',.False.,' ' )) PrtFor=.True.
m801.f:    all=logkey(ops,'PRINT=DRT=ALL',.false.,' ' )
m801.f:    if(all.or.logkey(Ops,'PRINT=DRT=TABLE',.False.,' ' )) then
m801.f:    if(all.or.logkey(Ops,'PRINT=DRT=EWA',.False.,' ' )) then
m802.f:    if(all.or.logkey(Ops,'PRINT=DRT=IPA',.False.,' ' )) then
m805.f:    if(pass.eq.0.and.logkey(ops,'PRINT=DRT',.false.,' ' )) then
m811.f:    if(pass.eq.0.and.logkey(ops,'PRINT=DRTMC',.false.,' ' )) then
m811.f:    (logkey(ops,'PRINT=TRANSFORMATION=VECTOR=INPUT',.false.,
m921.f:    (ops,"PRINT=TRANSFORMATION=VECTOR=REORDERED',.false.,
m921.f:    if(logkey(ops,'PRINT=WALKS=FULL',.false.,' ' )) then
m921.f:    if(logkey(ops,'PRINT=CI-VECTOR',.false.,' ' )) then
m921.f:    if(logkey(ops,'PRINT=WALKS=FULL',.false.,' ' )) then
m921.f:    if(logkey(ops,'PRINT=CI=DENSITY".false.,' ' )) then
```

```
m921.f:     if(logkey(ops,'PRINT=CI-VECTOR',.false.,' ' )) then
m921.f:     if(logkey(Ops,'PRINT=CI=TDENSITY',.False.,' ' ))then
m921.f:     if(logkey(Ops,'PRINT=KOHN=TDENSITY',.False.,' ' ))then
m921.f:     if(logkey(Ops,'PRINT=CI=NO',.False.,' ' )) then
m921.f:     else if(logkey(Ops,'PRINT=CI=NO_OCC',.False.,' ' )) then
m921.f:     full=logkey(ops,'PRINT=WALKS=FULL',.false.,' ' )
m929.f:     if(logkey(ops,'PRINT=KOHN=P-SPACE',.false.,' ' )) then
src820.f:   if(logkey(ops,'PRINT=M820=TWO-ELECTRON-INTEGRALS',
src820.f:   (logkey(ops,"PRINT=M820=ONE-ELECTRON-INTEGRALS',.false.,' ' ))
src830.f:   if(logkey(ops,'PRINT=M830=GUGA-DENSITY-MATRIX',.false.,' ' ))
src902.f:   if(logkey(ops,'PRINT=CI=HPQ',.false.,' ' )) then
src902.f:   if(logkey(ops,'PRINT=CI=HMATRIX',.false.,' ' )) then
src902.f:   if(logkey(ops,'PRINT=CI=GUESS-MATRIX',.false.,' ' )) then
src902.f:   if(logkey(ops,'PRINT=CI=HMATRIX',.false.,' ' )) then
src902.f:   if(logkey(ops,'PRINT=CI=HMATRIX=ONLY',.false.,' ' )) return
src902.f:   if(logkey(ops,'PRINT=CI=GUGA-INTEGRALS',.false.,' ' )) then
src903.f:   if(logkey(ops,'PRINT=CI=DIAGONALS',.false.,' ' )) then
src903.f:   if(logkey(ops,'PRINT=CI=TRIALS',.false.,' ' )) then
m1001.f:    prtrhs=logkey(ops,'CPMC=PRINT=RHS',.false.,' ' )
m1001.f:    prtsol=logkey(ops,'CPMC=PRINT=SOLUTION',.false.,' ' )
m1001.f:    prtena=logkey(ops,'CPMC=PRINT=ENERGY',.false.,' ' )
m1001.f:    prttah=logkey(ops,'CPMC=PRINT=TA_H',.false.,' ' )
m1001.f:    prttag=logkey(ops,'CPMC=PRINT=TA_G',.false.,' ' )
m1001.f:    prtum=logkey(ops,'CPMC=PRINT=U-MATRIX',.false.,' ' )
m1001.f:    prtall=logkey(ops,'CPMC=PRINT=ALL',.false.,' ' )
m1001.f:    prtcp=logkey(ops,'CPMC=PRINT=CPMC',.false.,' ' )
m1001.f:    prtfor=logkey(ops,'CPMC=PRINT=FORCE',.false.,' ' )
m1010.f:    if(logkey(ops,'PRINT=DERIVATIVE=MO-LAGRANGIAN',.false.,' ' ))
m1010.f:    (logkey(ops,'PRINT=DERIVATIVE=MO-CORE-LAGRANGIAN',.false.,
m1010.f:    if(logkey(ops,'PRINT=DERIVATIVE=AO-LAGRANGIAN',.false.,' ' ))
m1010.f:    if(logkey(ops,'PRINT=GRADIENTS=SCF-FROM-LAGRANGIANS',
m1011.f:    if(logkey(ops,'PRINT=GENERALISED-LAGRANGIANS',.false.,' ' ))
m1012.f:    if(logkey(ops,'PRINT=CPHF=HESSIAN',.false.,' ' )) then
m1013.f:    if(logkey(ops,'PRINT=CPHF=B0',.false.,' ' )) then
m1014.f:    if(logkey(ops,'PRINT=GRADIENT=OMEGA',.false.,' ' )) then
m1014.f:    if(logkey(ops,'PRINT=GRADIENT=OMEGA',.false.,' ' )) then
```

```
m1020.f:    if(logkey(ops,'PRINT=CPHF=SOLUTIONS',.false.,' ' )) then
m1021.f:    if(logkey(ops,'PRINT=CPHF=SOLUTIONS',.false.,' ' )) then
m1022.f:    (logkey(ops,'PRINT=SECOND-DERIVATIVE=CPHF-TERM',.false.,' ' ))
m1022.f:    (logkey(ops,'PRINT=SECOND-DERIVATIVE=SA*EB-TERM',.false.,' ' ))
m1022.f:    if(logkey(ops,'PRINT=SECOND-DERIVATIVE=TOTAL',.false.,' ' ))
m1031.f:    if(logkey(ops,'PRINT=CI=LAGRANGIAN',.false.,' ' )) then
m1031.f:    if(logkey(ops,'PRINT=CI=LAGRANGIAN',.false.,' ' )) then
m1032.f:    if(logkey(ops,'PRINT=GRADIENT=LAGRANGIAN-TERM',.false.,' ' ))
m1032.f:    if(logkey(ops,'PRINT=GRADIENT=TOTAL',.false.,' ' ))
m1033.f:    if(logkey(ops,'PRINT=GRADIENT=LAGRANGIAN-TERM',.false.,' ' ))
m1033.f:    if(logkey(ops,'PRINT=GRADIENT=TOTAL',.false.,' ' ))
m1902.f:    if(logkey(Ops,'PRINT=PROPERTIES=DEL',.False.,refops)) then
m1902.f:    if(logkey(Ops,'PRINT=PROPERTIES=MV',.False.,refops)) then
m1902.f:    if(logkey(Ops,'PRINT=PROPERTIES=FERMI',.False.,refops)) then
m1902.f:    if(logkey(Ops,'PRINT=PROPERTIES=V0',.False.,refops)) then
m1902.f:    if(logkey(Ops,'PRINT=PROPERTIES=V1',.False.,refops)) then
m1902.f:    if(logkey(Ops,'PRINT=PROPERTIES=' //Mult(1:2),.False.,
m1951.f:    if(logkey(ops,'PRINT=SCF=VECTOR',.false.,' ' )) then
m1951.f:    if(chrkey(ops,'PRINT=SCF=VECTOR',' ',' ').eq.'ALL') then
m1951.f:    if(logkey(ops,'PRINT=NATURAL-ORBITALS',.false.,' ' )) then
m1951.f:    if(chrkey(ops,'PRINT=NATURAL-ORBITALS',' ',' ').eq.
m1951.f:    if(logkey(ops,'PRINT=NATURAL-ORBITALS=ROOTS',.false.,' ' ))
m1951.f:    if(logkey(ops,'PRINT=MCSCF=VECTOR',.false.,' ' )) then
m1951.f:    if(chrkey(ops,]PRINT=MCSCF=VECTOR',' ',' ').eq.
m1951.f:    (logkey(ops,'PRINT=NATURAL-ORBITALS=AVERAGED',.false.,' ' ))
```

LINKS:

Summary: A one line description of the MESA programs.

Program:

Defaults:

Description:

M0	MESA initialization program. It destroys old OUT and RWF files
M1:	Read the $Route input a generates a Route
M101:	Reads the $Geom input
M102:	Reads the basis set info and computes the number of electrons
M201:	Drives geometry optimization calculations
M202:	Generates molecular coordinates from Z-Matrix input
M203:	Transforms cartesian force constant matrix into internal coordinates
M204:	Computes harmonic vibrational frequencies given a force constant matrix
M205:	Drives finite difference force constant calculation
M302:	Computes one-electron and ECP integrals
M303:	Computes derivative contracted one-electron and ECP integrals
M312:	Computes two-electron integrals
M313:	Computes derivative contracted two-electron integrals
M323:	Computes derivative contracted two-electron integrals and writes a chained output file. All of the derivative integrals associated with a particular degree of freedom are chained together.
M330:	Orders two-electron integrals
M333:	Orders derivative two-electron integrals
M401:	Generates an initial set of MOs for SCF, MCSCF and CI calculations
M501:	General SCF program
M502	IVO program
M551:	MCSCF program
M701:	Transforms the contracted one-electron AO density to the primitive basis
M702:	Computes derivative primitive one-electron integrals and traces them with the density matrix

M712: Computes derivative primitive two-electron integrals and traces them with the density matrix

M731: Transforms the cartesian gradient into internal coordinates

M801: DRT program used by M901 and M916

M802: DRT program used by MCSCF and CI links M902 and M903

M805: DRT program used by MCSCF in a multireference CI calculation

M811: Transforms AO integrals into the MO basis and optionally sorts them to GUGA order

M812: Transforms gradient AO integrals into the MO basis

M814: Constructs a Fock matrix with derivative integrals

M820: This link sorts ordered MO integrals into GUGA order

M821: This links reads GUGA ordered integrals and adds supermatrix integrals to the GUGA integral file

M824: Reads the integrals produced by M323, orders them, and then constructs derivative Lagrangians and other quantities (Fock matrices and transformed integrals for a CPMCSCF calculation) which depend on derivative integrals. The integrals are processed one degree of freedom at a time. This code performs the function of M333, M814, and M1010.

M830: Sorts GUGA ordered density in MO order

M840: Sorts AO two-electron density into the Shell Block order needed by M712

M901: Shape driven GUGA multireference CI code

M902: General CI program for small CSF expansions. Constructs two-particle density.

M903: Full CI code. Constructs two-particle density.

M911 Constructs two-particle density for Guga-CI codes (M901 and M916)

M914: Determines the length of the formula tape for link M916

M915: Computes the formulas for M916

M916: Formula tape version of M901

M921: Analyzes CI wavefunctions and computes the CI one-electron (transition) density

M1001: Coupled perturbed MCSCF program

M1010: Constructs an SCF Lagrangian with derivative integrals

M1011: Constructs generalized SCF Lagranian

M1012: Constructs the CPHF hessian

M1013: Builds the RHS of the CPHF equations

M1014:	Builds omega term for SCF second derivatives
M1020:	Solves CPHF equations
M1021:	This link adds occupied-occupied and virtual-virtual terms to the CPHF solutions
M1022:	Assembles final SCF force-constant matrix
M1031:	Constructs the CI lagrangian
M1032:	Assembles the final CI gradient in a single reference CI calculation
M1033	Assembles the final CI gradient in a multireference CI calculation
M1902:	Computes property integrals
M1951:	Analyzes molecular orbitals and evaluates molecular properties
M2001:	Prints timing information and copies data from the RWF file to the CHK file

EXAMPLES: Sample Inputs

Summary:

Input decks for the most commonly used Routes through MESA

Program:

Defaults:

Description:

```
$title
  dzp h2o closed shell scf
 $end
$route
maxsiz=900000
2s+1=1
print=(basis,scf=vector)
hf scf=(pulay)
guess=(huckel)
basis=dzp,sym=off
$end


$geom
o
h1 o r1
h2 o r2 h1 a1

r1=1.949
r2=1.949
a1=112.3
$end
```

```
$dzp h
/ hydrogen      scaled by 1.2
 type=s
        19.2406        0.032828
        2.8992         0.231208
        0.6534         0.817238
 type=s
        0.1776         1.0
 type=p
        1.0000         1.0
$end
$dzp o
/ oxygen
 type=s
        7816.5400      0.002031
        1175.8200      0.015436
        273.1880       0.073771
        81.1696        0.247606
        27.1836        0.611832
        3.4136         0.241205
 type=s
        9.5322         1.0
 type=s
        0.9398         1.0
 type=s
        0.2846         1.0
 type=p
        35.1832        0.019580
        7.9040         0.124189
        2.3051         0.394727
        0.7171         0.627375
 type=p
        0.2137         1.0
 type=d
        0.85           1.0
$end
```

```
$title
   general-scf test case   boron atom   dunning basis
$end
$route
 2s+1=2  q=0  geom=(coord,inau)
 print=(scf=(vector=all)) sym=off  guess=core
 hf  scf=(convergence=5, gscf, nshell=2,
 shlnbf=(2,3),   fn=(1.,1.),fd=(1.,6.),
  an=(2.,1.,0.),ad=(1.,3.,1.),
  bn=(-1.,-1.,0.),bd=(1.,6.,1.))
$end
$geom
b(basis=dz) 0.  0.  0.
$end
$dz b
/ boron
      type=s
      2788.4100     0.002122
      419.0390      0.016171
      96.4683       0.078356
      28.0694       0.263250
      9.3760        0.596729
      1.3057        0.230397
      type=s
      3.4062        1.0
      type=s
      0.3245        1.0
      type=s
      0.1022        1.0
      type=p
      2.4360        0.110339
      0.6836        0.383111
      0.2134        0.647860
      type=p
      0.0701        1.0
$end
```

```
$title
  dzp h2o quadratic scf
 $end
 $route
 maxsiz=900000
 2s+1=3
 print=(basis,scf=vector)
 hf=quadratic scf=(pulay,core-fock)
 guess=(huckel)
 basis=dzp,sym=off
 $end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=2.949
r2=1.949
a1=112.3
$end
```

```
$title
 tz  scf  singlet-ivo  test run for h2 with singles ci
 $end
$route
hf basis=tz 2s+1=1 scf=(core-fock,pulay)
guess=huckel
print=(scf=(vector=all),ivo)
drt=(nrefs=1,ngroups=2,nsym=1)
ci=(nroots=5)
ivo=singlet
sym=off
$end

$drt
1typ1;1 5typ2;1
na=1 nb=0 ns=1
$end
$groups
numel=(1,1)
$end

$nonstd
1//1,2;
2//2;
3//2,12,30;
4//1;
5//1,2;
8//2,11;
9//2;
20//1;
$end
```

```
$geom
h1
h2 h1 r1

r1=0.75

$end
```

```
$title
 tz  scf  triplet-ivo test run for h2 with singles ci
 $end
$route
hf basis=tz 2s+1=1 scf=(core-fock,pulay)
guess=huckel print=(scf=(vector=all),ivo)
drt=(nrefs=1,ngroups=2,nsym=1)
ci=(nroots=5) sym=off
$end
 $drt
1typ1;1 5typ2;1
na=0 nb=2 ns=1
$end
$groups
numel=(1,1)
$end
$nonstd
1//1,2;
2//2;
3//2,12,30;
4//1;
5//1,2;
8//2,11;
9//2;
20//1;
$end
$geom
h1
h2 h1 r1

r1=0.75

$end
```

```
$title
dz  scf  singlet-ivo test run for h2o with singles ci
$end
$route
hf basis=dz 2s+1=1 scf=(core-fock,pulay)
guess=huckel print=(scf=(vector=all),ivo)
drt=(nrefs=1,ngroups=3,nsym=1)
ci=(nroots=5) ivo=singlet  sym=off
$end
$drt
4typ1;1 1typ2;1 9typ3;1
na=5 nb=0 ns=1
$end
$groups
numel=(8,1,1)
$end
$nonstd
1//1,2;
2//2;
3//2,12,30;
4//1;
5//1,2;
8//2,11;
9//2;
20//1;
$end
$geom
o1(dz)
h2(dz) o1 r1
h3(dz) o1 r2  h2 a1

r1=1.07
r2=1.07
a1=109.2

$end
```

```
$title
dz  scf  triplet-ivo test run for h2o singles ci
$end
$route
hf basis=dz 2s+1=1 scf=(core-fock,pulay)
guess=huckel  print=(scf=(vector=all),ivo)
drt=(nrefs=1,ngroups=3,nsym=1) ci=(nroots=5)
sym=off
$end

$drt
4typ1;1 1typ2;1 9typ3;1
na=4 nb=2 ns=1
$end
$groups
numel=(8,1,1)
$end

$nonstd
1//1,2;  2//2;  3//2,12,30;
4//1;
5//1,2;
8//2,11;  9//2;
20//1;
$end

$geom
o1(dz)
h2(dz) o1 r1
h3(dz) o1 r2  h2 a1

r1=1.07
r2=1.07
a1=109.2

$end
```

```
$title
  dzp   h2o   casscf test case   guess from r=1.949 scf
 $end
$route
maxsiz=1300000
2s+1=1
print=(basis,mcscf=vector)
mcscf=(cas,cycles=15,number-of-core-orbitals=3)
drt=(nbf=4,vexcite=99)
guess=(chk)
basis=dzp,sym=off
$end

$drt
2doc1 2uoc1
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=1.249
r2=2.949
a1=112.3
$end
```

```
$title
  dzp  h2o  general mcscf test case  guess from r=1.949 scf
  doubles from a single reference
 $end
$route
maxsiz=1300000
2s+1=1
print=(basis,mcscf=vector)
mcscf=(cycles=15,number-of-core-orbitals=3)
drt=(nbf=4,ngroups=2,nrefs=2)
guess=(chk)
basis=dzp  sym=off
$end

$drt
2typ1;1 2typ2;1
na=2 nb=0 ns=1
$end
$groups
numel=(4,0)
numel=(2,2)
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=1.249
r2=2.949
a1=112.3
$end
```

```
$title
  h2o  general mcscf test case  guess from r=1.949 scf
  2in2 x 2in2
 $end
$route
maxsiz=1300000
2s+1=1
print=(basis,mcscf=vector)
mcscf=(cycles=15,number-of-core-orbitals=3)
drt=(nbf=4,ngroups=2,nrefs=1)
guess=chk
basis=dzp,sym=off
$end

$drt
1typ1;1 2typ2;1 1typ1;1
na=2 nb=0 ns=1
$end
$groups
numel=(2,2)
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=1.249
r2=2.949
a1=112.3
$end
```

```
$title
 h2o  dzp  single-reference ci (srci)  test case
 $end
$route
 2s+1=1  print=basis
 hf  scf=(core-fock)
 guess=(huckel)
 basis=dzp  sym=off
 ci  drt=(vexcite=2,refexcite=99)
$end


drt for ci
$drt
1cor1 4doc1 20uoc1 1vir1
$end


$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=.959447
r2=.959447
a1=104.8318
$end
```

```
 $title
  dzp   h2o   multireference ci (mrci) test case   guess from r=1.949 scf
 $end
$route
2s+1=1  print=(basis,mcscf=vector,ci=no)
mcscf=(cas,cycles=15,number-of-core-orbitals=3)
drtmc=(nbf=4,vexcite=99)
ci  drt=(vexcite=2)
guess=(chk)
basis=dzp  sym=off
$end

$drtmc
2doc1 2uoc1
$end

$drt
1cor1 2doc1 2%doc1 2%uoc1 18uoc1 1vir1
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=.958
r2=.958
a1=107.3
$end
```

```
$title
  dzp h2o closed shell scf optimization
 $end
$route
maxsiz=900000
2s+1=1
hf scf=(pulay)
guess=(huckel)
opt=(berny)
basis=dzp sym=off
$end


$geom
o
h1 o r1
h2 o r2 h1 a1

r1=0.949
r2=0.949
a1=112.3
$end
```

```
$title
  dzp   h2o   high-spin   open shell scf   optimization
 $end
$route
maxsiz=900000
2s+1=3
hf scf=(pulay)
guess=(huckel)
opt=(berny)
basis=dzp  sym=off
$end


$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=0.949
r2=0.949
a1=112.3
$end
```

```
$title
  dzp  h2o  casscf  optimization
 $end
$route
maxsiz=1300000
2s+1=1
print=(basis,mcscf=vector)
mcscf=(cas,cycles=15,number-of-core-orbitals=3)
drt=(nbf=4,vexcite=99)
guess=chk=mcscf
basis=dzp  sym=off
opt=berny
$end

$drt
2doc1 2uoc1
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=1.249
r2=2.949
a1=112.3
$end
```

```
$title
 dzp  h2o  srci  optimization
 $end
$route
ciopt
2s+1=1
print=basis
hf scf=(core-fock)
ci transformation=scf
guess=(huckel)
m702=check-energy m812=(check-energy)
basis=dzp,sym=off
drt=(vexcite=2,refexcite=99)
opt=(berny,cycles=10,convf=1.e-4)
$end

drt for ci
$drt
1cor1 4doc1 20uoc1 1vir1
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=.949
r2=.949
a1=112.3
$end
```

```
$title
 dzp  h2o  mrci  optimization
 $end
$route
mrciopt transformation=scf
2s+1=1 m812=(check-energy)
mcscf=(cas,cycles=15,number-of-core-orbitals=3)
drtmc=(nbf=4,vexcite=99)
ci drt=(vexcite=2)
guess=chk=mcscf
opt=(berny,cycles=10)
basis=dzp,sym=off
$end

$drtmc
2doc1 2uoc1
$end
$drt
1cor1 2doc1 2%doc1 2%uoc1 18uoc1 1vir1
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=.958
r2=.958
a1=107.3
$end
```

```
$title
  dzp   h2o   finite difference force-constants   internal   one-point
 $end
$route
maxsiz=900000
2s+1=1
print=(basis,scf=vector)
hf  scf=(pulay)
guess=chk  geom=rdchk
opt=(force-constants)  frequencies
basis=dzp  sym=off
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=.996
r2=.996
a1=104.7
$end
```

```
$title
  dzp   h2o   finite difference force-constants   cartesian   two-point
 $end
 $route
 maxsiz=900000
 2s+1=1
 print=(basis,gradient=(total))
 hf scf=(pulay)
 guess=huckel
 opt=(force-constants) frequencies
 force-constants=(cartesian,two-point)
 basis=dzp sym=off
 $end


 $geom
 o
 h1  o  r1
 h2  o  r2  h1  a1

 r1=.944087
 r2=.944087
 a1=106.6001
 $end
```

```
$title
  h2o  dzp  casscf  finite-difference  force constants  internal  two-point
 $end
$route
maxsiz=1250000
basis=dzp
mcscf=(cas,number-of-core-orbitals=3,cycles=10,convergence=12)
drt=(nbf=4,vexcite=99)
guess=chk=mcscf  geom=rdchk
sym=off
print=gradient=total
opt=(force-constants)  frequencies
force-constants=(two-point)
$end

$drt
2doc1 2uoc1
$end

$geom
o1(dz)
h2(dz) o1 r1
h3(dz) o1 r2  h2 a1

r1=.979
r2=.979
a1=108.1

$end
```

```
$title
 dzp  h2o  srci  force-constants  internal  one-point
 $end
$route
ciopt
2s+1=1 print=basis
hf scf=(core-fock)
ci transformation=scf
guess=(huckel)
m702=check-energy m812=check-energy
basis=dzp,sym=off
drt=(vexcite=2,refexcite=99)
opt=(force-constants) frequencies
$end


drt for ci
$drt
1cor1 4doc1 20uoc1 1vir1
$end


$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=.959447
r2=.959447
a1=104.8318
$end
```

```
$title
  h2o mrci test case  force-constants  cartesian  two-point
 $end
$route
maxsiz=1250000
mrciopt transformation=scf
2s+1=1 m812=(check-energy)
print=gradient=total
mcscf=(cas,cycles=15,number-of-core-orbitals=3)
drtmc=(nbf=4,vexcite=99)
ci drt=(vexcite=2)
guess=chk=mcscf
opt=(force-constants) frequencies
force-constants=(two-point,cartesian)
basis=dzp,sym=off
$end

$drtmc
2doc1 2uoc1
$end
$drt
1cor1 2doc1 2%doc1 2%uoc1 18uoc1 1vir1
$end

$geom
o
h1  o  r1
h2  o  r2  h1  a1

r1=.961364
r2=.961364
a1=104.6598
$end
```

```
$title
 h2o  dzp  mrci  force-constants  frequencies  internal  one-point
 $end
$route
maxsiz=1250000
mrciopt transformation=scf
2s+1=1 m812=(check-energy)
mcscf=(cas,cycles=15,number-of-core-orbitals=3)
drtmc=(nbf=4,vexcite=99)
ci drt=(vexcite=2)
guess=chk=mcscf
opt=(force-constants) frequencies
basis=dzp sym=off
$end

$drtmc
2doc1 2uoc1
$end
$drt
1cor1 2doc1 2%doc1 2%uoc1 18uoc1 1vir1
$end

$geom
o
h1 o r1
h2 o r2 h1 a1

r1=.961364
r2=.961364
a1=104.6598
$end
```