T. D. Robb

InterCall

This manual describes InterCall – a *Mathematica* package that allows interactive calling of external code.

This document is a greatly abridged version of the full InterCall manual.

Dr. Terry Robb
Mathematics Department
Monash University
Clayton 3168
Melbourne VIC
Australia
Email: `tdr@vaxc.cc.monash.edu.au`

The distributor of InterCall **is:**
Analytica
PO Box 343
Subiaco 6008
Perth WA
Australia
Fax/Phone +61 9 388 2590
Email: `analytic@earwax.pd.uwa.oz.au`

For details about ordering InterCall contact the distributor.

Table of Contents

1 Introduction

1.1 What is InterCall?

InterCall is a *Mathematica* package that enables its user to interactively call external code, for example from an external numeric library. It allows various default settings to be defined for any of the arguments that need to be passed to any routine in that library, and initial default settings for all the routines in the IMSL and NAG libraries are included with InterCall in the form of a small defaults-database file.

InterCall's name is derived from its ability to Inter-actively Call external numeric libraries. In addition to this, InterCall provides other features that will allow you to import your own numerical routines. Those routines (written in a language such as FORTRAN, C, or Pascal) can then be called as if they were normal *Mathematica* functions.

InterCall even allows you to call external routines on a remote computer. To specify remote calling, you just type `InterCall["`*host*`"]` into *Mathematica*, where *host* is the hostname (typically the internet address) of the remote computer.

1.2 How Does One Use InterCall?

One common way to use InterCall is to interactively call routines in the IMSL or NAG libraries. For example the procedure to get a NAG routine to integrate *Mathematica*'s `Sin[`x`]` function from $x = 0$ to $x = \pi$, is as follows:

- read the on-line NAG manual and choose an integration routine, say D01AKF,
- then type, as at the lines *In*[3] and *In*[4] below,

```
In[3]:= GetDefault[ d01akf ]            (* this returns summary-help *)
       <detailed-help omitted>
Out[3]= d01akf[$F_, $A_, $B_] -> $RESULT

In[4]:= d01akf[ Sin[#]&, 0, Pi ]        (* this returns the integral *)
Out[4]= 2.
```

The `GetDefault[]` command above prints a detailed-help message, which can usually be ignored, plus it returns a short summary-help message written in the form of a *Mathematica* rule. This rule provides a convenient template for you to copy when you call the external routine in the last input step above.

Similarly the procedure to get a NAG routine to find the eigenvalues of some given band-symmetric matrix, is as follows:

- read the on-line NAG manual and choose a band matrix routine, say F02FHF,
- then type, as at the lines *In*[5], *In*[6] and *In*[7] below,

```
In[5]:= GetDefault[ f02fhf ]            (* this returns summary-help *)
       <detailed-help omitted>
Out[5]= f02fhf[$A_] -> $D

In[6]:= myBandSymmetricMatrix = {{0,-1,-1,-1,-1,-1},{1,2,3,4,5,6}};

In[7]:= f02fhf[ myBandSymmetricMatrix ] (* this returns eigenvalues *)
Out[7]= {0.253806820113374, 1.789472411695431, 2.964906355385796,
         4.035093644614204, 5.210527588304569, 6.746193179886627}
```

Documentation in the NAG manual will explain the meaning of the dummy arguments named A and D, which are mentioned in the rule on line *Out*[5] above.

1.3 Why Use InterCall?

InterCall extends the type of problems that *Mathematica* can solve. For example if you have reduced a symbolic problem down to solving a system of linear equations involving a large, but sparse, numeric matrix then you might choose to simply call an IMSL or NAG routine directly from within *Mathematica* in order to solve that system quickly and efficiently.

Typically a library routine will have many arguments (a dozen or more is not uncommon) but InterCall allows you to supply just a small number of arguments, with the rest being defaulted to some intelligent values. Those defaults can easily be inspected and modified with new commands called `GetDefault[]` and `SetDefault[]`.

A further feature of InterCall is that it completely integrates the symbolic capabilities of *Mathematica* with the numeric routines of any external library. You can therefore pass a *Mathematica* function, array, or any other expression, as an argument to an external routine and trust InterCall to send, where possible, the correct type of information to that external routine.

1.4 Adding Your Own Routines

The features of InterCall that allow you to access numerical libraries can also be used to import your own user-written functions into *Mathematica*. A simple example of how this can be done is as follows.

Suppose you have written in FORTRAN (or C, Pascal *etc.*) a Function, of one argument, called say NEWFUN. If you compile the source code and produce a UNIX object code file named say `"./mynewf.o"`, then the NEWFUN Function in that object code can be added into *Mathematica* using InterCall's command called `AddDefault[]`. For this example the syntax would be:

```
In[3]:= AddDefault[ NEWFUN[$X], RF[R], "./mynewf.o" ];
```

where the second argument, that is `RF[R]`, indicates that the external routine named NEWFUN is a Real-Function of a single Real argument. You can now use `newfun[`x`]` as if it were any other *Mathematica* function, thus

```
In[4]:= (1+newfun[3])^2
Out[4]= 64.
```

You should note that, by default, this new function has a completely lowercase name, as do all new functions that are imported into *Mathematica* with InterCall.

To give another example, consider the following FORTRAN Subroutine called NEWTON, which implements Newton-Raphson iteration to find a zero called X, of a function called $FUNC$, whose gradient function is called $GRAD$.

```
      subroutine NEWTON(X,FUNC,GRAD,EPS,MAXITS)
        integer  MAXITS,ITER
        real*8   X,FUNC,GRAD,EPS,XGUESS
        external FUNC,GRAD
        ITER   = 0
10      XGUESS = X
        ITER   = ITER + 1
        X      = XGUESS - FUNC(XGUESS)/GRAD(XGUESS)
        if ( abs(X-XGUESS).gt.EPS .and. ITER.lt.MAXITS ) goto 10
        if (ITER.ge.MAXITS) print *, 'NOT CONVERGING'
      end
```

If the above source code is compiled to produce the UNIX object code named say "./mynewt.o", then a new *Mathematica* function called newton[*guess*,*function*] can be added with

```
In[5]:= AddDefault[ NEWTON[$X->In:>Out,$FUNC->In,
                          $GRAD->$FUNC',$EPS->10^-9,$MAXITS->20],
                   S[R,RF[R],RF[R],R,I], "./mynewt.o" ];
```

Notice that sensible default values for the gradient function $GRAD$, and the control variables EPS and $MAXITS$, are given directly in the AddDefault[] command above. These defaults are set-up with the special operators -> and :>, and also with the special symbols In and Out. The precise syntax for setting default actions like the above will be described in detail later, but for now you should note that the -> operator is used in exactly the same way as it is used in *Mathematica*'s SetOptions[] command.

Although the above example is not a very efficient use of InterCall (it would be much easier to simply write the NEWTON Subroutine directly as a *Mathematica* function), it does however illustrate some of the features of InterCall.

As an example of using the external routine NEWTON, consider the equation $2\sin(x) - x \equiv 0$. To find a zero of that equation, starting with an initial guess of $x = 2$, just type as on the line $In[6]$ below

```
In[6]:= newton[ 2.0, 2*Sin[#]-# & ]
Out[6]= 1.895494267033981
```

where 2*Sin[#]-# & is one of *Mathematica*'s ways of writing an unnamed function of a single formal parameter. The line $Out[6]$ returns the zero that was found by the FORTRAN Subroutine NEWTON, using a default tolerance of $EPS = 10^{-9}$.

Similarly to find a zero of the first Bessel function near $x = 4.0$, just type

```
In[7]:= newton[ 4.0, BesselJ[1,#]& ]
Out[7]= 3.831705970207512
```

If you forget how to call the external routine NEWTON, then the GetDefault[] command will give some appropriate help messages. Thus

```
In[8]:= GetDefault[ newton ]
NEWTON[ (* TYPE=S *)
        $X -> In :> Out,                 (* DATA=R *)
        $FUNC -> In,                     (* DATA=RF[R] *)
        $GRAD -> Derivative[1][$FUNC],   (* DATA=RF[R] *)
        $EPS -> 10^(-9),                 (* DATA=R *)
        $MAXITS -> 20                    (* DATA=I *)
      ] (* CODE="./mynewt.o" *)
Out[8]= newton[$X_, $FUNC_] -> $X
```

Alternatively the summary-help message above could instead be retrieved by using *Mathematica*'s own Information[] operator "?", thus

```
In[9]:= ?newton
newton[$X_, $FUNC_] -> $X
```

which indicates that newton can be called with two arguments, the first of which will be modified and returned as the result.

1.5 Some Examples

This section gives some examples of using InterCall to call some NAG routines from within *Mathematica*.

In each of the following examples, InterCall has been started using:

```
In[1]:= <<InterCall.m;        (* Load in the InterCall package *)
In[2]:= <<InterData.m;        (* Load database for various libraries *)
```

The above two input lines are sufficient to allow dynamic importing of external routines from the local computer. See section 2.2 if you want to call external routines on a remote host computer.

The first time you call any external routine it will be dynamically imported into *Mathematica*. To *immediately* import a set of external routines however, such as the ones used in the examples below, you should type

```
In[3]:= Import[ {F01AAF,F04ADF,F03ADF,F02FHF,S18ACF,
                 C05NCF,X02AAF,E04LBF,D01AKF,D02EAF} ];
```

The above will perform all the imports at once, thus saving some time.

1. Inverse of a real matrix.

```
In[3]:= GetDefault[f01aaf]    (* finds inverse of a real matrix *)
In[4]:= a = {{ 33,  16,  72 },
            {-24, -10, -57 },
            { -8,  -4, -17 }};
In[5]:= f01aaf[a]                     (* returns inverse *)
In[6]:= a =.;
```

2. Inverse of a complex matrix.

```
In[3]:= GetDefault[f04adf]    (* finds inverse of a complex matrix *)
In[4]:= SetDefault[f04adf,$B->IdentityMatrix[$N]]
In[5]:= a = Table[ Random[] + I*Random[], {20}, {20} ]; (* a random matrix *)
In[6]:= ai= f04adf[a];                (* returns inverse *)
In[7]:= Max[Abs[ a.ai-IdentityMatrix[20] ]] (* check that it is the inverse *)
In[8]:= SetDefault[f04adf]; a =.; ai =.;
```

3. Determinant of a complex matrix.

```
In[3]:= GetDefault[f03adf]    (* finds determinant of a complex matrix *)
In[4]:= a = {{1,  1+2I,  2+10I},
            {1+I,  3I, -5+14I},
            {1+I,  5I, -8+20I}};
In[5]:= f03adf[a]                     (* returns determinant *)
In[6]:= a =.;
```

4. Eigenvalues of a band-symmetric matrix.

```
In[3]:= GetDefault[f02fhf]    (* finds eigenvalue of band matrix problem *)
In[4]:= SetDefault[f02fhf,$B->In]     (* allow user to input $B *)
In[5]:= a = {{0,0,-1,-1,-1,-1,-1}, {0,1,2,3,4,4,3}, {5,6,7,8,9,8,7}};
In[6]:= b = {{0,0,-2,-2,-2,-2,-2}, {0,2,1,1,1,1,1}, {4,5,6,6,6,6,6}};
In[7]:= f02fhf[a,b]                   (* returns eigenvalues of Ax = lam Bx *)
In[8]:= SetDefault[f02fhf]; a =.; b =.;
```

5. Calling special functions.

```
In[3]:= GetDefault[s18acf]    (* modified Bessel function K0(x) *)
In[4]:= s18acf[ 3.0 ]                 (* returns function value *)
In[5]:= Plot[ s18acf[x], {x,1,2} ];   (* plot a graph of it *)
```

■ 6. Zero of a system of non-linear equations.

Solve the system of equations $x_4^2 - x_1 \equiv 1$, $x_3^2 - x_2 \equiv 1$, $x_2^2 - x_3 \equiv 1$, $x_1^2 - x_4 \equiv 1$ using an initial guess of $(x_1, x_2, x_3, x_4) = (2, 1, 1, 2)$.

```
In[3]:= GetDefault[c05ncf]    (* finds a zero of a system of equations *)
In[4]:= Import[ {C05NCF,X02AAF} ];    (* import these routines right now *)
In[5]:= fcn = Function[{n,x,fvec,iflag}, fvec=Reverse[x^2]-x-1 ];
In[6]:= c05ncf[ fcn, {2,1,1,2} ]      (* returns a zero *)
In[7]:= ?c05ncf`*
In[7]:= c05ncf`nfev
In[8]:= c05ncf[ fcn, {2,1,1,2}, $XTOL->0.0001 ]
In[9]:= fcn =.;
```

■ 7. Minimization of a multivariate function, subject to constraints.

Minimize the function $(x+10y)^2+5(z-t)^2+(y-2z)^4+10(x-t)^4$ between the constraints $(1, -2, -10^6, 1) < (x, y, z, t) < (3, 0, 10^6, 3)$ using an initial guess of $(x, y, z, t) = (3, -1, 0, 1)$.

```
In[3]:= GetDefault[e04lbf]    (* minimizes a multivariate function *)
In[4]:= Import[ {E04LBF,X02AAF} ];    (* import these routines right now *)
In[5]:= SetDefault[e04lbf,$IBOUND->0,$BL->{1,-2,-10^6,1},$BU->{3,0,10^6,3}]
In[6]:= func=Function[{x,y,z,t}, (x+10y)^2 +5(z-t)^2 +(y-2z)^4 +10(x-t)^4];
In[7]:= e04lbf[ func, {3,-1,0,1}, $STEPMX->4.0 ]
In[8]:= Peek[$HESS]                   (* check out the Hessian of $FUNCT *)
In[9]:= SetDefault[e04lbf]; func =.;  (* e04lbf does 116 calls to $FUNCT *)
```

■ 8. Numerical integration of a function.

Integrate various functions in various different ways.

```
In[3]:= NagInt = d01akf;      (* define an alias for d01akf *)
In[4]:= GetDefault[NagInt]    (* integrates a function between limits *)
In[5]:= NagInt[ Sin[#]&, 0, Pi ]      (* integrate Sin from 0 to Pi *)
In[6]:= NagInt[ BesselJ[2,#]&, 0, 1 ] (* integrate 2'nd Bessel function *)
In[7]:= myfcn = Function[x, If[x<1,0,(x-1)^2/x^2] ];
In[8]:= NagInt[ myfcn, 0, 2 ]         (* integrate the above function *)
In[9]:= myfcn =.;
```

If you have the file "`./mycode.o`" in your current working directory (or in the top-level directory of the remote computer), then

```
In[10]:= AddDefault[ MYFUNC[$X], RF[R], "./mycode.o" ]; Import[{MYFUNC}];
In[11]:= NagInt[ MYFUNC, 0, 2 ]       (* integrate external function *)
```

■ 9. Solution of a system of first order ODE's.

Advance the system of ODE's $\dot{y}_1 = y_2$, $\dot{y}_2 = -y_1$, $\dot{y}_3 = ty_3$ from the initial condition $y_1 = 0$, $y_2 = 1$, $y_3 = 1$ at $t = 0$ to the solution at $t = 1$.

```
In[3]:= GetDefault[d02eaf]    (* solves ODE dy/dt = f[y,t], for y[t] *)
In[4]:= SetDefault[d02eaf,$TOL->10^-5]
In[5]:= t0 = 0; t1 = 1; y0 = {0,1,1};
In[6]:= mysub = Function[{t,y,f}, f = {y[[2]], -y[[1]], t*y[[3]]} ];
In[7]:= d02eaf[ t0, t1, y0, mysub ]   (* this call can be slow *)
```

If you have the file "`./mycode.o`" in your current working directory (or in the top-level directory of the remote computer), then

```
In[8]:= AddDefault[ MYSUBR[$X,$Y,$F], S[R,R[3],R[3]], "./mycode.o" ];
In[9]:= Import[ {MYSUBR} ];           (* use FORTRAN compiled code *)
In[10]:= d02eaf[ t0, t1, y0, MYSUBR ] (* this call is very fast *)
In[11]:= SetDefault[d02eaf]; t0 =.; t1 =.; y0 =.; mysub =.;
```

The following are some examples of using InterCall to call IMSL routines.

■ 1. Minimize a function, with constraints, using an IMSL routine.
Minimize the function $(x_1 - 1)^2 + 100(x_2 - x_1^2)^2$ between the constraints $(-2,-1) < (x_1, x_2) < (0.5, 2)$ using an initial guess of $(x_1, x_2) = (-1.2, 1)$.

```
In[3]:= GetDefault[dbconf]    (* minimizes a function with constraints *)
In[4]:= myfunc = Function[{n,x,f}, f=(x[[1]]-1)^2+100*(x[[2]]-x[[1]]^2)^2 ];
In[5]:= dbconf[ myfunc, {-1.2,1}, $XLB->{-2,-1}, $XUB->{0.5,2} ]
In[6]:= Peek[$IPARAM][[{3,4,5}]]       (* iters, func evals, grad evals *)
In[7]:= myfunc =.;
```

■ 2. Minimize a function, with constraints, using an IMSL routine.
Minimize the function $(x_1 - 1)^2 + 100(x_2 - x_1^2)^2$ between the constraints $(-2,-1) < (x_1, x_2) < (0.5, 2)$ using an initial guess of $(x_1, x_2) = (-1.2, 1)$. The analytic form of the function gradient can be used.

```
In[3]:= GetDefault[dbcong]    (* minimizes a function with constraints *)
In[4]:= myfunc = Function[{n,x,f}, Module[{x1,x2}, {x1,x2}=x;
                      f = (x1-1)^2 + 100*(x2-x1^2)^2 ] ];
In[5]:= mygrad = Function[{n,x,g}, Module[{x1,x2}, {x1,x2}=x;
                      g = { 2*(x1-1) - 400*x1*(x2-x1^2), 200*(x2-x1^2) } ] ];
In[6]:= dbcong[ myfunc, mygrad, {-1.2,1}, $XLB->{-2,-1}, $XUB->{0.5,2} ]
In[7]:= Peek[$IPARAM][[{3,4,5}]]       (* iters, func evals, grad evals *)
In[8]:= myfunc =.; mygrad =.;
```

■ 3. Numerical integration of a function, using an IMSL routine.
Compute the integral $\int_0^2 x e^x \, \mathrm{d}x$.

```
In[3]:= GetDefault[dqdag]     (* integrates a function between limits *)
In[4]:= f[x_] := x*Exp[x]
In[5]:= dqdag[ f, 0, 2 ]               (* integrates the function *)
In[6]:= dqdag`errest                   (* the error estimate *)
In[7]:= ClearAll[f];
```

■ 4. Solution of a system of first order ODE's, using an IMSL routine.
In the object file "`./mycode.o`", a Subroutine called MYCHAOS has been written to represent the Lorentz Attractor equations: $\mathrm{d}y_1/\mathrm{d}x = -3(y_1 - y_2)$, $\mathrm{d}y_2/\mathrm{d}x = -y_1 y_3 - y_2 + 26.5 y_1$, $\mathrm{d}y_3/\mathrm{d}x = y_1 y_2 - y_3$. Use that Subroutine to compute a table of points giving the solution for $x = 30$ through to $x = 50$ in steps of 0.1 (line *In*[7]), starting with the initial condition $y = (0, 1, 0)$ for $x = 0$ (line *In*[6]). Also plot a graph of that solution (line *In*[9]). Consult the IMSL documentation for DIVPRK to find out the meaning of the dummy argument *IDO*, and also to understand how DIVPRK is called.

```
In[3]:= GetDefault[divprk]    (* solves an initial-value problem *)
In[4]:= AddDefault[ MYCHAOS, S, "./mycode.o" ];  (* chaotic equations *)
In[5]:= Import[ {MYCHAOS,DIVPRK} ];              (* import routines now *)
In[6]:= divprk[ 20.0, $IDO->1,         (* first call: power-up divprk *)
                     $FCN->MYCHAOS, $X->0, $Y->{0,1,0}, $PARAM->0 ];
In[7]:= points = Table[ divprk[xend], {xend,30,50,0.1} ];
In[8]:= divprk[ 60.0, $IDO->3 ];       (* last call: power-down divprk *)
In[9]:= Show[ Graphics3D[ {Line[points]} ], Axes->Automatic ];
```

2 InterCall

Obtaining InterCall

To obtain InterCall, type the following two lines of input into your copy of *Mathematica* and send the output lines off to the distributor of InterCall:

```
In[1]:= { $MachineType, $MachineName, $MachineID }//InputForm
In[2]:= { $System, $OperatingSystem }//InputForm
```

For example the output lines might be something like:

```
Out[1]//InputForm= {"DEC RISC", "solar", "2312-51366-59971"}
Out[2]//InputForm= {"DEC_RISC", "Unix"}
```

The above information will enable the distributor to send you a customized disk containing all the files needed to run InterCall on your computer. A *Mathematica* package called `InterCall_`*XXXX*`.m` (where *XXXX* represents the `$MachineName` of the computer on which you run your *Mathematica* kernel) will be customized so that it can be loaded only into your copy of *Mathematica*. If you have a license to run the same copy of the *Mathematica* kernel on several different computers, then you will need to have several different `InterCall_`*XXXX*`.m` packages – one for each `$MachineName`.

With InterCall you will also receive a disk with copies of a remote driver for many computer systems including Apollo, Connection Machine, DEC/Ultrix, IBM RISC, Iris, Sony, Sparc, Sun-3 and VAX/VMS. If you install a remote driver onto a remote computer, then you can call external routines on that computer. From *Mathematica* the syntax for connecting to a remote computer, with hostname *host*, is simply to type the InterCall command `InterCall["`*host*`"]`.

InterCall uses standard protocols to connect to the remote computer and in particular if your local computer (*i.e.* the one that runs the *Mathematica* kernel) can `rlogin` to a remote computer then you should be able to use InterCall's remote driver on that computer.

4 Datatypes

4.1 Datatype Structure

In the `AddDefault[]` command, and also in the detailed-help message returned by the `GetDefault[]` command, the datatype structure of any dummy argument or external routine is declared by using either the letters `R`, `I`, `C`, `L`, `Z`, the pair of letters `RF`, `IF`, `CF`, `LF`, `ZF` or the letter `S`, in a recursive definition. The initial letters `R`, `I`, `C`, `L`, `Z`, stand respectively for the FORTRAN datatypes of Real*8, Integer*4, Character*(*), Logical*4, Complex*16, while the letter `S` stands for a Subroutine, and the `F` specifies a Function. Thus, for example, the datatype `RF[R,I[4],R[5,6]]` would mean a Real*8 Function that requires a Real*8 number, a 4-vector of Integer*4 numbers and a 5x6 array of Real*8 numbers. Similarly `S[C,Z]` represents a Subroutine that requires a Character*(*) string and a Complex*16 number.

A datatype structure may reference named dummy arguments, for example `RF[I,R[$N]]` where `$N` is the dummy name of another argument. A datatype structure may also reference unnamed arguments as in `RF[I,R[#1]]` where `#1` represents the first unnamed Integer argument.

Unnamed argument referencing is useful for declaring nested datatype structures. For example in `S[R[5],RF[I,R[#1]]]`, the scope of `#1` will pick up the value of the first Integer. Without unnamed referencing there would be no way of getting the value of that Integer. In general you can use the formal parameter `#`n, for n a positive integer, to reference the n'th unnamed argument.

4.2 Coercing

Any argument, that needs to be passed to an external routine, can be given in standard *Mathematica* syntax. InterCall applies some natural datatype coercions to those passed arguments, so that for example a *Mathematica* list can be used to represent a FORTRAN array. The following is a table of some natural coercions.

FORTRAN Datatype	InterCall Datatype	*Mathematica* input	input example
real*8 X	`R`	`_?NumberQ`	`1.0`
integer*4 N	`I`	`_?NumberQ`	`2`
character*(*) S	`C`	`_String`	`"Hello"`
logical*4 B	`L`	`True|False`	`True`
complex*16 W	`Z`	`_?NumberQ`	`2+3*I`

real*8 V(4)	`R[4]`	`_Function`
		`_List`
		`_?NumberQ`
			`(#^2)&`
			`{1,2,3,4}`
			`0.0001`

real*8 A(2,3)	`R[2,3]`	`_Function`
		`_List`
		`_?NumberQ`
			`(#1^2+#2^2)&`
			`{{1,2,3},{4,5,6}}`
			`0.000001`

real*8 function F(X)	`RF[R]`	`_Function`	`Sin[#]&`
subroutine SUB(X,F)	`S[R,RF[R]]`	`_Function`	`(#1=#2[#1])&`

A Sample of Some Datatype Coercions

In general any *Mathematica* expression can be passed as an argument to an external routine. The expression will first get *Mathematica*'s numerical operator `N[]` applied to it, so that for example you can pass `Pi+1/2` when a real number is expected by the external routine.

For 1-dimensional FORTRAN arrays, you can pass a *Mathematica* list, so that for example `{1,2,3,4}` can be used to represent a 4 element FORTRAN vector. Similarly for 2-dimensional FORTRAN arrays, you can pass a list of lists and in general for an n-dimensional FORTRAN array, you can pass a *tensor*, provided `TensorRank[`*tensor*`]` is equal to n. As an example the expression `{{1,2,3},{4,5,6}}` could be used to represent a 2x3 FORTRAN matrix. It is also possible to pass just a single number, even if the expected datatype is an n-dimensional FORTRAN array. The number is simply used to fill out the array.

For FORTRAN Functions or Subroutines, you can pass a user-written *Mathematica* function and InterCall will use that function to emulate the action of a FORTRAN Function or Subroutine. This feature will be described in detail later in the chapter on emulation.

A very useful feature is that you can also pass the literal name of a dummy argument or even the literal name of an external routine. As an example of passing the literal name of an external routine, consider the root finding routine BISECT, defined in the object code file `"./mycode.o"`. It expects to receive, as it's first argument, a function with a datatype structure of `RF[R]`. A default-entry for the BISECT routine could be added with:

```
In[3]:= AddDefault[ BISECT[$FUNC->In,$X1->In,$X2->In,
                           $XACC->($X1+$X2)/10^6,$MAXBIS->40],
                    RF[ RF[R], R, R, R, I ], "./mycode.o" ];
```

You can pass the literal name of a FORTRAN routine, such as NEWFUN, as the first argument to BISECT, and the routine BISECT will be able to call that routine without involving *Mathematica* at all. The routine NEWFUN, which is defined in the object code file `"./mynewf.o"`, should have first had a default-entry added by

```
In[4]:= AddDefault[ NEWFUN[$X->In], RF[R], "./mynewf.o" ];
```

With the above declaration, you can find a root of the FORTRAN Function NEWFUN using the root finding routine BISECT as follows:

```
In[5]:= Import[ {BISECT,NEWFUN} ];      (* import these routines now *)

In[6]:= bisect[NEWFUN, 0, 2]            (* search between 0 and 2 *)
Out[6]= 1.414213180541992

In[7]:= bisect[NEWFUN, 0, 2, $MAXBIS->50, $XACC->10^-14]
Out[7]= 1.414213562373092
```

In the above example, InterCall is only needed to fill in the extra arguments, which it does by using the default-entry for the routine BISECT. Once InterCall has done that then all the work (*i.e.* function calls to NEWFUN) is done in FORTRAN and so the above call to BISECT is very efficient. It is as efficient as coding the entire problem directly in FORTRAN.

■ 6.4 A Sample Database

The following sample defaults-database file, named `sample.m`, defines default-entries for a small selection of routines in the IMSL and NAG libraries. (InterCall comes with a much bigger defaults-database file called `InterData.m`.) To load such a database type:

```
In[1]:= <<sample.m;
```

To debug a list of default-entries, such as the one below, just type

```
In[2]:= sampleRoutines = List @@ Apply[ AddDefault, $DefaultEntries, {1} ];
```

which will print a message if there is any syntax error in any of the entries.

```
Needs["InterCall`"];
$DefaultEntries = $DefaultEntries ~Join~ HoldList[
{ C05AXF[ $X->In:>In, $FX->In, $TOL->10^(-6), $IR->0,
        $SCALE->$X*Sqrt[x02aaf[]], $C:>Null, $IND->In:>In, $IFAIL->-1 ],
  S[ R, R, R, I, R, R[26], I, I ] },
{ C05NCF[ $FCN->In, $N->ROWS[$X], $X->In:>Out, $FVEC:>c05ncf`fvec,
        $XTOL->Sqrt[x02aaf[]], $MAXFEV->200*($N+1), $ML->($N-1),
        $MU->($N-1), $EPSFCN->0.0, $DIAG->1.0:>c05ncf`diag, $MODE->0,
        $FACTOR->100.0, $NPRINT->0, $NFEV:>c05ncf`nfev,
        $FJAC:>c05ncf`fjac, $LDFJAC->$N, $R:>Null, $LR->$N*($N+1)/2,
        $QTF:>Null, $W:>Null, $IFAIL->-1 ],
  S[ S[I,R[$N],R[$N],I], I, R[$N], R[$N], R, I, I, I, R, R[$N], I, R,
        I, I, R[$LDFJAC,$N], I, R[$LR], I, R[$N], R[$N,4], I ] },
{ C05PCF[ $FCN->In, $N->ROWS[$X], $X->In:>Out, $FVEC:>c05pcf`fvec,
        $FJAC:>c05pcf`fjac, $LDFJAC->$N, $XTOL->Sqrt[x02aaf[]],
        $MAXFEV->100*($N+1), $DIAG->1.0:>c05pcf`diag, $MODE->0,
        $FACTOR->100.0, $NPRINT->0, $NFEV:>c05pcf`nfev,
        $NJEV:>c05pcf`njev, $R:>Null, $LR->$N*($N+1)/2, $QTF:>Null,
        $W:>Null, $IFAIL->-1 ],
  S[ S[I,R[$N],R[$N],R[$LDFJAC,$N],I,I], I, R[$N], R[$N], R[$LDFJAC,$N],
        I, R, I, R[$N], I, R, I, I, I, R[$LR], I, R[$N], R[$N,4], I ] },
{ C06EAF[ $X->In:>Out, $N->ROWS[$X], $IFAIL->-1 ],
  S[ R[$N], I, I ] },
{ D01AKF[ $F->In, $A->In, $B->In, $EPSABS->0.0, $EPSREL->0.00001,
        $RESULT:>Out, $ABSERR:>d01akf`abserr, $W:>Null, $LW->2000,
        $IW:>Null, $LIW->$LW/8+2, $IFAIL->-1 ],
  S[ RF[R], R, R, R, R, R, R, R[$LW], I, I[$LIW], I, I ] },
{ D01FCF[ $NDIM->Min[ROWS[$A],ROWS[$B]], $A->In, $B->In,
        $MINPTS->1:>d01fcf`minpts, $MAXPTS->100, $FUNCTN->In,
        $EPS->0.00001, $ACC:>d01fcf`acc,
        $LENWRK->($NDIM+2)*(1+$MAXPTS/(2^$NDIM+2*$NDIM^2+2*$NDIM+1)),
        $WRKSTR:>Null, $FINVAL:>Out, $IFAIL->-1 ],
  S[ I, R[$NDIM], R[$NDIM], I, I, RF[I,R[$NDIM]], R, R, I, R[$LENWRK],
        R, I ] },
{ D02EAF[ $X->In, $XEND->In, $N->ROWS[$Y], $Y->In:>Out,
        $TOL->10^(-6), $FCN->In, $W:>Null, $IW->18+$N, $IFAIL->-1 ],
  S[ R, R, I, R[$N], R, S[R,R[$N],R[$N]], R[$N,$IW], I, I ] },
{ E04DGF[ $N->ROWS[$X], $OBJFUN->In, $ITER:>Null, $OBJF:>Out, $OBJGRD:>Null,
        $X->In:>Out, $IWORK:>Null, $WORK:>Null, $IUSER->0:>Null,
        $USER->0:>Null, $IFAIL->-1 ],
  S[ I, S[I,I,R[$N],R,R[$N],I,I[1],R[1]], I, R, R[$N], R[$N], I[$N+1],
        R[12*$N], I[1], R[1], I ] },
{ E04DJF[ $IOPTNS->In, $INFORM:>Out ], S[ I, I ] },
{ E04DKF[ $STRING ], S[ C ] },
```

```
{ E04LBF[ $N->ROWS[$X],
        $FUNCT->( Function[{$iflag,$n,$xc,$fc,$gc,$iw,$liw,$w,$lw},
                        $fc=In@@$xc; $gc=Through[GC@@$xc]; ]
          /. GC->Table[ Partial[$i][Ins[1]], {$i,1,$N} ] ),
        $HESS->( Function[{$iflag,$n,$xc,$fhesl,$lh,$fhesd,$iw,$liw,$w,$lw},
                        $fhesl=Through[HL@@$xc]; $fhesd=Through[HD@@$xc]; ]
          /. HL->Flatten @ Table[ Partial[$i,$j][Ins[1]],
                                        {$i,1,$N-1},{$j,$i+1,$N} ]
          /. HD->Table[ Partial[$i,$i][Ins[1]], {$i,1,$N} ] ),
        $MONIT->( Function[{$n,$xc,$fc,$gc,$istate,$gpjnrm,$cond,$posdef,
                         $niter,$nf,$iw,$liw,$w,$lw}, Null ] ),
        $IPRINT->-1, $MAXCAL->50*$N, $ETA->0.9, $XTOL->10.0*Sqrt[x02aaf[]],
        $STEPMX->100000.0, $IBOUND->1, $BL->-10^6, $BU->10^6, $X->In:>Out,
        $HESL:>e04lbf`hesl, $LH->Max[$N*($N-1)/2,1], $HESD:>e04lbf`hesd,
        $ISTATE:>e04lbf`istate, $F:>Out, $G:>e04lbf`g, $IW:>Null, $LIW->2,
        $W:>Null, $LW->Max[7*$N+$N*($N-1)/2,8], $IFAIL->-1 ],
  S[ I, S[I,I,R[$N],R,R[$N],I[$LIW],I,R[$LW],I],
        S[I,I,R[$N],R[$LH],I,R[$N],I[$LIW],I,R[$LW],I],
        S[I,R[$N],R,R[$N],I[$N],R,R,L,I,I,I[$LIW],I,R[$LW],I],
        I, I, R, R, R, I, R[$N], R[$N], R[$N], R[$LH], I, R[$N], I[$N],
        R, R[$N], I[$LIW], I, R[$LW], I, I ] },
{ F01AAF[ $A->In, $IA->$N, $N->COLS[$A], $UNIT:>Out, $IUNIT->$N,
        $WKSPCE:>Null, $IFAIL->-1 ],
  S[ R[$IA,$N], I, I, R[$IUNIT,$N], I, R[$N], I ] },
{ F01AGF[ $N->COLS[$A], $TOL->x02adf[], $A->In:>Null, $IA->$N,
        $D:>Out, $E:>Out, $E2:>Null ],
  S[ I, R, R[$IA,$N], I, R[$N], R[$N], R[$N] ] },
{ F02FHF[ $N->COLS[$A], $MA->$NRA-1, $A->In, $NRA->ROWS[$A], $MB->$NRB-1,
        $B->Table[1,{1},{$N}], $NRB->ROWS[$B], $D:>Out, $WORK:>Null,
        $LWORK->Max[$N,(3*$MA+$MB)*($MA+$MB+1)], $IFAIL->-1 ],
  S[ I, I, R[$NRA,$N], I, I, R[$NRB,$N], I, R[$N], R[$LWORK], I, I ] },
{ F03ADF[ $A->In, $IA->$N, $N->COLS[$A], $DETR//(#+I*Peek[$DETI])&:>Out,
        $DETI:>Null, $WKSPCE:>Null, $IFAIL->-1 ],
  S[ Z[$IA,$N], I, I, R, R, R[$N], I ] },
{ F04ADF[ $A->In, $IA->$N, $B->In, $IB->$N, $N->COLS[$A], $M->COLS[$B],
        $C:>Out, $IC->$N, $WKSPCE:>Null, $IFAIL->-1 ],
  S[ Z[$IA,$N], I, Z[$IB,$M], I, I, I, Z[$IC,$M], I, R[$N], I ] },
{ S18ACF[ $X->In, $IFAIL->-1 ], RF[ R, I ] },
{ X02AAF[ $DUMMY:>Null ], RF[ R ] }, { X02ADF[ $DUMMY:>Null ], RF[ R ] },
{ DBCONF[ $FCN->In, $N->ROWS[$XGUESS], $XGUESS->In, $IBTYPE->0,
        $XLB->-10^6, $XUB->10^6, $XSCALE->1.0, $FSCALE->1.0,
        $IPARAM->0, $RPARAM->0.0, $X:>Out, $FVALUE:>Out ],
  S[ S[I,R[$N],R], I, R[$N], I, R[$N], R[$N], R[$N], R, I[7], R[7], R[$N], R ]},
{ DBCONG[ $FCN->In, $GRAD->In, $N->ROWS[$XGUESS], $XGUESS->In, $IBTYPE->0,
        $XLB->-10^6, $XUB->10^6, $XSCALE->1.0, $FSCALE->1.0,
        $IPARAM->0, $RPARAM->0.0, $X:>Out, $FVALUE:>Out ],
  S[ S[I,R[$N],R], S[I,R[$N],R[$N]], I, R[$N], I, R[$N], R[$N], R[$N], R,
        I[7], R[7], R[$N], R ] },
{ DIVPRK[ $IDO->2, $NEQ->ROWS[$Y], $FCN->$FCN, $X->$X, $XEND->In,
        $TOL->0.0005, $PARAM:>Null, $Y->$Y:>Out ],
  S[ I, I, S[I,R,R[$NEQ],R[$NEQ]], R, R, R, R[50], R[$NEQ] ] },
{ DQDAG[ $F->In, $A->In, $B->In, $ERRABS->0.0, $ERRREL->0.00001,
        $IRULE->1, $RESULT:>Out, $ERREST:>dqdag`errest ],
  S[ RF[R], R, R, R, R, I, R, R ] },
{ ERSET[ $IERSVR->0, $IPACT->1, $ISACT->0 ], S[ I, I, I ] }
] ; (* end of HoldList *)
```

■ Appendix

■ A.1 Description of The InterCall Commands

■ 1. GetDefault

■ `GetDefault[` *routine* `]`

This prints the detailed-help and summary-help messages for the *routine*. Use of this command, together with the external routine's documentation, should give you all the information that you need in order to call the routine. The external routine's uppercase name, that is *Routine*, will be searched for in a pre-loaded defaults-database if you have not already explicitly added the routine with use of the `AddDefault[]` command. The detailed-help message gives a list of default-settings (see `SetDefault[]` below) for each dummy argument, plus their datatypes. The summary-help message is written in the form of a *Mathematica* rule which indicates how to call the external routine with the defaults described in the detailed-help message. The summary-help message can also be printed by typing `?`*routine*. The `GetDefault[]` command is similar to *Mathematica*'s `Options[]` command.

E.g. `In[3]:= GetDefault[d01akf]`

E.g. `In[4]:= ?d01akf`

■ 2. SetDefault

■ `SetDefault[` *routine*, $Setting_1$ <,$Setting_2$ ...> `]`

This allows the default-entry for the given *routine* to be changed. Each of the *Setting*'s is just a default-setting which can take any one of the following forms:

`$`*Symbol*	`(* equivalent to $`*Symbol*`->In *)`
`$`*Symbol*`->`*input*	`(* set a new input-value for $`*Symbol* `*)`
`$`*Symbol*`:>`*output*	`(* set a new output-name for $`*Symbol* `*)`
`$`*Symbol*`->`*input*`:>`*output*	`(* set a new input-value, output-name *)`
`$`*Symbol*`//`*function*`:>`*output*	`(* output-name will hold: *)`
`$`*Symbol*`->`*input*`//`*function*`:>`*output*	`(*` *function*`[$`*Symbol*`] *)`

The usage of the `:>` operator in this context is *not* exactly the same as its usage in standard *Mathematica*. In the context of a default-setting it means set a default output-name for the corresponding dummy uppercase argument, named *Symbol*. As in standard *Mathematica* syntax the `->` operator however still means set a default initial value, and the `//` operator still means apply a function. Initial default-settings for all the arguments of a routine can be shown by the `GetDefault[`*routine*`]` command. The `SetDefault[]` command is similar to *Mathematica*'s `SetOptions[]` command.

E.g. `In[5]:= SetDefault[d01akf, $EPSREL->10^-5]`

E.g. `In[6]:= SetDefault[d01akf, $EPSREL]`

■ `SetDefault[` *routine* `]`

This resets to the initial set of defaults that were first given for the *routine*.

E.g. `In[7]:= SetDefault[d01akf]`

■ 3. AddDefault

■ `AddDefault[` *Routine* `,` *Type* <`,`*objectcode*> `]`

This declares that an external routine named *Routine* is to be found in the given *objectcode*. If no *objectcode* is given then the routine is assumed to be in a standard default library as defined in the remote driver file. The *Type* can be either of `RF`, `IF`, `CF`, `LF`, `ZF` or `S`, standing respectively for the FORTRAN types of Real*8 Function, Integer*4 Function, Character*(*) Function, Logical*4 Function, Complex*16 Function or Subroutine. The external routine will be imported into *Mathematica* if it is ever needed.

```
E.g.   In[3]:= AddDefault[MYFUNC, RF, "./mycode.o"]
E.g.   In[4]:= AddDefault[MYSUBR,  S, "./mycode.o"]
E.g.   In[5]:= AddDefault[DGAMMA, RF, "-limsl"]
E.g.   In[6]:= AddDefault[S18ACF,  S]
```

■ `AddDefault[` *Routine*`[`$Setting_1$`,...]`, *Type*`[`$Datatype_1$`,...]` <`,`*objectcode*>`]`

If a sequence of default-settings and their datatypes are also given in the `AddDefault[]` command then the external routine can be called directly from *Mathematica*, and not just passed as an argument, to another external routine, as in the previous form.

```
E.g.   In[3]:= AddDefault[MYFUNC[$X],        RF[R],             "./mycode.o"]
E.g.   In[4]:= AddDefault[MYSUBR[$X,$Y,$F], S[R,R[3],R[3]], "./mycode.o"]
E.g.   In[5]:= AddDefault[DGAMMA[$X->In],    RF[R],             "-limsl"]
E.g.   In[6]:= AddDefault[S18ACF[$X,$IFAIL -> -1 :> ifail], S[R,I]]
```

■ `AddDefault[ $`*Symbol*`->`*value* `,` *Datatype* `]`

This usage of `AddDefault[]` allows external symbols to be created and initialized to any value.

```
E.g.   In[3]:= AddDefault[$MYSCALAR->0.5,                R     ]
E.g.   In[4]:= AddDefault[$MYVECTOR->{5,2,2,7},          R[4]  ]
E.g.   In[5]:= AddDefault[$MYMATRIX->{{1,2,3},{4,5,6}}, R[2,3]]
E.g.   In[6]:= AddDefault[$MYFUNCTION->(Tan[#]&),        RF[R] ]
E.g.   In[7]:= AddDefault[$WORKSPACE->None,              R[100]]
```

■ 4. Import

■ `Import[` *Match* `]`

This does a wildcard search of the defaults-database for all routines that match the string *Match*, and then adds the default-entry for those routines into *Mathematica*. If the *Match* does not contain the wildcard characters of '`*`' or '`@`' then it is not necessary to enclose the *Match* in double quotes.

```
E.g.   In[3]:= Import["X02*"]
E.g.   In[4]:= Import[S18ACF]
```

■ `Import[ {`$Match_1$ <`,`$Match_2$ ...>`} ]`

If the argument to `Import[]` is a list of routines, then all those routines will be searched for and then imported into *Mathematica* at the same time, thus saving a considerable overhead in the cases where you plan to import many external routines. This command is the most common way of importing routines using InterCall.

```
E.g.   In[5]:= Import[{S18ACF,"X02*",MYFUNC,MYSUBR}]
```

5. Peek

Peek[$\$Symbol$]

This is a way to access the external data structure that is created when an external routine is called. Dummy arguments passed to and from any external routine always use the external data structure, and so any argument can always be peeked into, no matter what the current default-setting is for that argument. The values actually passed to an external routine can also be confirmed with this command.

```
E.g.   In[11]:= Peek[$MYSCALAR]        (* returns its value *)
E.g.   In[12]:= Peek[$MYVECTOR]        (* returns vectors value *)
E.g.   In[13]:= Peek[$MYMATRIX]        (* returns entire matrix *)
E.g.   In[14]:= Peek[$MYFUNCTION]      (* returns function definition *)
E.g.   In[15]:= Peek[D01AKF]           (* returns -ExternalCode- *)
```

Peek[n]

When `-ExternalCode-` is applied to a set of unnamed arguments it may modify some of their values. To see those modified values, you needs a way to refer to the n'th unnamed argument, and the command `Peek[`n`]` with n a positive integer allows you to do that. This feature is possible because unnamed arguments are put onto a temporary stack, which is not immediately purged.

```
E.g.   In[17]:= Peek[5]        (* will look into 5'th stack argument *)
```

6. Poke

Poke[$\$Symbol$, $value$]

This is a way to modify the external data structure in-between calls.

```
E.g.   In[18]:= Poke[$MYSCALAR, 0.5      ]   (* sets the scalar *)
E.g.   In[19]:= Poke[$MYVECTOR, {5,2,2,7}]   (* sets value of vector *)
```

7. Purge

Purge[$Match$]

This purges (that is removes) all the values and external structures that are associated with all dummy arguments, that match the string *Match*. You can also purge lowercase routine names too, and this frees up internal InterCall structures.

```
E.g.   In[20]:= Purge["$MY*"]          (* purges $MYSCALAR, etc *)
E.g.   In[21]:= Purge[d01akf]          (* purges D01AKF and d01akf *)
E.g.   In[22]:= Purge["$*"]            (* purges every dummy name *)
```

Other Commands

`InterCall[]` allows interactive calling of code on the local computer.

`InterCall["`*host*`"]` allows interactive calling of code on a remote *host*.

`InterCall[Off]` shuts down an InterCall connection.

`InterCall[On]` starts up an InterCall connection immediately.

`ROWS[`*tensor*`]` gives the number of rows in *tensor*.

`COLS[`*tensor*`]` gives the number of columns in *tensor*.

`DIMS[`*d*`][`*tensor*`]` gives the size of the d'th dimension.