ElmerParam Manual

Erik Edelmann and Peter Råback
CSC – IT Center for Science

2006–2026

Copyright

1 Introduction

ElmerParam is a simple tool for parametrized computation for software that uses ASCII-format input and output files. It was primarily designed to be used with Elmer, but can easily be used with other software packages as well. The parametrized approach is particularly useful for optimization purposes.

ElmerParam exists both as a standalone program and as a library with bindings for C, Fortran, R and Matlab.

In this document, the following designations are used:

`[whatever]`	`whatever` is optional.
$A\|B$	either A xor B.
`${ELMER_HOME}`	Environment variable supposed to be set to the directory where Elmer (including ElmerParam) was installed.
ElmerParam	(in normal font) The ElmerParam package.
`ElmerParam`	(`this font`) The standalone `ElmerParam` program.

2 General overview

ElmerParam acts a layer between computational programs such as ElmerSolver and programs that wants to call the computational programs as func-

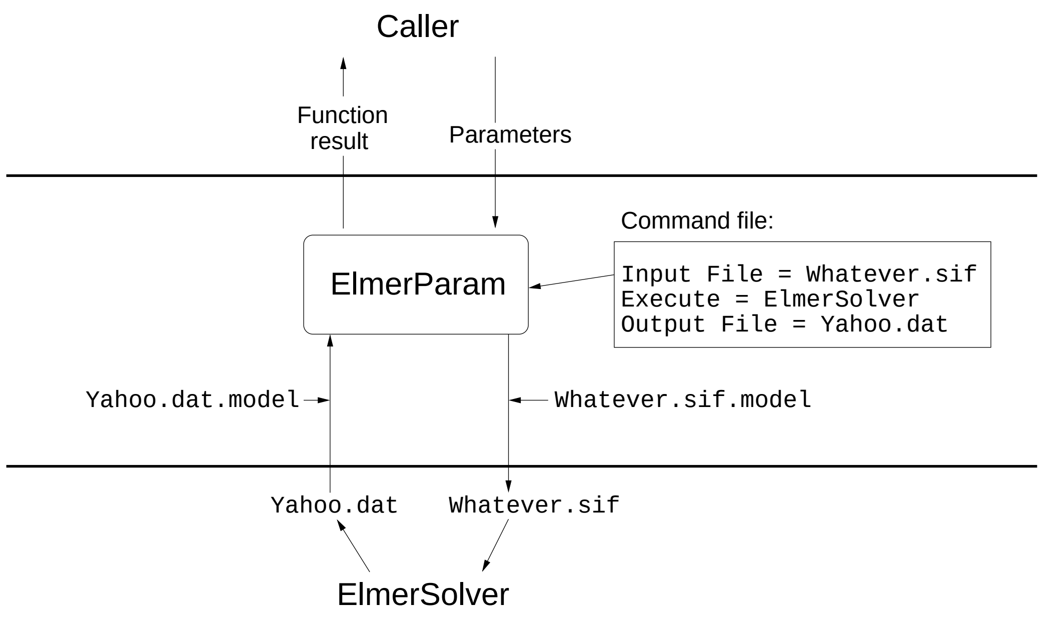

Figure 1: ElmerParam as a layer between ElmerSolver and a program that wants to call ElmerSolver as function of some parameters.

tions of some parameters. It provides functions which take real and/or integer parameters as arguments, and return a scalar or vector of real output values:

$$f : \mathbf{R}^n \times \mathbf{Z}^m \longrightarrow \mathbf{R}$$

or

$$f : \mathbf{R}^n \times \mathbf{Z}^m \longrightarrow \mathbf{R}^k$$

where n is the number of real parameters, m the number of integer parameters, and k the length of the return vector. These functions are from the caller's point of view black boxes and therefore no information about the case is provided. This information is instead provided in a set of files read by ElmerParam.

The most important file is a ElmerParam command file, which contains the commands to run the computation programs. A typical command file will create input files with parameter values for the computation program, execute the computation program, and read parameter values from it's output files. The input files are created and output files interpreted with the help of template (model) files. Simple calculations specified in the MATC language can be performed by ElmerParam itself (see section 3.1.2).

In addition to computation parameters, ElmerParam can also be given a special “tag” parameter. The tag can be expanded not only in the input files, but also be used in the command file. It can be used to give unique names to all the input and output files, which can be important when running several instances of ElmerParam in parallel.

3 ElmerParam Files

ElmerParam routines need a number of different files with information on the problem to be solved.

- The `ELMERPARAM_STARTINFO` file has a fixed name and has only one line, the name of the ElmerParam command file.
- ElmerParam command file is the file containing the execution instructions needed for the evaluation of the function return value.
- Each case needs a set of model files that are generalized versions of the input and output files. In the model files the parameters are given in brackets. For example the first real valued parameter is inserted where `<!R1!>` occurs. Input file models are used to create input files for the computation of, for example, ElmerGrid or ElmerSolver. Output file models are used to read result information from the output file.

3.1 ElmerParam command file

The ElmerParam command file includes some simple statements that are run in the order of appearance. The general form of a statement is

```
Command = argument
```

Commands are case insensitive. Lines starting with `#`, `!` or `*` are ignored. A statement can be split onto several lines by putting a ’\’ as the last character on a line. `<!T!>` in an argument will be expanded to the value of the tag parameter.

3.1.1 Commands:

`Comment = string`

Echo `string` to stdout.

`Echo = True|False`

Turn ECHO on|off. If 'on', all commands are echoed to stdout when executed. 'On' by default.

`Matc = True|False`

Turn MATC support on|off. Has to be 'on' if you use MATC expressions. Requires that ElmerParam was compiled with MATC support. Off by default.

`Input File = fname1 [Using fname2]`

Create the file `fname1` from the model file `fname2`. If no model file name is given, the name `fname1.model` is assumed.

`Execute = cmd`

Execute the shell command `cmd`. Usually this would be the computation part, or mesh generation if using parametrized mesh generation.

`Output Files = fname1 [Using fname2]`

Read parameters from the output file `fname1` using the model file `fname2`. If `fname2` isn't given, `fname1.model` is assumed.

`Save File = fname`

Save history data of all the computations in file `fname`. If this keyword is active all the input parameters and the function return value are saved to a line that is appended in the given file.

`$`<*expression*>

Use MATC to evaluate the MATC expression <*expression*>. See section 3.1.2 for more.

Below is a simple example of a ElmerParam command file.

```
#
# This comment will be ignored.
#
Comment = ElmerOptim routine
Echo = True
Input File = OptimTemp2.sif
Execute = ElmerSolver
Output File = cost.dat
Save File = evals.dat
```

3.1.2 MATC extension

The ElmerParam includes the MATC language written by Juha Ruokolainen that may be used, for example, to evaluate the values of the parameters. The MATC extensions are by default not evaluated and the variable `MATC` need to be set `True` to activate the library.

In MATC, the real parameters are stored in a vector called `R`, integer paremeters in the vector `I`, and the function result in the vector `O`. Thus, `R`*n* can be accessed through `R(`*n*`)`, and so on.

Below is an example command file that shows how the MATC library may be used.

```
Comment = Testing routine
MATC = True
$ apu1 = R(0) + R(1)
$ apu2 = sin(apu1)
$ R(1) = apu2
$ I(0) = 5
$ I(1) = sum(I(2:4))
$ O = R(0) + R(1) + I(0)
```

3.2 Model files

3.2.1 Creating Input files

When creating an input file, ElmerParam simple copies the contents of the model file, expanding parameter references where they occur. A parameter reference in the model file has the form

```
<!ParamSpec[^]!>
```

where `ParamSpec` can be:

T	The tag parameter
X*n*	Parameter of type 'X' and index *n*.
X(*n*:*m*)	A vector of parameters from X*n* to X*m*.
X	A vector of all parameters of type 'X'

Here 'X' can be 'R' for real, or 'I' for integer. Note that indexing starts at 0; thus the first real parameter will be R0, the second R1, and so on. An optional transpose operator `^` after the `ParamSpec` denotes that it is a column vector (vectors are by default row vectors.). If a line contains a column vector, it will be repeated *n* times, where *n* is the lentgh of the vector, with the *i*:th vector component at the i:th copy of the line. If a line contains more than one columns, they have better be of the same length, or the vicious Yeti will eat your computer.

Example: if the model file looks like

```
r = <!R0!>
i = <!I!>
  Temperature = Variable Coordinate 2
    Real
        <!R(1:19)^!>     <!R(20:38)^!>
    End
```

and I = [1, 2, 3], R0 = 1.0, R(1:19) = [0.0, 0.2, 0.4 ...], and R(20:38) = [1.0, 2.0, 3.0, ...] the input file becomes

```
r = 1.0
i = 1  2  3
  Temperature = Variable Coordinate 2
    Real
        0.0     1.0
        0.2     2.0
        0.4     3.0
         .       .
         .       .
         .       .
    End
```

3.2.2 Reading Output files

When reading parameter values from an output file using a model file, ElmerParam will skip everything in the output file except those places where there is a parameter reference in the model file, in which case ElmerParam reads values from these places in the output file into the specified parameters.

The parameter references for output files has the same general form as for input files, where `ParamSpec` in this case can be

Xn	The n:th parameter of type 'X'.
X(n:m)	A vector of parameters from Xn to Xm.
X	A vector of all parameters of type 'X'

where 'X' can be 'R' for real, 'I' for integer, or 'O' (uppercase 'o') for function result.

For example, if the model file looks like

```
r = <!R!>
i = <!I0!>
y = <!O(0:4)^!> <!O(5:9)^!>
```

and the output file is

```
r = 1.0 2.0 3.0
i = 6
y = 1.0     6.0
    2.0     7.0
    3.0     8.0
    4.0     9.0
    5.0    10.0
```

ElmerParam will read [1.0, 2.0, 3.0] into R(0:2) (assuming the number of real parameters is 3), 6 into I0, and [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0] into O(0:9).

4 Interfaces

4.1 Standalone program

`ElmerParam [inputfile [outputfile [tag]]]`

`inputfile`: Name of file containing input parameters (use '-' or leave empty for stdin). It shall have the following format:

```
[nr
R0
R1
.
.
.
[ni
I0
I1
.
.
.
[nfun]]]
```

If `nfun` is absent, 1 will be assumed. Example: Three real parameters, no integer parameters, and 3 output parameters:

```
    3
    1.0
    2.0
    3.0
    0
    3
```

`outputfile`: Name of file where output parameters will be written in a column (use '-' or leave empty for stdout).

`tag`: Optional tag parameter.

4.2 C interface

```
#include <elmerparam.h>

double elmer_param(int nr, const double *xr, int ni, const int *xi,
                   const char *tag)

void elmer_param_vec(int nfun, double *fun,
                     int nr, const double *xr, int ni, const int *xi,
                     const char *tag)
```

`nfun`	Number of output parameters. (`elmer_param_vec` only)
`fun`	Array of output parameters, corresponds to O(0:`nfun`-1) in the ElmerParam input files. Must be big enough to hold `nfun` values. (`elmer_param_vec` only.)
`nr`	Number of real parameters.
`xr`	Array of real parameters, corresponds to R(0:`nr`-1) in the ElmerParam input files. (Use `NULL` for no real parameters.)
`ni`	Number of integer valued parameters.
`xi`	Array of integer parameters, corresponds to I(0:`ni`-1) in the ElmerParam input files. (Use `NULL` for no integer parameters.)
`tag`	Tag parameter. (Use `NULL` for no tag.)
Return value	Scalar output parameter, corresponds to O (or O0) in the ElmerParam input files. (`elmer_param` only.)

To compile a C code that calls `elmer_param()`, add

```
-I${ELMER_HOME}/include
```

to the compiler flags. For linking, add

```
-L${ELMER_HOME}/lib -lelmerparam -lmatc -lm
```

to the linker flags. (`-lmatc` can be omitted if ElmerParam was compiled without MATC support).

4.3 Fortran interface

```
use elmerparam

interface elmer_param

    function elmer_param_scal (xr, xi, tag) result(y)
        double precision, optional, intent(in) :: xr(:)
        integer, optional, intent(in) :: xi(:)
        character(*), optional, intent(in) :: tag

        double precision :: y
    end function elmer_param_scal

    function elmer_param_vec (nfun, xr, xi, tag) result(y)
        integer, intent(in) :: nfun
        double precision, optional, intent(in) :: xr(:)
        integer, optional, intent(in) :: xi(:)
        character(*), optional, intent(in) :: tag

        double precision :: y(nfun)
    end function elmer_param_vec

end interface elmer_param
```

`nfun`	Length of return vector (`elmer_param_vec` only).
`xr`	Array of real parameters, corresponds to R(0:`size(xr)`-1) in the ElmerParam input files.
`xi`	Array of integer parameters, corresponds to I(0:`size(xi)`-1) in the ElmerParam input files.
`tag`	Tag parameter.
Return value	`elmer_param_scal`: Scalar output parameter, corresponds to O (or O0) in the ElmerParam input files. `elmer_param_vec`: `nfun` Output parameters, corresponds to O(1:`nfun`) in the ElmerParam input files.

To compile a Fortran code that USEs `elmerparam`, add

```
-I${ELMER_HOME}/include
```

to the compiler flags. You'll have to replace "`-I`" with whatever option your compiler uses to tell where to look for `*.mod` files. This varies from one compiler to another – check the documentation of your compiler[1]. Also note that you have to use the same compiler that was used to compile ElmerParam. For linking, add

```
-L${ELMER_HOME}/lib -lelmerparamf -lelmerparam -lmatc -lm
```

to the linker flags. (`-lmatc` can be omitted if ElmerParam was compiled without MATC support).

4.4 R interface

```
library("elmerparam")

elmer_param <- function(xr = NULL, xt = NULL, xi = NULL, tag = "", nfun = 1)
```

`xr`	Array of real parameters, corresponds to R(0:`length(xr)`-1) in the ElmerParam input files.
`xi`	Array of integer parameters, corresponds to I(0:`length(xi)`-1) in the ElmerParam input files.
`tag`	Tag parameter.
`nfun`	Length of return vector.
Return value	Vector of output parameters, corresponds to O(0:`nfun`-1) in the ElmerParam input files.

[1]Most compilers uses `-I`; among the exceptions are Sun f95 which uses -M.

Note: Unless the package was installed in the default R library tree, you have to tell R where to find it. This can be done two ways; either using the `lib.loc` argument to `library()`:

```
library("elmerparam", lib.loc="/where/to/find/it/lib/R")
```

or by setting the environment variable "R_LIBS" to "`/where/to/find/it/lib/R`", where "`/where/to/find/it`" in both cases typically would be `$ELMER_HOME`.

4.5 Matlab interface

```
path(path, '/where/to/find/it/lib')

elmer_param(xr, xt, xi, tag, nfun)
```

Arguments are the same as for the R interface. Arguments `xr, xi` and `tag` can be a zero length vectors (`[ ]`), and arguments at the end of the argument list can be omitted. If `nfun` is omitted, a value of 1 is assumed.

Again, "`/where/to/find/it`", would typically be `$ELMER_HOME`.

5 Example: the Rosenbrock function

(Note: More examples can be found in the `examples/` directory in the source distribution of ElmerParam.)

Let's assume we have a computation program called "rosenbrock" that reads values for x_1 and x_2 from stdin, calculates the Rosenbrock function

$$z = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

and writes the result to stdout. The implementation of "rosenbrock" is not important; it can be written in any language, as long as it can be run from the command line as a stand alone program. For an implementation in awk, see code listing 1.

To run rosenbrock via ElmerParam we need to create these files:

1. costfunction.epc, an ElmerParam model file with the following contents:

```
Input file = xvalues
Execute = rosenbrock < xvalues > zval
Output file = zval
```

Listing 1 An awk implementation of "rosenbrock"

```
#!/bin/awk -f

BEGIN{
        getline;
        x = $1
        y = $2
        print (1-x)^2 + 100*(y - x^2)^2
}
```

2. ELMERPARAM_STARTINFO, to tell the name of the command file. The contents of ELMERPARAM_STARTINFO is thus simply

   ```
   costfunction.epc
   ```

3. xvalues.model, a model file used to instruct ElmerParam to write <!R0!> and <!R1!> to "xvalues":

   ```
   <!R0!>  <!R1!>
   ```

4. zval.model, a model file to tell ElmerParam how to extract the function value from the file "zval":

   ```
   <!0!>
   ```

5.1 Optimize using R

To find the minimum of the rosenbrock function in R using the function "optim", start up R and type the commands:

```
> library("elmerparam")
> optim(c(0,0), elmer_param)
...
$par
[1] 0.9999564 0.9999085

$value
```

```
[1] 3.72849e-09

$counts
function gradient
     169       NA

$convergence
[1] 0

$message
NULL
```

The yielded result $x_{\min} = (0.9999564,\ 0.9999085)$ with $z_{\min} = 3.72849e - 09$ is very close to the expected result $x_{\min} = (1.0, 1.0)$ with $z_{\min} = 0$.

5.2 Optimize using APPSPACK; standalone program

APPSPACK[2] is an optimization package designed to be used to optimize functions defined using external computation programs, and is thus very well suited to be use with Elmer via ElmerParam. We have no ambitions to provide a complete APPSPACK manual here, only a simple example is given. Please see the APPSPACK web page for more information on APPSPACK.

APPSPACK comes in two flavors; a serial version, and a MPI parallel version. Their usage is very similar, and we'll cover both.

APPSPACK needs a standalone program to evaluate the cost function; for this we can use `ElmerParam` directly.

For the serial version of APPSPACK the tag parameter doesn't matter, but for the MPI version it's important. To use it, we have to make some small modifications to the ElmerParam command file costfunction.epc:

```
Input file = xvalues.<!T!> Using xvalues.model
Execute = rosenbrock < xvalues.<!T!> > zval.<!T!>
Output file = zval.<!T!> Using zval.model

# Clean up afterwards:
Execute = rm xvalues.<!T!> zval.<!T!>
```

No other ElmerParam files needs changes. Finally, an input file for APPSPACK is needed. For this case, we use a file named "appspack_input.apps":

[2] `http://software.sandia.gov/appspack/`

```
# SAMPLE APPSPACK INPUT FILE
@ "Linear"
"Upper" vector 2  100 100
"Lower" vector 2 -100 -100
"Scaling" vector 2 1 1
@@
@ "Evaluator"
"Executable Name" string "ElmerParam"
@@
@ "Solver"
"Debug" int 3
"Initial X" vector 2  0 0
"Step Tolerance" double 1.0e-5
@@
```

With this, we can run APPSPACK with the command

```
> mpirun -n 2 /usr/local/bin/appspack_mpi appspack_input.apps
```

for the MPI version, or

```
> appspack_serial appspack_input.apps
```

for the serial version, yielding the result

```
Final State: Step Converged

Final Min: f=5.290e-10 x=[ 1.000e-00  1.000e-00 ] \
    step=1.000e-05 tag=27176 state=Evaluated  Success: 11251
```

Again, the result is correct (albeit at 11257 function evaluations, it's not very efficient. Increasing the "Step Tolerance" in appspack_input.apps reduces the numer of function evaluations, but also the accuracy.)