

Parallel Programming Laboratory  
University of Illinois at Urbana-Champaign

---

CHARM++  
NetFEM  
Manual

The initial version of CHARM++ NetFEM Framework was developed by Orion Lawlor in 2001.

---

Version 1.0

University of Illinois  
CHARM++/CONVERSE Parallel Programming System Software  
Non-Exclusive, Non-Commercial Use License

Upon execution of this Agreement by the party identified below ("Licensee"), The Board of Trustees of the University of Illinois ("Illinois"), on behalf of The Parallel Programming Laboratory ("PPL") in the Department of Computer Science, will provide the CHARM++/CONVERSE Parallel Programming System software ("CHARM++") in Binary Code and/or Source Code form ("Software") to Licensee, subject to the following terms and conditions. For purposes of this Agreement, Binary Code is the compiled code, which is ready to run on Licensee's computer. Source code consists of a set of files which contain the actual program commands that are compiled to form the Binary Code.

1. The Software is intellectual property owned by Illinois, and all right, title and interest, including copyright, remain with Illinois. Illinois grants, and Licensee hereby accepts, a restricted, non-exclusive, non-transferable license to use the Software for academic, research and internal business purposes only, e.g. not for commercial use (see Clause 7 below), without a fee.
2. Licensee may, at its own expense, create and freely distribute complimentary works that interoperate with the Software, directing others to the PPL server (<http://charm.cs.uiuc.edu>) to license and obtain the Software itself. Licensee may, at its own expense, modify the Software to make derivative works. Except as explicitly provided below, this License shall apply to any derivative work as it does to the original Software distributed by Illinois. Any derivative work should be clearly marked and renamed to notify users that it is a modified version and not the original Software distributed by Illinois. Licensee agrees to reproduce the copyright notice and other proprietary markings on any derivative work and to include in the documentation of such work the acknowledgement:

"This software includes code developed by the Parallel Programming Laboratory in the Department of Computer Science at the University of Illinois at Urbana-Champaign."

Licensee may redistribute without restriction works with up to 1/2 of their non-comment source code derived from at most 1/10 of the non-comment source code developed by Illinois and contained in the Software, provided that the above directions for notice and acknowledgement are observed. Any other distribution of the Software or any derivative work requires a separate license with Illinois. Licensee may contact Illinois ([kale@cs.uiuc.edu](mailto:kale@cs.uiuc.edu)) to negotiate an appropriate license for such distribution.

3. Except as expressly set forth in this Agreement, THIS SOFTWARE IS PROVIDED "AS IS" AND ILLINOIS MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY PATENT, TRADEMARK, OR OTHER RIGHTS. LICENSEE ASSUMES THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS. LICENSEE AGREES THAT UNIVERSITY SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES WITH RESPECT TO ANY CLAIM BY LICENSEE OR ANY THIRD PARTY ON ACCOUNT OF OR ARISING FROM THIS AGREEMENT OR USE OF THE SOFTWARE AND/OR ASSOCIATED MATERIALS.
4. Licensee understands the Software is proprietary to Illinois. Licensee agrees to take all reasonable steps to insure that the Software is protected and secured from unauthorized disclosure, use, or release and will treat it with at least the same level of care as Licensee would use to protect and secure its own proprietary computer programs and/or information, but using no less than a reasonable standard of care. Licensee agrees to provide the Software only to any other person or entity who has registered with Illinois. If licensee is not registering as an individual but as an institution or corporation each member of the institution or corporation who has access to or uses Software must agree to and abide by the terms of this license. If Licensee becomes aware of any unauthorized licensing, copying or use of the Software, Licensee shall promptly notify Illinois in writing. Licensee expressly agrees to use the Software only in the manner and for the specific uses authorized in this Agreement.
5. By using or copying this Software, Licensee agrees to abide by the copyright law and all other applicable laws of the U.S. including, but not limited to, export control laws and the terms of this license. Illinois shall have the right to terminate this license immediately by written notice upon Licensee's breach of, or non-compliance with, any terms of the license. Licensee may be held legally responsible for any copyright infringement that is caused or encouraged by its failure to abide by the terms of this license. Upon termination, Licensee agrees to destroy all copies of the Software in its possession and to verify such destruction in writing.
6. The user agrees that any reports or published results obtained with the Software will acknowledge its use by the appropriate citation as follows:

"CHARM++/CONVERSE was developed by the Parallel Programming Laboratory in the Department of Computer Science at the University of Illinois at Urbana-Champaign."

Any published work which utilizes CHARM++ shall include the following reference:

"L. V. Kale and S. Krishnan. CHARM++: Parallel Programming with Message-Driven Objects. In 'Parallel Programming using C++' (Eds. Gregory V. Wilson and Paul Lu), pp 175-213, MIT Press, 1996."

Any published work which utilizes CONVERSE shall include the following reference:

"L. V. Kale, Milind Bhandarkar, Narain Jagathesan, Sanjeev Krishnan and Joshua Yelon. CONVERSE: An Interoperable Framework for Parallel Programming. Proceedings of the 10th International Parallel Processing Symposium, pp 212-217, April 1996."

Electronic documents will include a direct link to the official CHARM++ page at <http://charm.cs.uiuc.edu/>

7. Commercial use of the Software, or derivative works based thereon, REQUIRES A COMMERCIAL LICENSE. Should Licensee wish to make commercial use of the Software, Licensee will contact Illinois ([kale@cs.uiuc.edu](mailto:kale@cs.uiuc.edu)) to negotiate an appropriate license for such use. Commercial use includes:
  - (a) integration of all or part of the Software into a product for sale, lease or license by or on behalf of Licensee to third parties, or
  - (b) distribution of the Software to third parties that need it to commercialize product sold or licensed by or on behalf of Licensee.
8. Government Rights. Because substantial governmental funds have been used in the development of CHARM++/CONVERSE, any possession, use or sublicense of the Software by or to the United States government shall be subject to such required restrictions.
9. CHARM++/CONVERSE is being distributed as a research and teaching tool and as such, PPL encourages contributions from users of the code that might, at Illinois' sole discretion, be used or incorporated to make the basic operating framework of the Software a more stable, flexible, and/or useful product. Licensees who contribute their code to become an internal portion of the Software agree that such code may be distributed by Illinois under the terms of this License and may be required to sign an "Agreement Regarding Contributory Code for CHARM++/CONVERSE Software" before Illinois can accept it (contact [kale@cs.uiuc.edu](mailto:kale@cs.uiuc.edu) for a copy).

UNDERSTOOD AND AGREED.

Contact Information:

The best contact path for licensing issues is by e-mail to [kale@cs.uiuc.edu](mailto:kale@cs.uiuc.edu) or send correspondence to:

Prof. L. V. Kale  
Dept. of Computer Science  
University of Illinois  
201 N. Goodwin Ave  
Urbana, Illinois 61801 USA  
FAX: (217) 333-3501

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Compiling and Installing</b>	<b>3</b>
<b>3</b>	<b>Running NetFEM Online</b>	<b>3</b>
<b>4</b>	<b>Running NetFEM Offline</b>	<b>3</b>
<b>5</b>	<b>NetFEM with other Visualization Tools</b>	<b>4</b>
<b>6</b>	<b>Interface Basics</b>	<b>4</b>
<b>7</b>	<b>Simple Interface</b>	<b>5</b>
<b>8</b>	<b>Advanced “Field” Interface</b>	<b>6</b>

## 1 Introduction

NetFEM was built to provide an easy way to visualize the current state of a finite-element simulation, or any parallel program that computes on an unstructured mesh. NetFEM is designed to require very little effort to add to a program, and connects to the running program over the network via the network protocol CCS (Converse Client/Server).

## 2 Compiling and Installing

NetFEM is part of CHARM++, so it can be downloaded as part of charm. To build NetFEM, just build FEM normally, or else do a make in `charm/net-linux/tmp/libs/ck-libs/netfem/`.

To link with NetFEM, add `-module netfem` to your program’s link line. Note that you do *not* need to use the FEM framework to use NetFEM.

The netfem header file for C is called “netfem.h”, the header for fortran is called ‘netfem.f.h’. A simple example NetFEM program is in `charm/pgms/charm++/fem/simple2D/`. A more complicated example is in `charm/pgms/charm++/fem/crack2D/`.

## 3 Running NetFEM Online

Once you have a NetFEM program, you can run it and view the results online by starting the program with CCS enabled:

```
foo.bar.edu> ./charmrun ./myprogram +p2 ++server ++server-port 1234
```

“++server-port” controls the TCP port number to use for CCS—here, we use 1234. Currently, NetFEM only works with one chunk per processor—that is, the `-vp` option cannot be used.

To view the results online, you then start the NetFEM client, which can be downloaded for Linux or Windows from

<http://charm.cs.uiuc.edu/research/fem/netfem/>

Enter the name of the machine running charmrun and the TCP port number into the NetFEM client—for example, you might run:

```
netfem foo.bar.edu:1234
```

The NetFEM client will then connect to the program, download the most recent mesh registered with `NetFEM.POINTAT`, and display it. At any time, you can press the “update” button to reload the latest mesh.

## 4 Running NetFEM Offline

Rather than using CCS as above, you can register your meshes using `NetFEM.WRITE`, which makes the server write out binary output dump files. For example, to view timestep 10, which is written to the “NetFEM/10/” directory, you’d run the client program as:

```
netfem NetFEM/10
```

In offline mode, the “update” button fetches the next extant timestep directory.

## 5 NetFEM with other Visualization Tools

You can use a provided converter program to convert the offline NetFEM files into an XML format compatible with the powerful offline visualization tool ParaView(<http://paraview.org>). The converter is located in `.../charm/src/libs/ck-libs/netfem/ParaviewConverter/`. Build the converter by simply issuing a “make” command in that directory(assuming NetFEM already has been built).

Run the converter from the parent directory of the “NetFEM” directory to be converted. The converter will generate a directory called “ParaViewData”, which contains subdirectories for each timestep, along with a “timestep” directory for index files for each timestep. All files in the ParaViewData directory can be opened by ParaView. To open all chunks for a given timestep, open the desired timestep file in “ParaView-Data/timesteps”. Also, individual partition files can also be opened from “ParaViewData / <timestep> / <partition\_num>”.

## 6 Interface Basics

You publish your data via NetFEM by making a series of calls to describe the current state of your data. There are only 6 possible calls you can make.

`NetFEM_Begin` is the first routine you call. `NetFEM_End` is the last routine to call. These two calls bracket all the other NetFEM calls.

`NetFEM_Nodes` describes the properties of the nodes, or vertices of the domain. `NetFEM_Elements` describes the properties of your elements (triangles, tetrahedra, etc.). After making one of these calls, you list the different data arrays associated with your nodes or elements by making calls to `NetFEM_Scalar` or `NetFEM_Vector`.

For example, a typical finite element simulation might have a scalar mass and vector position, velocity, and net force associated with each node; and have a scalar stress value associated with each element. The sequence of NetFEM calls this application would make would be:

```
NetFEM_Begin
NetFEM_Nodes -- lists position of each node
NetFEM_Vector -- lists velocity of each node
NetFEM_Vector -- lists net force on each node
NetFEM_Scalar -- lists mass of each node

NetFEM_Elements -- lists the nodes of each element
NetFEM_Scalar -- lists the stress of each element

NetFEM_End
```

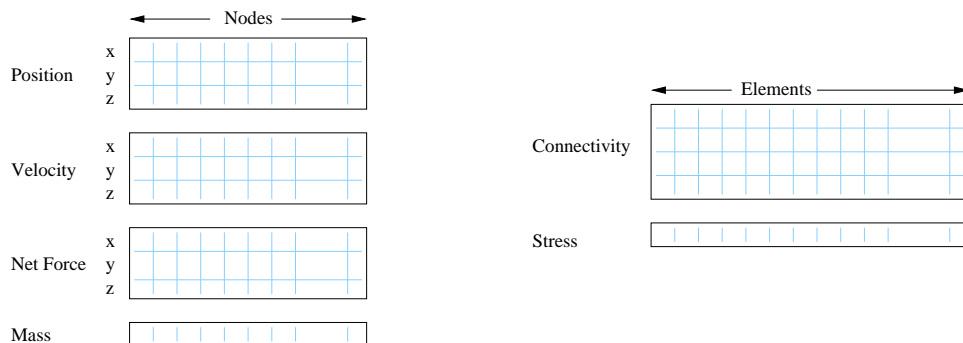


Figure 1: These arrays, typical of a finite element analysis program, might be passed into NetFEM.

## 7 Simple Interface

The details of how to make each call are:

```
NetFEM NetFEM_Begin(int source, int step, int dim, int flavor);
integer function NetFEM_Begin(source,step,dim,flavor)
    integer, intent(in) :: source,step,dim,flavor
```

Begins describing a single piece of a mesh. Returns a handle that is used for each subsequent call until `NetFEM_End`. This call, like all `NetFEM` calls, is collective—every processor should make the same calls in the same order.

*source* identifies the piece of the mesh—use `FEM_My_partition` or `CkMyPe`.

*step* identifies which version of the mesh this is—for example, you might use the timestep number. This is only used to identify the mesh in the client.

*dim* is the number of spatial dimensions. For example, in a 2D computation, you'd pass `dim==2`; in a 3D computation, `dim==3`. The client currently only supports 2D or 3D computations.

*flavor* specifies what to do with the data. This can take the value `NetFEM_POINTAT`, which is used in online visualization, and specifies that `NetFEM` should only keep a pointer to your data rather than copy it out of your arrays. Or it can take the value `NetFEM_WRITE`, which writes out the data to files named “`NetFEM/step/source.dat`” for offline visualization.

```
void NetFEM_End(NetFEM n);
subroutine NetFEM_End(n)
    integer, intent(in) :: n
```

Finishes describing a single piece of a mesh, which then makes the mesh available for display.

```
void NetFEM_Nodes(NetFEM n,int nNodes,const double *loc,const char *name);
subroutine NetFEM_Nodes(n,nNodes,loc,name)
    integer, intent(in) :: n, nNodes
    double precision, intent(in) :: loc(dim,nNodes)
    character*(*), intent(in) :: name
```

Describes the nodes in this piece of the mesh.

*n* is the `NetFEM` handle obtained from `NetFEM_Begin`.

*nNodes* is the number of nodes listed here.

*loc* is the location of each node. This must be double-precision array, laid out with the same number of dimensions as passed to `NetFEM_Begin`. For example, in C the location of a 2D node *n* is stored in `loc[2*n+0]` (x coordinate) and `loc[2*n+1]` (y coordinate). In Fortran, location of a node *n* is stored in `loc(:,n)`.

*name* is a human-readable name for the node locations to display in the client. We recommend also including the location units here, for example “Position (m)”.

```
void NetFEM_Elements(NetFEM n,int nElements,int nodePerEl,const int *conn,const char *name);
subroutine NetFEM_Elements(n,nElements,nodePerEl,conn,name)
    integer, intent(in) :: n, nElements, nodePerEl
    integer, intent(in) :: conn(nodePerEl,nElements)
    character*(*), intent(in) :: name
```

Describes the elements in this piece of the mesh. Unlike `NetFEM_Nodes`, this call can be repeated if there are different types of elements (For example, some meshes contain a mix of triangles and quadrilaterals).

*n* is the `NetFEM` handle obtained from `NetFEM_Begin`.

*nElements* is the number of elements listed here.

*nodePerEl* is the number of nodes for each element. For example, a triangle has 3 nodes per element; while tetrahedra have 4.

*conn* gives the index of each element's nodes. Note that when called from C, the first node is listed in *conn* as 0 (0-based node indexing), and element *e*'s first node is stored in *conn*[*e*\**nodePerEl*+0]. When called from Fortran, the first node is listed as 1 (1-based node indexing), and element *e*'s first node is stored in *conn*(1,*e*) or *conn*((*e*-1)\**nodePerEl*+1).

*name* is a human-readable name for the elements to display in the client. For example, this might be "Linear-Strain Triangles".

```
void NetFEM_Vector(NetFEM n,const double *data,const char *name);
subroutine NetFEM_Vector(n,data,name)
    integer, intent(in) :: n
    double precision, intent(in) :: data(dim,lastEntity)
    character*(*), intent(in) :: name
```

Describes a spatial vector associated with each node or element in the mesh. Attaches the vector to the most recently listed node or element. You can repeat this call several times to describe different vectors.

*n* is the NetFEM handle obtained from *NetFEM\_Begin*.

*data* is the double-precision array of vector values. The dimensions of the array have to match up with the node or element the data is associated with—in C, a 2D element *e*'s vector starts at *data*[2\**e*]; in Fortran, element *e*'s vector is *data*(:,*e*).

*name* is a human-readable name for this vector data. For example, this might be "Velocity (m/s)".

```
void NetFEM_Scalar(NetFEM n,const double *data,int dataPer,const char *name);
subroutine NetFEM_Scalar(n,data,dataPer,name)
    integer, intent(in) :: n, dataPer
    double precision, intent(in) :: data(dataPer,lastEntity)
    character*(*), intent(in) :: name
```

Describes some scalar data associated with each node or element in the mesh. Like *NetFEM\_Vector*, this data is attached to the most recently listed node or element and this call can be repeated. For a node or element, you can make the calls to *NetFEM\_Vector* and *NetFEM\_Scalar* in any order.

*n* is the NetFEM handle obtained from *NetFEM\_Begin*.

*data* is the double-precision array of values. In C, an element *e*'s scalar values start at *data*[*dataPer*\**e*]; in Fortran, element *e*'s values are in *data*(:,*e*).

*dataPer* is the number of values associated with each node or element. For true scalar data, this is 1; but can be any value. Even if *dataPer* happens to equal the number of dimensions, the client knows that this data does not represent a spatial vector.

*name* is a human-readable name for this scalar data. For example, this might be "Mass (Kg)" or "Stresses (pure)".

## 8 Advanced "Field" Interface

This more advanced interface can be used if you store your node or element data in arrays of C structs or Fortran *TYPE*s. To use this interface, you'll have to provide the name of your struct and field. Each "field" routine is just an extended version of a regular *NetFEM* call described above, and can be used in place of the regular *NetFEM* call. In each case, you pass a description of your field in addition to the usual *NetFEM* parameters.

In C, use the macro "*NetFEM\_Field*(*theStruct*,*theField*)" to describe the *FIELD*. For example, to describe the field "loc" of your structure named "node\_t",

```
node_t *myNodes=...;
..., NetFEM_Field(node_t,loc), ...
```

In Fortran, you must pass as FIELD the byte offset from the start of the structure to the start of the field, then the size of the structure. The FEM "offsetof" routine, which returns the number of bytes between its arguments, can be used for this. For example, to describe the field "loc" of your named type "NODE",

```
TYPE(NODE), ALLOCATABLE :: n(:)
..., foffsetof(n(1),n(1)%loc),foffsetof(n(1),n(2)), ...
```

```
void NetFEM_Nodes_field(NetFEM n,int nNodes,FIELD,const void *loc,const char *name);
subroutine NetFEM_Nodes_field(n,nNodes,FIELD,loc,name)
```

A FIELD version of NetFEM\_Nodes.

```
void NetFEM_Elements_field(NetFEM n,int nElements,int nodePerEl,FIELD,int idxBase,const int *conn,const char
*name);
subroutine NetFEM_Elements_field(n,nElements,nodePerEl,FIELD,idxBase,conn,name)
```

A FIELD version of NetFEM\_Elements. This version also allows you to control the starting node index of the connectivity array—in C, this is normally 0; in Fortran, this is normally 1.

```
void NetFEM_Vector_field(NetFEM n,const double *data,FIELD,const char *name);
subroutine NetFEM_Vector_field(n,data,FIELD,name)
```

A FIELD version of NetFEM\_Vector.

```
void NetFEM_Scalar_field(NetFEM n,const double *data,int dataPer,FIELD,const char *name);
subroutine NetFEM_Scalar(n,data,dataPer,FIELD,name)
```

A FIELD version of NetFEM\_Scalar.