Open Framework for Experimental Setup and Control (OpenFresco)

OpenFresco Command Language Manual

Andreas Schellenberg, Hong K. Kim, Yoshikazu Takahashi,
Gregory L. Fenves, and Stephen A. Mahin

OpenFresco.exe & OpenFresco.dll
Version 2.6

July 2009

Created on 9/26/07

Contents

CHAPTER 1

Introduction

In This Chapter

Introduction to OpenFresco

Hybrid simulation provides a versatile, realistic and cost-effective method for simulating the seismic response of structural systems experimentally. A hybrid simulation is a combination of physical simulation of a specimen in a laboratory, using standard servo-controlled actuators, with computational simulation of the system to simulate the overall response of the system to earthquake ground motion.

To address this important need in earthquake engineering, The Open Framework for Experimental Setup and Control (OpenFresco) is a software system for hybrid simulation of structural systems. For the earthquake engineering user, OpenFresco is a practical and easy to use software package that allows a wide variety of hybrid simulation algorithms, laboratory and control systems, experimental setups, and computational simulation models to be combined for a specific hybrid simulation. For the researcher or developer interested in new hybrid simulation methods, the architecture of OpenFresco provides a great deal of flexibility, extensibility, and re-usability through an object-oriented software framework.

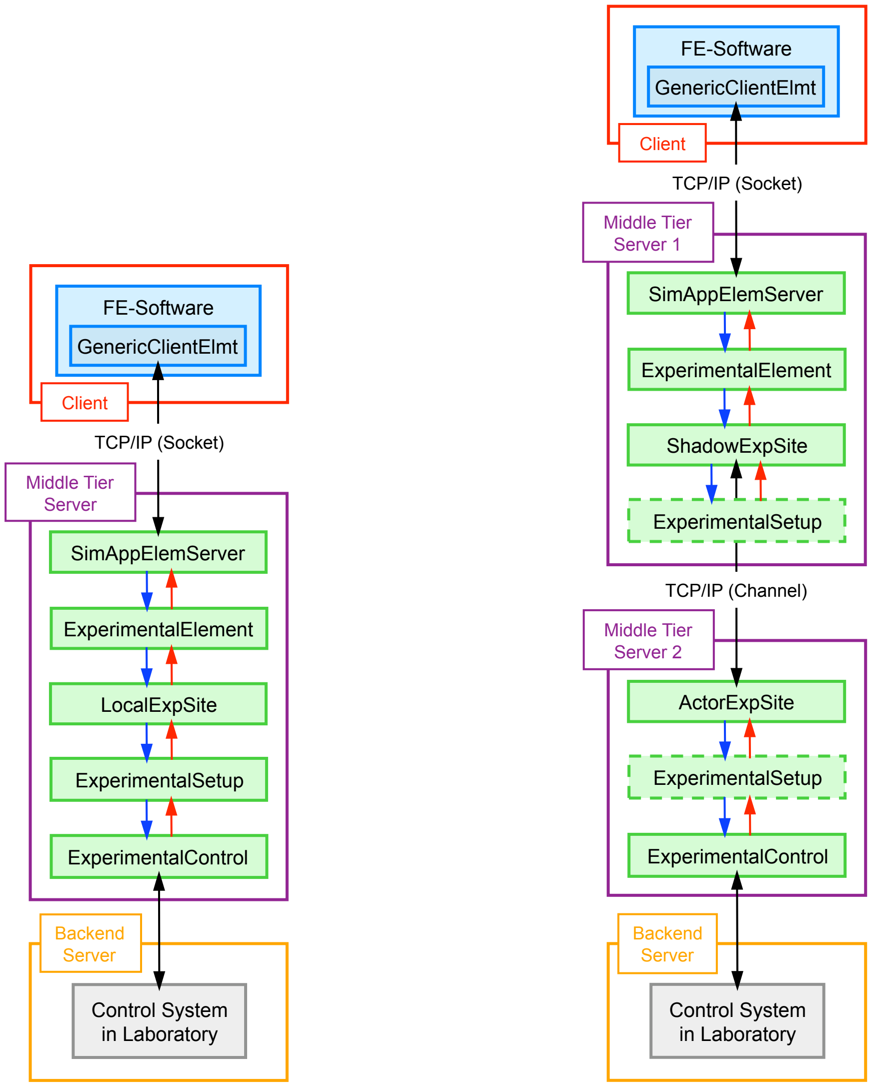


Figure 1: OpenFresco Software Architecture for Local (Left) and Distributed Simulation (Right).

The design, implementation, and proof-of-concept of OpenFresco was described in Takahashi and Fenves (2006). Schellenberg and Mahin (2006a) used OpenFresco with the computational simulation provided by OpenSees. Schellenberg and Mahin (2006b) applied OpenFresco to simulate structural collapse under earthquake ground motion using hybrid simulation. Schellenberg et al. (2006c) provided a description of the progress using OpenFresco for a range of hybrid simulation applications.

The original version of OpenFresco was tightly bound to the Open System for Earthquake Engineering Simulation (OpenSees, 2006) for the computational simulation. To provide users a broader range of choices of software packages for computational modeling and simulation, a generalization of OpenFresco has been completed and released as Version 2.6. The refactored OpenFresco is similar to the original design in Takahashi and Fenves (2006), but it is now based on a n-tier client-server architecture. The tiers are a client, one or two middle tier servers, and a backend server (Figure 1)[1]. In Figure 1, the blue box represents the finite element software for the computational simulation. The green boxes represent OpenFresco processes, and the gray box represents the laboratory control systems. The red box shows the processes that occur on the client side, the purple box shows the processes that occur on the middle tier server side, and the orange box shows that which is occurring on the server side.

As with Version 2.5 of OpenFresco, the Experimental Setup can be located on either the middle tier server side or the server side, which provides users a great deal of flexibility in defining the experimental configuration and control of the specimen in a laboratory.

An important aspect of OpenFresco is that the finite element software in Figure 1 uses a general form of an element, termed the Generic-Client Element. The user does not have to create an experimental element in the finite element software when the Generic-Client Element is used. The Generic-Client Element can be easily implemented into any finite element software that allows user-defined elements. This feature allows OpenFresco to be used with a wide variety of computational software packages, such as OpenSees, Matlab, LS-DYNA, UI-SimCor, Abaqus and Simulink. For developers and advanced users, experimental elements can be created for more sophisticated applications, but for most users interested in hybrid simulation, the Generic-Client Element should be sufficient.

Detailed information about installing OpenFresco can be found in the “OpenFresco Installation and Getting Started Guide”. This document also contains a simple hybrid simulation example to get the user started. The OpenFresco example manuals contain more comprehensive examples for advanced users. Both guides are available at

http://neesforge.nees.org/docman/index.php?group_id=36&selected_doc_group_id=36&language_id=1

Throughout this document, proprietary products for software and hardware are referred to without endorsement.

[1] For further information on the client-server architecture, see http://www.sei.cmu.edu/str/descriptions/clientserver.html.

Change Notes for Version 2.6

Modeling of Experimental Elements:

The OpenFresco experimental elements act within the finite element (FE) software to represent the portion of the structure that is physically tested in an experiment. Moreover, the experimental elements provide the necessary interface to the analysis procedures in the FE analysis software.

In addition to the experimental elements in Version 2.5, Version 2.6 has the following features:

1. The twoNodeLink experimental element has been modified to incorporate coupled shear and moment behavior if the length of the element is larger than zero. In addition, if the element does not have zero length, the user can now optionally specify how the P-Delta moments around the local x- and y-axis are distributed among a moment at node i, a moment at node j and a shear couple. The sum of these three ratios is always equal to 1.
2. A corotTruss experimental element is now available to test truss elements considering large displacements. A corotational formulation adopts a set of corotational axes, which rotate with the element, thus taking into account an exact geometric transformation between local and global frames of reference.

Representing the Setup of Experiments:

The experimental setup objects in OpenFresco are software abstractions of an experiment's actual setup in the laboratory. As a result, they are responsible for transforming the response quantities between the experimental element degrees of freedom and the actuator degrees of freedom, utilizing the geometry and the kinematics of the loading and instrumentation system, and back again.

Several improvements and additions are introduced in Version 2.6:

1. A FourActuators3d experimental setup has been added to test columns in 3D. As the name suggests, this setup utilized four actuators to control the two translational and the two rotational degrees of freedom of a frame specimen in 3D. The axial and the torsional degrees of freedom are not controlled. However, the setup accounts for the effect of a constant axial load (which is applied to the specimen with a spreader beam and two prestress rods) on the element resisting forces. All transformations in this experimental setup are implemented as non-linear transformations accounting for large displacements.
2. The response modification factors in the experimental setups can now be applied to the trial response quantities before they are transformed to control signals and to the output response quantities after they have been converted from the daq signals. This flexibility to apply factors to trial and output response quantities in addition to control and daq response quantities was necessary to correctly account for similitude scaling operations.

Interfacing with Controllers and Data Acquisition Systems:

OpenFresco provides a wide range of functions for communicating with the laboratory control and data acquisition systems. The functions provide command actions in the actuator coordinate system and they obtain measured data from the data acquisition system.

In Version 2.6, significantly new capability is provided for:

1. The deadlock problems that were encountered with xPC-Target versions newer than 3.1 have been resolved. The xPCtarget experimental control is working flawlessly with the current xPC-Target version 4.1 (Matlab R2009a).
2. The experimental control objects are now able to use signal filter objects to filter control and daq response quantities and to simulate experimental errors such as overshoot, undershoot, lead, lag or random noise.

Simulating Specimens (including Controllers and Data Acquisition Systems):

In order to check a hybrid model and estimate the response of a structure before running an actual test, specimens can be simulated using the OpenFresco simulation experimental control and data acquisition objects. Besides the existing SimUniaxialMaterial and SimDomain experimental control classes the SimFEAdapter class has been added in Version 2.6. The SimFEAdpater object enables users to simulate physical subassemblies and specimens in any finite element software of their choice. Adapter elements that act as the interfaces to the slave finite element software, in which the subassemblies are simulated, have so far been implemented for OpenSees, LS-DYNA and Abaqus. Because the adapter element approach is not utilizing any file system to exchange data between the master and the slave FE-software and such programs are not required to repetitively be shutdown and restarted, hybrid simulations can be performed continuously and concurrently.

Utilizing different Computational Drivers:

To interface with a variety of finite element software packages, OpenFresco uses the three-tier software architecture concept and a generic client element in the FE-software. A generic client element can be implemented into any FE-software, which provides an application programming interface (API) that allows for the addition of user-defined elements. In addition, a generic client element only needs to be implemented once for any FE-software that is desired to be used as a computational driver for performing hybrid simulations.

Generic client elements have been implemented for OpenSees, Matlab, LS-DYNA, UI-SimCor, Abaqus and Simulink. The Abaqus and the Simulink interfaces are new in OpenFresco Version 2.6.

Recording Experimental Data:

The recorder classes provide means for recording response quantities from all the experimental objects (such as elements, sites, setups and controls) that are of interest to the experimentalist. Data can be recorded in many different formats including, ASCII format without headers, ASCII comma separated format without headers, ASCII format including xml metadata and binary format. In addition, response data can be recorded to databases such as SQL or Berkeley DB.

Installing the OpenFresco Software:

Because of the quickly expanding library of interfaces with controllers and data acquisition systems supported by OpenFresco, the number of required dynamic link libraries is growing rapidly as well. In contrast, the installation of OpenFresco for users and/or developers should be as straightforward and flexible as possible.

In Version 2.6, the OpenFresco installer for Windows platforms, which was introduced in Version 2.5 using the NSIS (Nullsoft Scriptable Install System) professional open-source software, is employed again. The installer provides installation options for general users as well as developers of OpenFresco.

Documentation for OpenFresco:

Several major new documents are provided in public review form for support and explanations for different aspects of OpenFresco. These will be finalized following a brief public review period. The new documentation accompanying Version 2.6 describes:

1. Installation of the software and getting started,
2. OpenFresco specific TCL scripting language commands,
3. Hybrid simulation examples,
4. Adaptation of FE-software packages OpenSees, Abaqus, LS-Dyna and Matlab for hybrid testing, and
5. Generation of certificates needed for the secure communication channels.

Notation

The notation presented in this chapter is used throughout this document.

Input values are a string, unless the first character is a **$**, in which case an integer, floating point number or variable is to be provided. In the Tcl language, variable references start with the **$** character. Tcl expressions can also be used as input to the commands where the input value is specified by the first character being a **$**.

Optional values are identified in enclosing **<>** braces.

When specifying a variable quantity of values, the command line contains **(x $values)**. The number of values required, x, and the types of values, $values, are specified in the description of the command.

An arbitrary number of input values is indicated with the dot-dot-dot notation, i.e. **$value1 $value2 ...**

The OpenFresco interpreter constructs objects in the order they are specified by the user. New objects are often based on previously defined objects. When specified as an object parameter, a previously defined object must have already been added to the Domain. This requirement is specified in the description of the command arguments.

Copyright

Acknowledgement: This software is based on work supported by the National Science Foundation (NSF) under Award Number 019796 to the University of California, Berkeley, from the San Diego Supercomputer Center (SDSC) for Operations and Maintenance of NEES Simulation Component. It is a component of NSF Award Number 0402490, NEES Consortium Operations: FY2005-FY2014, through the NEES Information Technology (NEESit) Cyberinfrastructure subaward to SDSC. In addition, this work was also supported† by the Earthquake Engineering Research Centers Program of the National Science Foundation, under award number EEC-9701568 through the Pacific Earthquake Engineering Research Center (PEER).

Introduction to the Tcl command language

The Tcl scripting language is used to implement the OpenFresco commands. A user issues commands to create the objects needed for a hybrid simulation, such as experimental elements, experimental sites, experimental setups and experimental controls, all of which are described in this manual. The Tcl language provides useful programming tools, such as variables, mathematical expression evaluation, and control structures, which allow a user to create scripts that define their hybrid simulation.

Tcl is a string-based scripting language which allows the following:

- Variables and variable substitution
- Mathematical expression evaluation
- Basic control structures (if, while, for, foreach)
- Procedures
- File manipulation

More information on Tcl commands can be found at its web site: *Tcl/Tk Primer*
http://www.tcl.tk/man/tcl8.5/TclCmd/contents.htm (http://www.tcl.tk/man/tcl8.5/TclCmd/contents.htm)

Tcl Commands Format

Tcl scripts are made up of commands separated by new lines or semicolon (;).

The basic syntax for a Tcl command is:

```
command $arg1 $arg2  ...
```

command	name of the Tcl command or user-defined procedure
$arg1 $arg2 ...	arguments for the command

Tcl allows any argument to be a nested command:

```
command [nested-command1] [nested-command2]
```

where the [] are used to delimit the nested commands. The Tcl interpreter will first evaluate the nested commands and will then evaluate the outer command with the results to the nested commands.

The most basic command in Tcl is the set command:

```
set variable $value
```

EXAMPLE:

```
set a 5
```

The Tcl interpreter regards a command starting with the number sign (#) to be a comment statement, so it does not execute anything following the #. For example:

```
# this command assigns the value 5 to the variable a
set a 5
```

The number sign and the semicolon can be used together to put comments on the same line as the command. For example:

```
set a 5;  # this command assigns the value 5 to the variable a
```

Example Tcl Commands

arithmetic

```
>set a 1
1
>set b a
a
>set b $a
1
>expr 2 + 3
5
>expr 2 + $a
3
>set b [expr 2 + $a]
3
>
```

procedure

```
>proc sum {a b} {
    return [expr $a + $b]
}
>sum 2 3
5
>set c [sum 2 3]
5
>
```

for & foreach functions

```
for {set i 1} {$i < 10} {incr i 1} {
      puts “i equals $i”
}


set sum 0
foreach value {1 2 3 4} {
      set sum [expr $sum +
$value]
}
puts $sum
10
>
```

file manipulation

```
>set fileId [open tmp w]
anumber
>puts $fileId “hello”
>close $fileID
>type tmp
hello
>


>source Example1.tcl
```

procedure & if statement

```
>proc guess {value} {
     global sum
     if {$value < $sum} {
         puts “too low”
     } else {
         if {$value > $sum} {
             puts “too high”
         } else { puts “you got it!”}
     }
}
 > guess 9
too low
>
```

Additional Tcl Resources

Below, are additional resources for Tcl:

http://www.freeprogrammingresources.com/tcl.html (http://www.freeprogrammingresources.com/tcl.html) A large list of helpful resources.
http://www.tcl.tk/man/ (http://www.tcl.tk/man/) Tcl/Tk manual pages.
http://www.mit.edu/afs/sipb/user/golem/doc/tcltk-iap2000/TclTk1.html (http://www.mit.edu/afs/sipb/user/golem/doc/tcltk-iap2000/TclTk1.html) A tutorial describing many commands by describing their implementation in a short program.
http://www.beedub.com/book/ (http://www.beedub.com/book/) Several chapters from Practical Programming in Tcl and Tk, by Welch and Jones.

OpenFresco Interpreter

The main objects OpenFresco for hybrid simulation will be explained using the OpenFresco interpreter. The interpreter is an extension of the Tcl scripting language. The OpenFresco interpreter adds commands to Tcl for the experimental setup and control of hybrid simulations.

OpenFresco has commands for:

- Modeling – create nodes and experimental elements
- Sites – specify the experimental sites (e.g. laboratories or computational sites)
- Setups – specify what experimental setups are used in the laboratories
- Controls – specify what experimental controllers are used in the laboratories
- Signal Filters – specify how to filter experimental signals
- Recorders – specify what to record during a hybrid simulation

CHAPTER 2

expControl Commands

These commands are used to construct the expControl objects. The expControl objects are used to interface the different control and data acquisition systems in the laboratories. In addition, these commands can be utilized to simulate physical specimens. The expSetup objects can be defined using the expControl objects.

In This Chapter

dSpace Experimental Control

This command is used to construct a dSpace experimental control object.

**expControl dSpace $tag $type boardName <-ctrlFilters (5 $filterTag)>
<-daqFilters (5 $filterTag)>**

$tag	unique control tag
$type	Simulink predictor-corrector model type ▪ 1 = using Displacement ▪ 2 = using Displacement and Velocity ▪ 3 = using Displacement, Velocity, and Acceleration
boardName	name of dSpace board (DS1103 and DS1104 are available)
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

EXAMPLE:

```
# Define experimental signal filter
# --------------------------------
# expSignalFilter ErrorSimUndershoot $tag $error
expSignalFilter ErrorSimUndershoot 1 0.01

# Define experimental control
# --------------------------
expControl  dSpace  1  1  DS1103 -ctrlFilters 1 0 0 0 0
```

The above example command is used to talk to a Simulink model which is using displacements for prediction and correction running on a DS1103 digital-signal-processor board. ErrorSimUndershoot signal filter is applied to the displacements that are being sent to the control system.

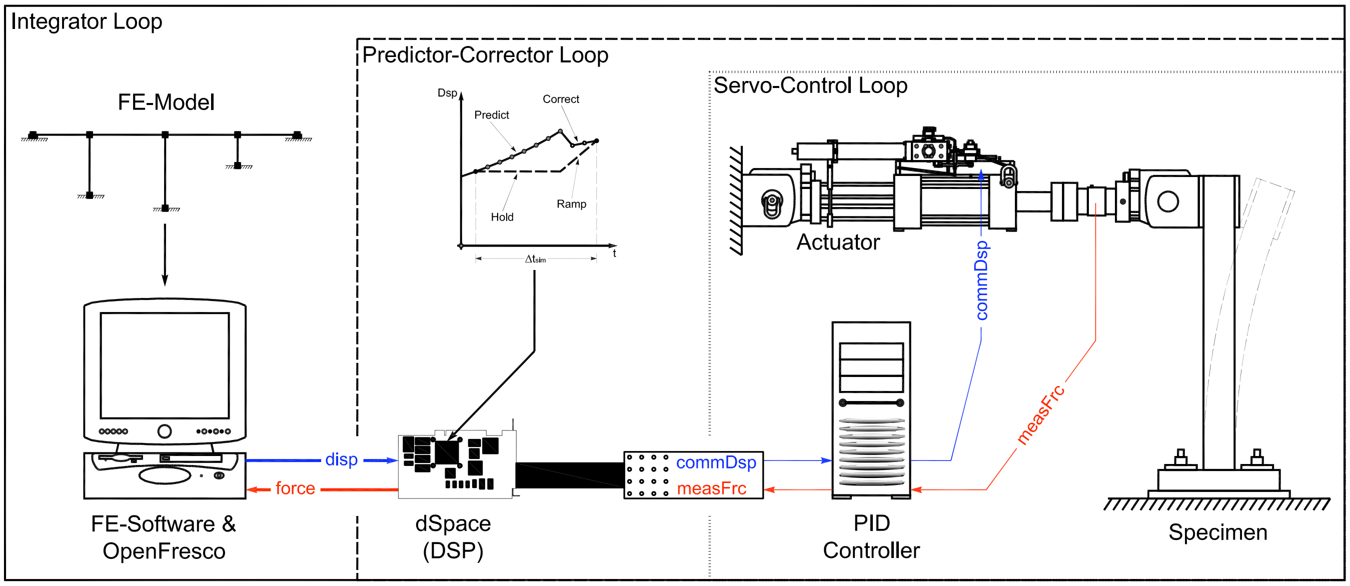


Figure 2: dSpace Experimental Control

The valid queries to a dSpace experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq forces:	daqForce, daqForces

Reference:

http://www.dspaceinc.com/ww/en/inc/home/products/hw/singbord.cfm
(http://www.dspaceinc.com/ww/en/inc/home/products/hw/singbord.cfm)

LabVIEW Experimental Control

This command is used to construct a LabVIEW experimental control object. LabVIEW is a software used to control the MTS MiniMost and LBCB at UIUC. LabVIEW uses the LabVIEW NTCP plugin communication protocol to communicate over a single persistent TCP connection. TR-2004-58, a NEESit document contains the specifics of the LabVIEW NTCP plugin. This command must be used in conjunction with the expControlPoint command.

expControl LabVIEW $tag ipAddr <$ipPort> –trialCP $cpTag1 … –outCP $cpTag2 … <-ctrlFilters (5 $filterTag)> <-daqFilters (5 $filterTag)>

$tag	unique control tag
ipAddr	IP address of LabVIEW computer
$ipPort	IP port number of LabVIEW computer (optional, default = 44000)
$cpTag	tags of previously defined control points
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

EXAMPLE:

```
# Define experimental control points
# ---------------------------------
expControlPoint  1  1  ux  disp  -fact 0.003  -lim -0.01 0.01
expControlPoint  2  1  ux  disp  -fact 0.003  ux  force  -fact [expr 18.0/7.0]

# Define experimental control
# --------------------------
expControl  LabVIEW  1  “130.126.242.175”  44000  -trialCP 1  -outCP 2
```

The above example command uses an IP address of “130.126.242.175” with 44000 as the port number. -trialCP and -outCP fields use the previously defined experimental control points. For a detailed explanation of the expControlPoint command, refer to Chapter 7 of this manual.

The valid queries to a LabVIEW experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control forces:	ctrlForce, ctrlForces
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq forces:	daqForce, daqForces

Reference:

Hubbard, P., Calderon, J., Gose, S. (2004). “Protocol Specification for the LabView NTCP plugin.” NEESit Technical Report (TR-2004-58).

MTSCsi Experimental Control

This command is used to construct a MTSCsi experimental control object. The MTSCsi (MTS Computer Simulation Interface) experimental control object is used to communicate with MTS hardware through the CSI API.

**expControl MTSCsi $tag cfgFile <$rampTime> <-ctrlFilters (5 $filterTag)>
<-daqFilters (5 $filterTag)>**

$tag	unique control tag
cfgFile	path of Csi configuration file including file name with .mtscs extension*****
<$rampTime>	ramp time [sec] (optional, default = 0.02)
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

NOTE: * The cfgFile field should be in quotes especially when there is a space in the path name, and forward slashes should be used in the path name.

EXAMPLE:

```
expControl  MTSCsi  1  “C:/MTSCsi/Example/OpenFresco_uNEES.mtscs”  0.1
```

The above example command uses a path of “C:/MTSCsi/Example/OpenFresco_uNEES.mtscs” and a ramp time of 0.1 seconds.

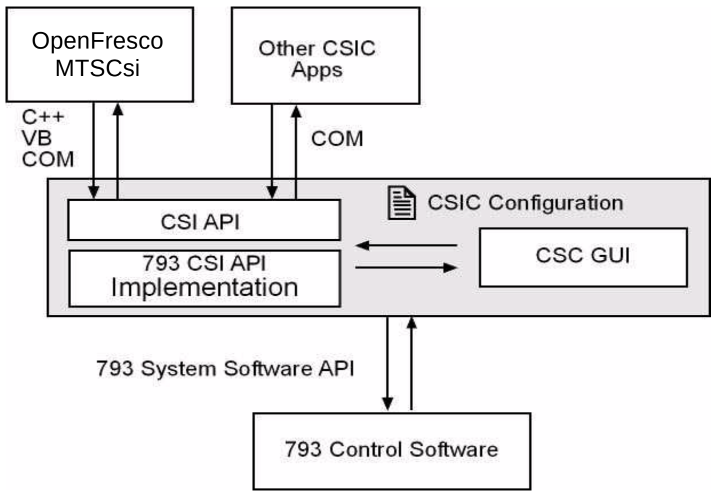


Figure 3: MTSCsi Experimental Control

The valid queries to a MTSCsi experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control forces:	ctrlForce, ctrlForces
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq forces:	daqForce, daqForces

The following is an excerpt from the MTS “Computer Simulation Interface and Configurator” manual:

“The MTS Computer Simulation Interface and Configurator (CSIC) is an application which provides a high level programming interface for users to connect their test application to an MTS controller and execute common structural testing commands and data acquisition operations. Users with minimum programming experience in Microsoft VB or C++, can take advantage of the simplified graphical user interface (GUI) of the Configurator to create their test application. Thus, they can focus on the structural analysis rather than learning the detailed, low level MTS 793 system programming libraries to interface with their MTS controller.

This software package consists of two parts:

1. The Computer Simulation Configurator (CSC) program with easy to use GUI allows the user to define the MTS controller’s hardware resources that correspond to the control points, degrees of freedom, and feedback vectors defined in the analysis software.

2. The Computer Simulation Interface (CSI) API which is the MTS 793 high level programming interface in C++, VB, and COM. This API is a wrapper of the existing 793 system software APIs. After the user creates the configuration using the Computer Simulation Configurator, they can load the configuration to the test program and interface with the MTS controller using the high level API.”

Reference:

http://www.mts.com/en/civil/Seismic/index.asp (http://www.mts.com/en/civil/Seismic/index.asp) - This website contains a link to a web-seminar on hybrid simulation.

SCRAMNet Experimental Control

This command is used to construct a SCRAMNet experimental control object. SCRAMNet (Shared Common RAM Network) is a real-time communications network, based on a replicated, shared memory concept. Each computer has a 2 MB memory module that is mirrored to all other computers in its network. Data written locally to one computer is copied to the other computers in the network. Figure 4 shows how SCRAMNet is implemented at nees@berkeley.

expControl SCRAMNet $tag $memOffset $numActCh <-ctrlFilters (5 $filterTag)> <-daqFilters (5 $filterTag)>

$tag	unique control tag
$memOffset	memory offset from SCRAMNet base memory address [bytes]
$numActCh	number of actuator channels in the control system
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

EXAMPLE:

```
expControl  SCRAMNet  1  381020  8
```

The above example command uses a memory offset of 381020 bytes and 8 actuator channels.

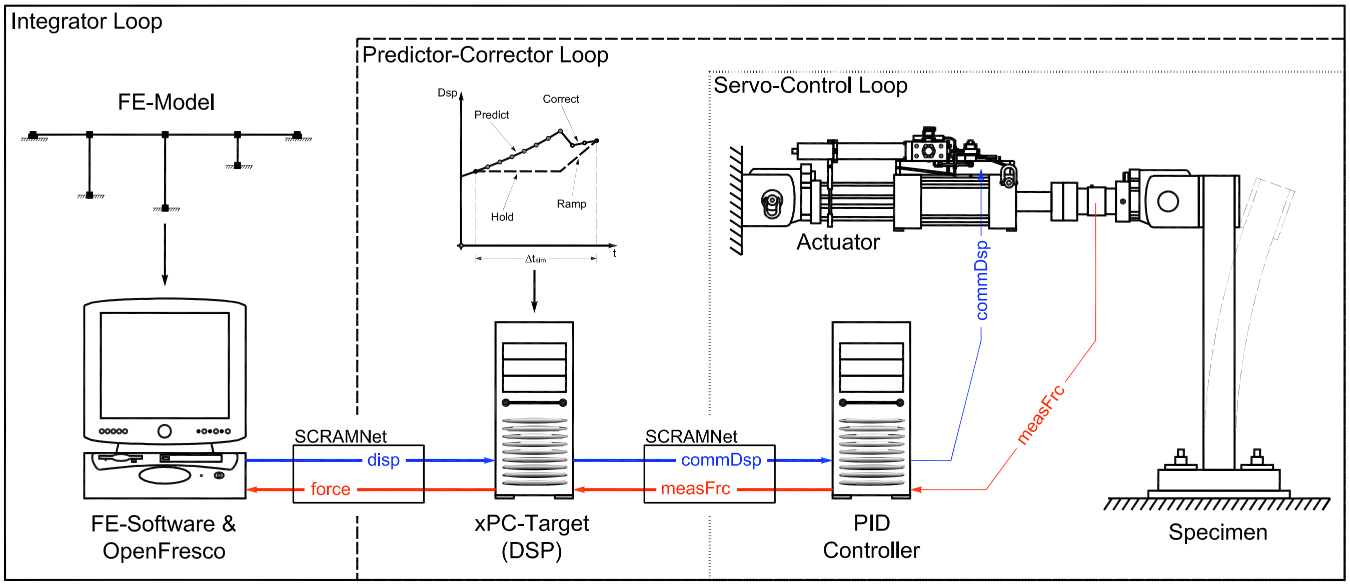


Figure 4: SCRAMNet Experimental Control

The valid queries to a SCRAMNet experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• control forces:	ctrlForce, ctrlForces
• control time:	ctrlTime, ctrlTimes
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations
• daq forces:	daqForce, daqForces
• daq time:	daqTime, daqTimes

Reference:

http://nees.berkeley.edu/Facilities/network-computers.shtml
(http://nees.berkeley.edu/Facilities/network-computers.shtml)

SimDomain Experimental Control

This command is used to construct a Simulation Domain experimental control object. The SimDomain command makes available in OpenFresco the entire OpenSees domain, which includes the material, section and element libraries.

This makes it possible to simulate an actual experimental specimen using OpenSees. It can be used to test the computational model, experimental setup, and network communication to ensure that all non-experimental aspects of a hybrid simulation are functioning properly before conducting an actual experiment. This command must be used in conjunction with the expControlPoint command.

**expControl SimDomain $tag –trialCP $cpTag1 … –outCP $cpTag2 …
<-ctrlFilters (5 $filterTag)> <-daqFilters (5 $filterTag)>**

$tag	unique control tag
$cpTag	tags of previously defined control points
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

EXAMPLE:

```
# Define materials
# ----------------
# uniaxialMaterial Steel02 $matTag $Fy $E $b $R0 $cR1 $cR2 $a1 $a2 $a3 $a4
uniaxialMaterial Elastic 1 [expr 2.26*29000]
uniaxialMaterial Steel02 2 [expr 1.5*54] 146966.4 0.01 18.5 0.925 0.15 0.0 1.0 0.0 1.0

# Define sections
# ---------------
# section Aggregator $secTag $matTag1 $string1 $matTag2 $string2
section Aggregator 2 1 P 2 Mz

# Define coordinate transformation
# --------------------------------
# geomTransf Linear $transfTag
geomTransf Linear 1

# Define element
# --------------
# element nonlinearBeamColumn $eleTag $iNode $jNode $numIntgrPts $secTag $transfTag
element nonlinearBeamColumn 1 1 2 5 2 1

# Define control points
# ---------------------
# expControlPoint tag nodeTag dir resp <-fact f> <-lim l u> ...
expControlPoint 1 2  ux disp uy disp rz disp
expControlPoint 2 2  ux disp ux force uy disp uy force rz disp rz force

# Define experimental control
# ---------------------------
# expControl SimDomain $tag -trialCP cpTags -outCP cpTags
expControl SimDomain  1  -trialCP 1  -outCP 2
```

The above example uses a trial control point of tag 1 and an output control point of tag 2. The trial control point sends the displacements in the ux, uy, and rz directions for node 2 of the nonlinear beam column element defined in OpenSees. Control point 2 measures the displacements and forces in the ux, uy, and rz directions at the same node. For a detailed explanation of the expControlPoint command, refer to Chapter 7 of this manual.

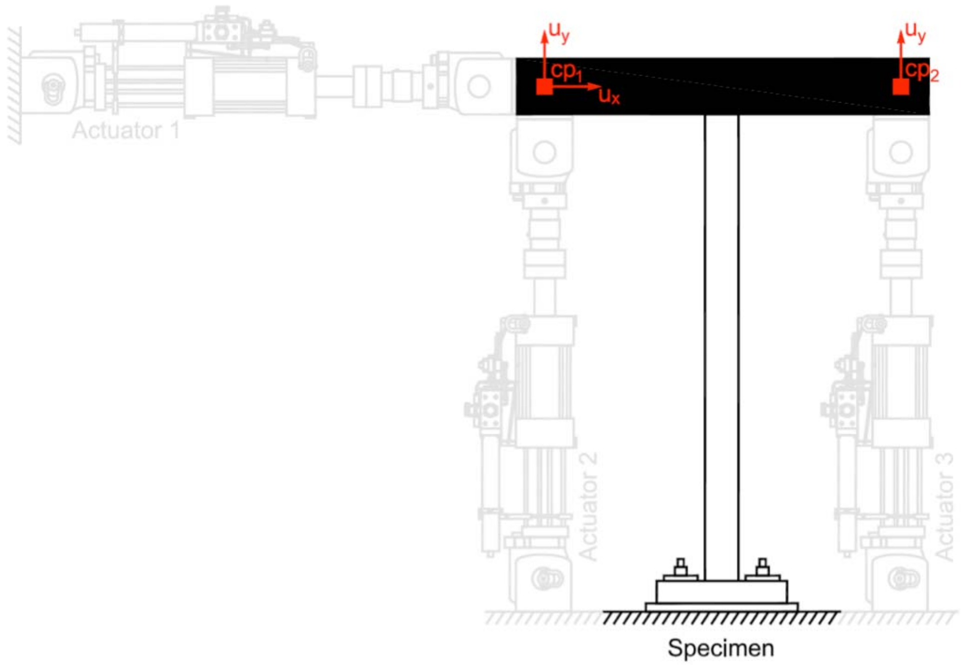


Figure 5: SimDomain Experimental Control

The valid queries to a SimDomain experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• control forces:	ctrlForce, ctrlForces
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations
• daq forces:	daqForce, daqForces

Reference:

http://opensees.berkeley.edu/OpenSees/manuals/usermanual/index.html
(http://opensees.berkeley.edu/OpenSees/manuals/usermanual/index.html)

SimFEAdapter Experimental Control

This command is used to construct a Simulation Finite-Element-Adapter experimental control object. This controller is used to communicate with an adapter element of another finite element software over a single persistent TCP/IP connection. It allows the user to couple two different finite element software packages together to perform a single analysis.

expControl SimFEAdapter $tag ipAddr $ipPort <-ctrlFilters (5 $filterTag)> <-daqFilters (5 $filterTag)>

$tag	unique control tag
ipAddr	IP address of slave process
$ipPort	IP port of slave process
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

EXAMPLE:

```
# Define experimental control
# ---------------------------
# expControl SimFEAdapter $tag ipAddr $ipPort
expControl SimFEAdapter 1 "127.0.0.1" 44000
```

The above example uses the SimFEAdapter experimental controller with an IP address of 127.0.0.1 and port number of 44000.

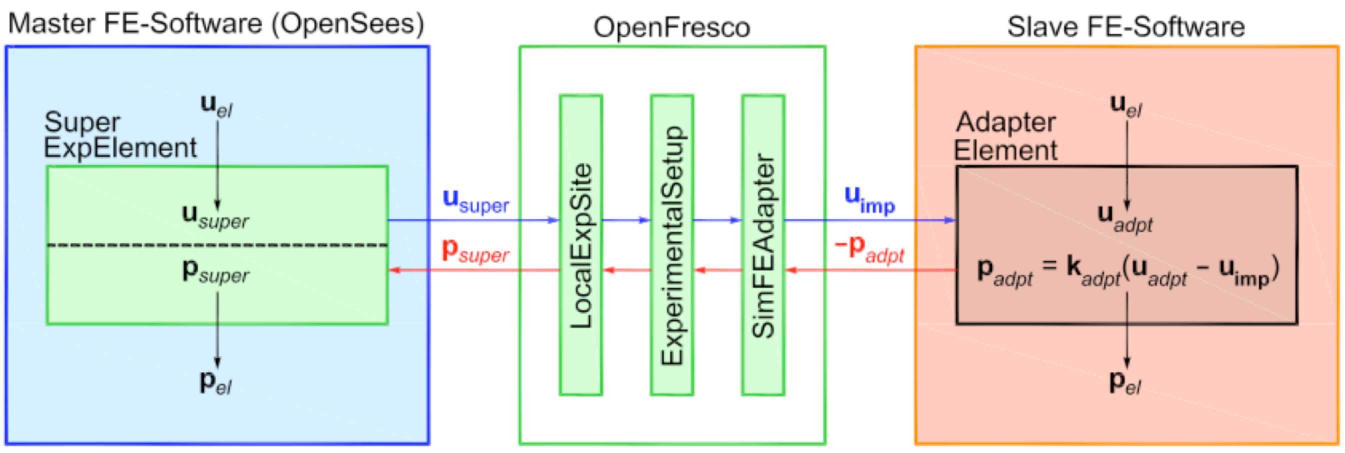


Figure 6: SimFEAdapter Experimental Control

The valid queries to a SimFEAdapter experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control forces:	ctrlForce, ctrlForces
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq forces:	daqForce, daqForces

Reference:

Schellenberg, A., Huang, Y., Mahin, S. A. (2008). “Structural FE-Software Coupling through the Experimental Software Framework, OpenFresco.” Proceedings, 14th World Conference on Earthquake Engineering. Beijing, China.

SimUniaxialMaterials Experimental Control

This command is used to construct a SimUniaxialMaterials experimental control object. It can be used to simulate a specimen with 1 to n uncoupled degrees of freedom using 1 to n OpenSees uniaxial material models. Since the material models simulate the forces that the 1 to n actuator load-cells would measure for a given set of actuator displacements, the units of the specified material values are force and displacement and not stress and strain.

expControl SimUniaxialMaterials $tag $matTag1 … <-ctrlFilters (5 $filterTag)>
<-daqFilters (5 $filterTag)>

$tag	unique control tag
$matTag	tags of previously defined uniaxial materials
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

EXAMPLE:

```
uniaxialMaterial  Elastic  2  5.6
expControl  SimUniaxialMaterials  1  2
```

The above example command uses an OpenSees uniaxial elastic material model with a tag of 2 and a stiffness of 5.6. The OpenSees Command Language Manual contains a complete list of all available uniaxial material models.

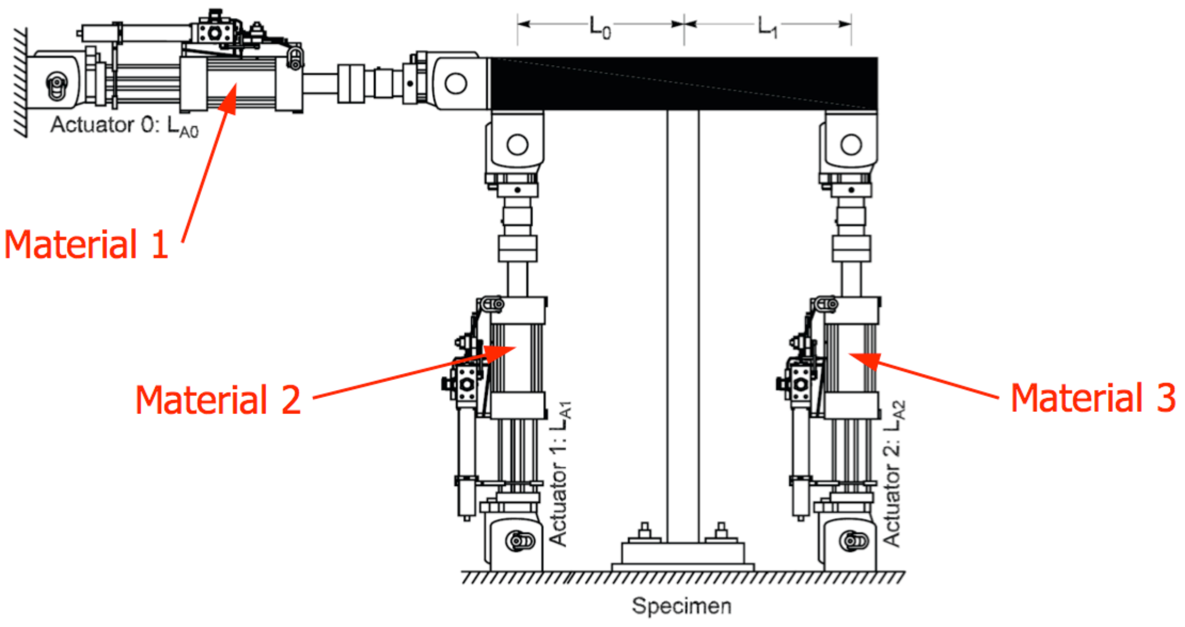


Figure 7: SimUniaxialMaterials Experimental Control

The valid queries to a SimUniaxialMaterials experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq forces:	daqForce, daqForces

Reference:

http://opensees.berkeley.edu/OpenSees/manuals/usermanual/index.html
(http://opensees.berkeley.edu/OpenSees/manuals/usermanual/index.html)

xPCtarget Experimental Control

This command is used to construct a xPCtarget experimental control object. In this real-time system, the target PC is accessed from the host PC through an Ethernet connection. The xPC-Target digital-signal-processor functions as a link between the host PC and the controllers. It runs a Simulink predictor-corrector model in real-time.

expControl xPCtarget $tag $type ipAddr $ipPort appName appPath
<-ctrlFilters (5 $filterTag)> <-daqFilters (5 $filterTag)>

$tag	unique control tag
$type	Simulink predictor-corrector model type ▪ 1 = using Displacement ▪ 2 = using Displacement and Velocity ▪ 3 = using Displacement, Velocity, and Acceleration
ipAddr	IP address of xPC-Target
$ipPort	IP port of xPC-Target
appName	name of Simulink application to be loaded (without the .dlm extension)
appPath	path to Simulink application*
$filterTag	id of previously defined filter tags; size of filter tag id is 5 (entries: [disp, vel, accel, force, time])

***NOTE:** Preferably, the appPath should be in quotes especially when there is a space in the path name, and forward slashes should be used in the path name.

EXAMPLE:

```
expControl  xPCtarget  1  1  “192.168.2.20”  22222  HybridControllerD3D3_1Act
"D:/PredictorCorrector/RTActualTestModels/cmAPI-xPCTarget-STS"
```

The above example command uses a Simulink model called HybridControllerD3D3_1Act which utilizes displacements for prediction and correction. It communicates to an xPC-Target machine with an IP address of “192.168.2.20” and a port of 22222. The path to the Simulink model is "D:/PredictorCorrector/RTActualTestModels/cmAPI-xPCTarget-STS".

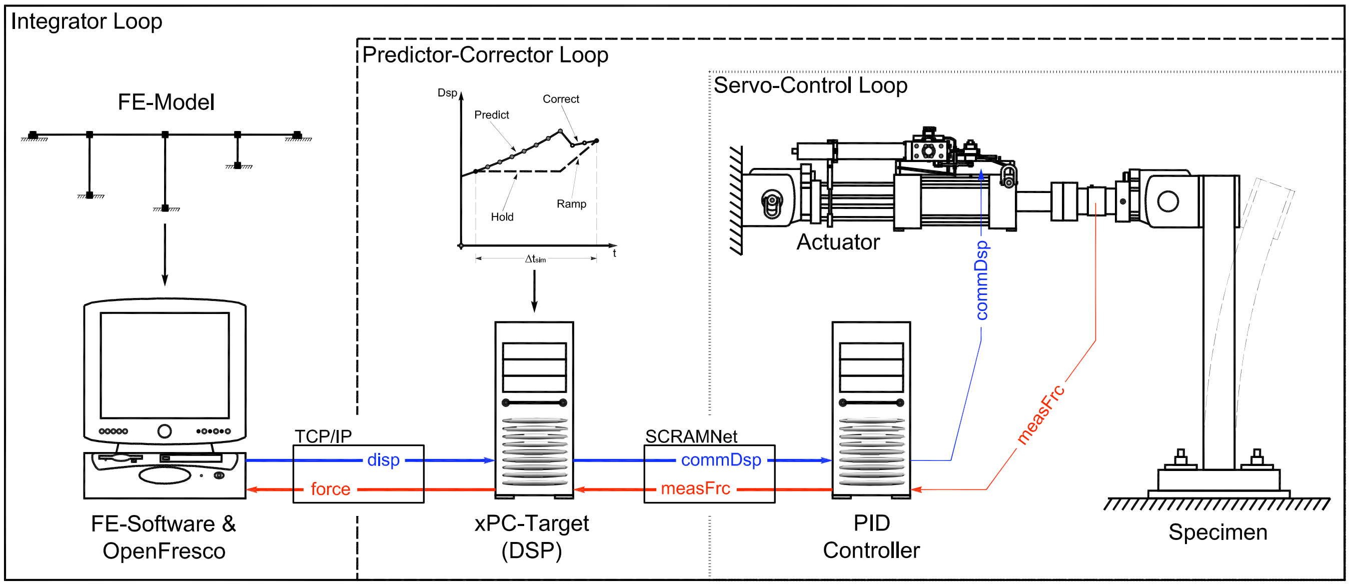


Figure 8: xPCtarget Experimental Control

The valid queries to a xPCtarget experimental control when creating an *ExpControlRecorder* object are:

• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq forces:	daqForce, daqForces

Reference:

http://nees.berkeley.edu/Facilities/network-computers.shtml
(http://nees.berkeley.edu/Facilities/network-computers.shtml)

CHAPTER 3

expElement Commands

These commands are used to construct the expElement objects. OpenFresco provides two ways of defining an experimental element. The first, and more common, way is to define the experimental element on the middle-tier server side and using a generic-client element in the finite element software as shown in Figure 6a. This method is highlighted in **YELLOW** for all the expElement commands. A second option is to define an experimental element directly in the finite element software and therefore not using any element on the middle-tier server side, as shown in Figure 6b. Specifically, if OpenSees is used as the finite element software, the commands highlighted in **GRAY** can be utilized. Other finite element analysis software may be used with this option also. The `recorder` command can be used with each element to capture the response quantities during simulation.

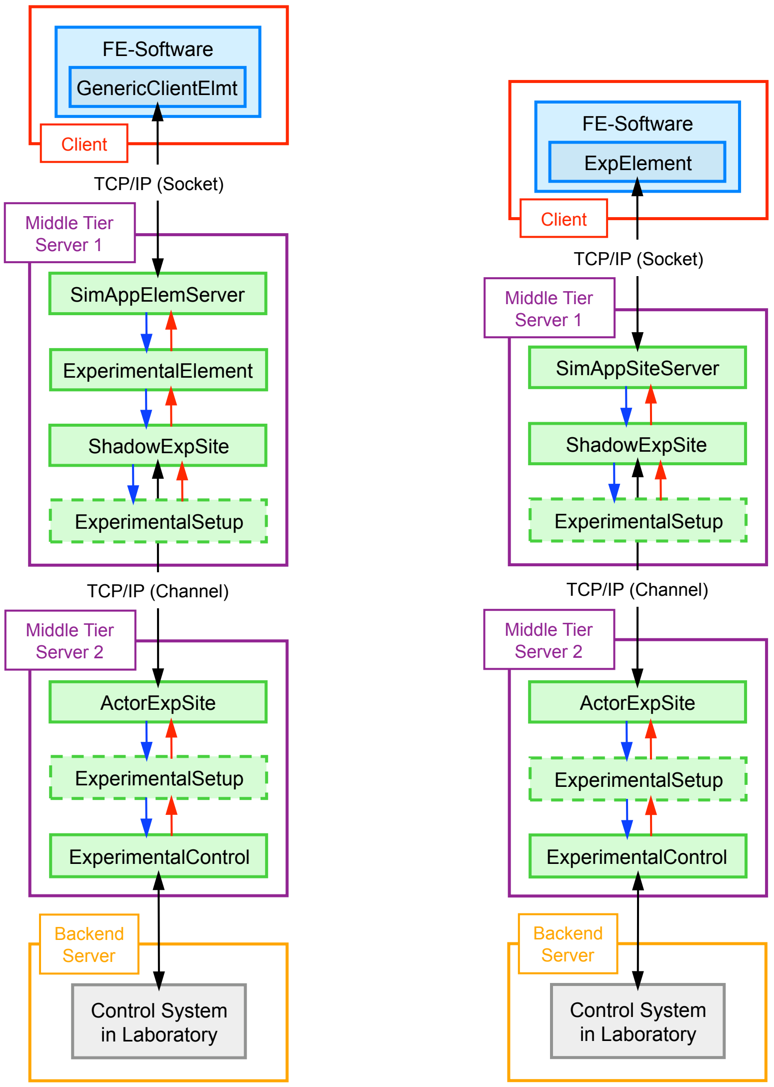


Figure 9: Generic client element (left) and experimental element (right) defined in FE-software

In This Chapter

beamColumn (2D or 3D) Experimental Element

This command is used to construct a beamColumn experimental element object. Two nodes define this element.

expElement beamColumn $tag $iNode $jNode $transTag –site $siteTag
–initStif $Kij … <–iMod> <–rho $rho>

expElement beamColumn $tag $iNode $jNode $transTag –server $ipPort <ipAddr>
<–ssl> <–dataSize $size> –initStif Kij … <–iMod> <–rho $rho>

$tag	unique element tag
$iNode,$jNode	end node tags
$transTag	tag of previously defined coordinate transformation object
$siteTag	tag of previously defined site object
$Kij	initial stiffness matrix components (row-wise) of the element
-iMod	error correction using Nakashima's initial stiffness modification (optional, default = false)
$rho	mass per unit length (optional, default = 0.0)
$ipPort	IP port of middle-tier server
ipAddr	IP address of middle-tier server (optional)
-ssl	secure transactions using OpenSSL (optional)
$size	data size being sent (optional)

EXAMPLE:

```
# Define geometry for model
# -------------------------
set mass 0.12
# node $tag $xCrd $yCrd $mass
node  1     0.0   0.00
node  2   108.0 -54.00
node  3   216.0 -42.00
node  4   324.0   0.00
node  5     0.0   0.00  -mass $mass $mass 116.6
node  6   108.0   0.00  -mass $mass $mass 116.6
node  7   216.0   0.00  -mass $mass $mass 116.6
node  8   324.0   0.00  -mass $mass $mass 116.6
node  9   108.0   0.00  -mass $mass $mass 116.6
node 10   216.0   0.00  -mass $mass $mass 116.6

# Define experimental site
# ------------------------
# expSite RemoteSite $tag <-setup $setupTag> $ipAddr $ipPort <$dataSize>
expSite  RemoteSite  1  "127.0.0.1"  8090
expSite  RemoteSite  2  "127.0.0.1"  8091
# Define geometric transformation
# ------------------------
# geomTransf type $tag
geomTransf  Linear  1
geomTransf  Corotational  2

# Define experimental elements
# ---------------------------
expElement  beamColumn  2  2  9  1  -site 1  -initStif 1213 0 0 0 11.2 -302.4 0 -302.4 10886.4
```

Nodes 2 and 9 connect the above 2D beamColumn experimental element. A linear coordinate transformation is used. The element is defined with a remote experimental site with “127.0.0.1” as the IP address and 8090 as the port number. The initial stiffness matrix of this element is **K**$_i$.

$$\mathbf{K}_i = \begin{bmatrix} 1213 & 0 & 0 \\ 0 & 11.2 & -302.4 \\ 0 & -302.4 & 10886.4 \end{bmatrix}$$

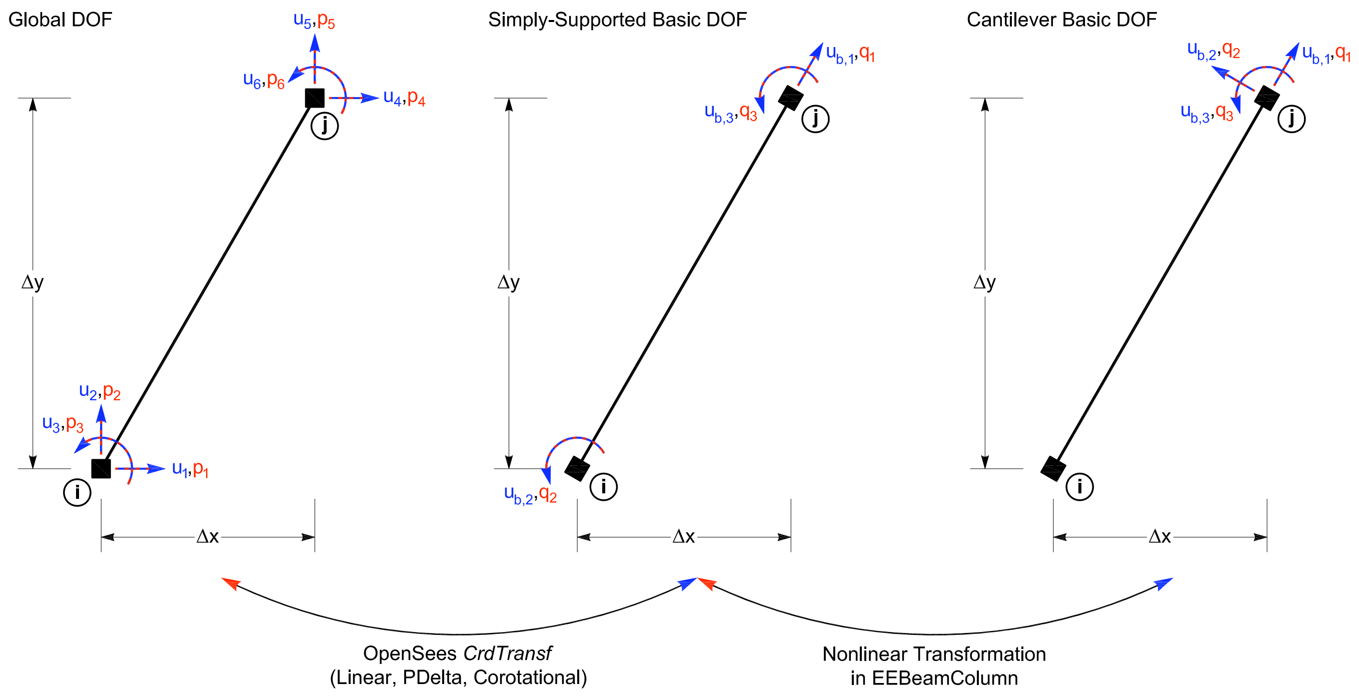


Figure 10: beamColumn Experimental Element

The valid queries to a beamColumn experimental element when creating an *ElementRecorder* (see OpenSees Manual) object are:

• global forces:	force, forces, globalForce, globalForces
• local forces:	localForce, localForces
• basic forces:	basicForce, basicForces, daqForce, daqForces
• control displacements:	defo, deformation, deformations, basicDefo, basicDeformation, basicDeformations, ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations

Reference for initial stiffness modification:

Nakashima, M. and Kato, H. (1987). “Experimental error growth behavior and error growth controlling on-line computer test control method,” Building Research Institute, BRI-Report No. 123, Ministry of Construction, Tsukuba, Japan,

generic (1D, 2D, or 3D) Experimental Element

This command is used to construct a generic experimental element object. This element is defined by any number of nodes and the degrees of freedom at these nodes.

expElement generic $tag –node $Ndi … –dof $dofNdi … –dof $dofNdj …
–site $siteTag –initStif Kij … <–iMod> <–mass Mij …>

expElement generic $tag –node $Ndi … –dof $dofNdi … –dof $dofNdj …
–server $ipPort <ipAddr> <–ssl> <–dataSize $size> –initStif Kij …
<–iMod> <–mass Mij …>

$tag	unique element tag
$Ndi	n end node tags (n = number of nodes)
$dofNdi, $dofNdj	degrees of freedom for n end nodes
$siteTag	tag of previously defined site object
$Kij	initial stiffness matrix components (row-wise) of the element
-iMod	error correction using Nakashima’s initial stiffness modification (optional, default = false)
$Mij	mass matrix components (row-wise) of element (optional, default = 0.0)
$ipPort	IP port of middle-tier server
ipAddr	IP address of middle-tier server (optional)
-ssl	secure transactions using OpenSSL (optional)
$size	data size being sent (optional)

EXAMPLE:

```
# Define geometry for model
# -------------------------
set mass3 0.04
set mass4 0.02
# node $tag $xCrd $yCrd $mass
node  1     0.0   0.00
node  2   100.0   0.00
node  3     0.0  54.00  -mass $mass3 $mass3
node  4   100.0  54.00  -mass $mass4 $mass4

# Define experimental site
# ------------------------
# expSite LocalSite $tag $setupTag
expSite  LocalSite  2  2
```

```
# Define experimental elements
# ----------------------------
expElement  generic  1  -node 1 3  -dof 1 2  -dof 1 2  -site 2
-initStif 130 150 110 100 150  200 220 180 100 110 180 150 125 100 100 125 200
```

Nodes 1 and 3 connect the above 2D generic experimental element. Translational degrees of freedom 1 and 2 are used for each node. The element is defined using a local experimental site. The initial stiffness matrix of this element is $\mathbf{K}_i$.

$$\mathbf{K}_i = \begin{bmatrix} 130 & 150 & 110 & 100 \\ 150 & 220 & 180 & 100 \\ 110 & 180 & 150 & 125 \\ 100 & 100 & 125 & 200 \end{bmatrix}$$

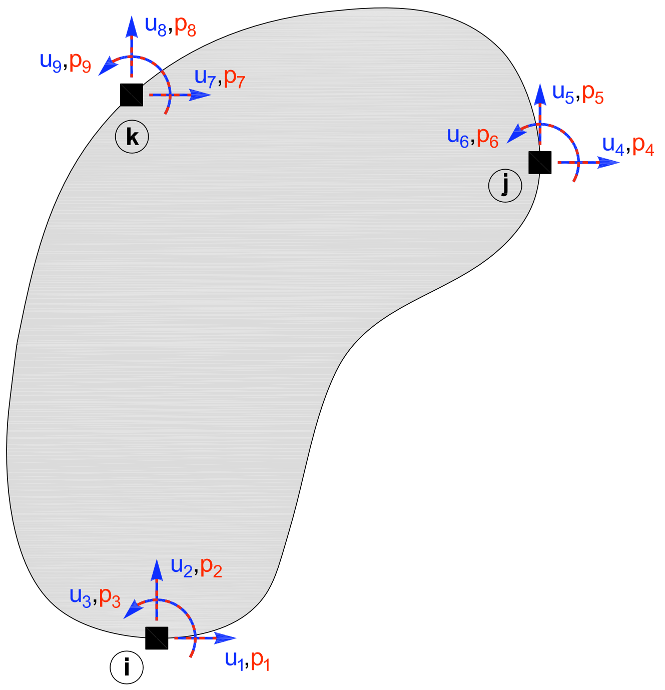


Figure 11: Arbitrary generic Experimental Element

The valid queries to a generic experimental element when creating an *ElementRecorder* (see OpenSees Manual) object are:

• global forces:	force, forces, globalForce, globalForces
• local forces (= global):	localForce, localForces
• basic forces:	basicForce, basicForces, daqForce, daqForces
• control displacements:	defo, deformation, deformations, basicDefo, basicDeformation, basicDeformations, ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations

invertedVBrace (2D) Experimental Element

This command is used to construct an invertedVBrace experimental element object. Three nodes define this element.

expElement invertedVBrace $tag $iNode $jNode $kNode –site $siteTag –initStif Kij … <–iMod> <–nlGeom> <–rho1 rho1> <–rho2 rho2>

expElement invertedVBrace $tag $iNode $jNode $kNode –server $ipPort <ipAddr> <–ssl> <–dataSize $size> –initStif Kij … <–nlGeom> <–rho1 rho1> <–rho2 rho2>

$tag	unique element tag
$iNode, $jNode, $kNode	end node tags according to the configuration in Figure 7
$siteTag	tag of already defined site object
$Kij	initial stiffness matrix components (row-wise) of the element
-iMod	error correction using Nakashima's initial stiffness modification (optional, default = false)
-nlGeom	nonlinear geometry (optional, default = false)
$rho1, $rho2	mass per unit length of brace legs (optional, default = 0.0)
$ipPort	IP port of middle-tier server
ipAddr	IP address of middle-tier server (optional)
-ssl	secure transactions using OpenSSL (optional)
$size	data size being sent (optional)

EXAMPLE:

```
# Define geometry for model
# -------------------------
set mass3 0.04
# node $tag $xCrd $yCrd $mass
node  1     0.0   0.00
node  2   100.0   0.00
node  3    50.0  54.00  -mass $mass3 $mass3

# Define experimental site
# ------------------------
# expSite LocalSite $tag $setupTag
expSite  LocalSite  1  1

# Define experimental elements
# ----------------------------
expElement  invertedVBrace  1  1  2  3  -site 1  -initStif 250 0 0 0 434 0 0 0
```

Nodes 1,2, and 3 connect the above invertedVBrace experimental element. Note the ordering of the nodes in Figure 5. The element is defined using a local experimental site. The initial stiffness of this element is $\mathbf{K}_i$.

$$\mathbf{K}_i = \begin{bmatrix} 250 & 0 & 0 \\ 0 & 434 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$ (if rotational DOF is ignored)

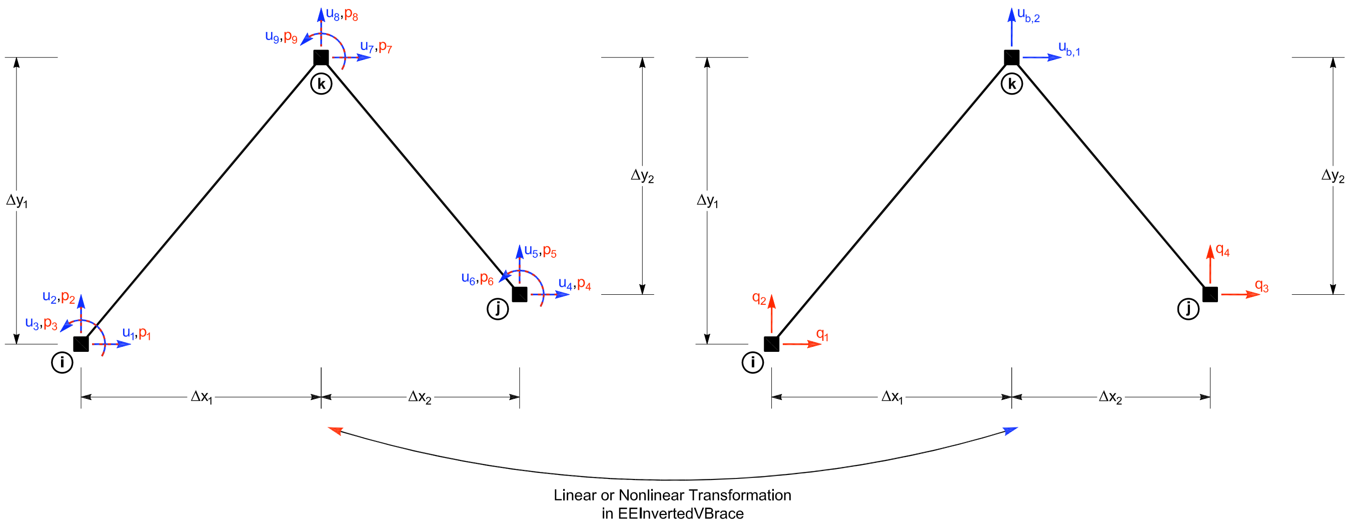


Figure 12: invertedVBrace Experimental Element

The valid queries to a invertedVBrace experimental element when creating an *ElementRecorder* (see OpenSees Manual) object are:

• global forces:	force, forces, globalForce, globalForces
• local forces (= global):	localForce, localForces
• basic forces:	basicForce, basicForces, daqForce, daqForces
• control displacements:	defo, deformation, deformations, basicDefo, basicDeformation, basicDeformations, ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations

truss (1D, 2D, or 3D) Experimental Element

This command is used to construct a truss experimental element object. Two nodes define this element.

expElement truss $tag $iNode $jNode –site $siteTag –initStif $K <–iMod> <–rho $rho>

expElement truss $tag $iNode $jNode –server $ipPort <ipAddr> <–ssl> <–dataSize $size> –initStif $K <–iMod> <–rho $rho>

$tag	unique element tag
$iNode,$jNode	end node tags
$siteTag	tag of previously defined site object
$K	initial stiffness of the element
-iMod	error correction using Nakashima’s initial stiffness modification (optional, default = false)
$rho	mass per unit length (optional, default = 0.0)
$ipPort	IP port of middle-tier server
ipAddr	IP address of middle-tier server (optional)
-ssl	secure transactions using OpenSSL (optional)
$size	data size being sent (optional)

EXAMPLE:

```
# Define geometry for model
# -------------------------
# node $tag $xCrd $yCrd $mass
node 1   0.0  0.0
node 2 144.0  0.0
node 3 168.0  0.0
node 4  72.0 96.0

# Define experimental site
# ------------------------
# expSite LocalSite $tag $setupTag
expSite  LocalSite  2  2

# Define experimental elements
# ----------------------------
expElement  truss  1  3  4  –site  2  -initStif [expr 3000.0*5.0/135.76]
```

Nodes 3 and 4 connect the above 2D truss experimental element. The element is defined using a local experimental site. The initial stiffness of this element is [expr 3000.0*5.0/135.76] = 110.5.

$$\mathbf{K}_i = \left[110.5\right]$$

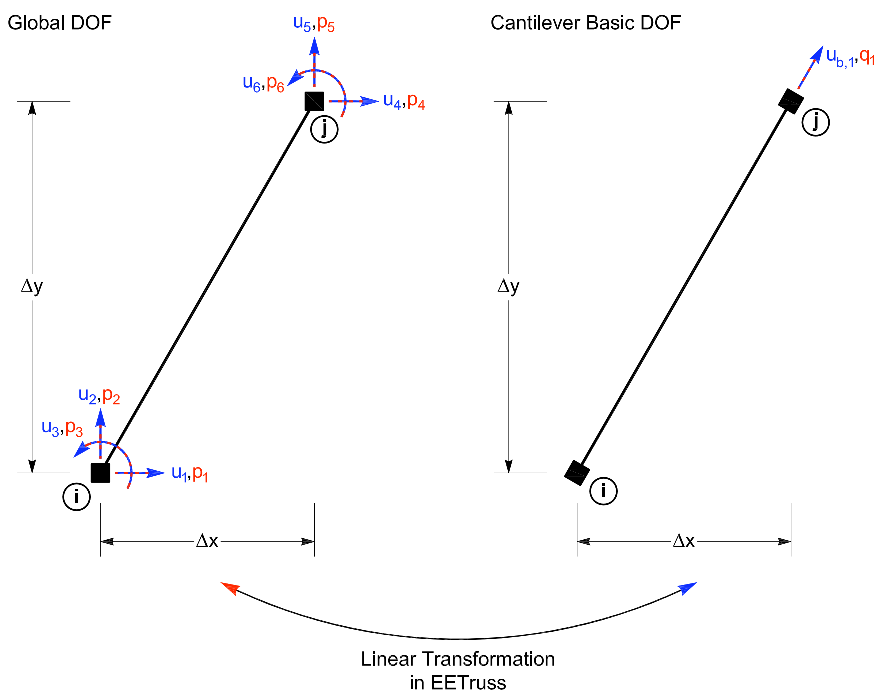


Figure 13: truss Experimental Element

The valid queries to a truss experimental element when creating an *ElementRecorder* (see OpenSees Manual) object are:

• global forces:	force, forces, globalForce, globalForces
• local forces:	localForce, localForces
• basic forces:	basicForce, basicForces, daqForce, daqForces
• control displacements:	defo, deformation, deformations, basicDefo, basicDeformation, basicDeformations, ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations

corotTruss (1D, 2D, or 3D) Experimental Element

This command is used to construct a corotational truss experimental element object. A corotational formulation adopts a set of corotational axes, which rotate with the element, thus taking into account an exact geometric transformation between local and global frames of reference. Two nodes define this element.

expElement corotTruss $tag $iNode $jNode –site $siteTag –initStif $K <–iMod> <–rho $rho>

expElement corotTruss $tag $iNode $jNode –server $ipPort <ipAddr> <–ssl> <–dataSize $size> –initStif $K <–iMod> <–rho $rho>

$tag	unique element tag
$iNode,$jNode	end node tags
$siteTag	tag of previously defined site object
$K	initial stiffness of the element
-iMod	error correction using Nakashima's initial stiffness modification (optional, default = false)
$rho	mass per unit length (optional, default = 0.0)
$ipPort	IP port of middle-tier server
ipAddr	IP address of middle-tier server (optional)
-ssl	secure transactions using OpenSSL (optional)
$size	data size being sent (optional)

EXAMPLE:

```
# Define geometry for model
# -------------------------
# node $tag $xCrd $yCrd $mass
node 1   0.0  0.0
node 2 144.0  0.0
node 3 168.0  0.0
node 4  72.0 96.0

# Define experimental site
# ------------------------
# expSite LocalSite $tag $setupTag
expSite  LocalSite  2  2

# Define experimental elements
# ----------------------------
expElement  corotTruss  1  3  4  –site  2  -initStif [expr 3000.0*5.0/135.76]
```

Nodes 3 and 4 connect the above 2D corotational truss experimental element. The element is defined using a local experimental site. The initial stiffness of this element is [expr 3000.0*5.0/135.76] = 110.5.

$$\mathbf{K}_i = [110.5]$$

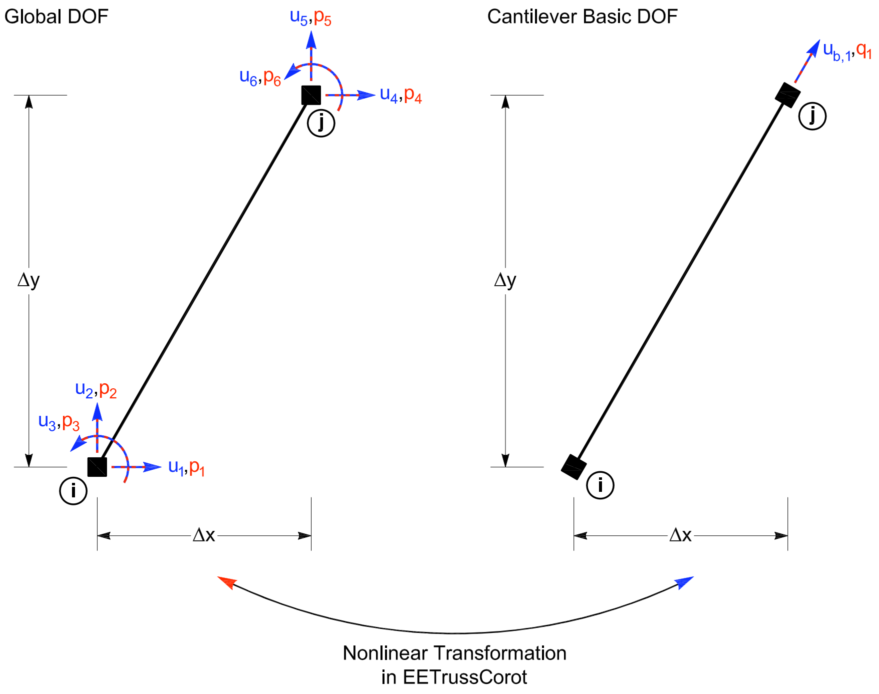


Figure 14: corotTruss Experimental Element

The valid queries to a corotational truss experimental element when creating an *ElementRecorder* (see OpenSees Manual) object are:

• global forces:	force, forces, globalForce, globalForces
• local forces:	localForce, localForces
• basic forces:	basicForce, basicForces, daqForce, daqForces
• control displacements:	defo, deformation, deformations, basicDefo, basicDeformation, basicDeformations, ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations

twoNodeLink (1D, 2D, or 3D) Experimental Element

This command is used to construct a twoNodeLink experimental element object. Two nodes define this element, which do not need to be at the same location. This element can have 1 to 6 degrees of freedom, where only the transverse and rotational degrees of freedom are coupled as long as the element does not have zero length. In addition, if the element length is larger than zero, the user can optionally specify how the P-Delta moments around the local x- and y-axis are distributed among a moment at node i, a moment at node j and a shear couple. The sum of these three ratios is always equal to 1. It is important to recognize that if this element has zero length, it does not consider the geometry as given by the nodal coordinates, but utilizes the local orientation vectors to determine the directions of the springs.

```
expElement twoNodeLink $tag $iNode $jNode –dir $dirs –site $siteTag
        –initStif $Kij … <–orient <$x1 $x2 $x3> $y1 $y2 $y3> <-pDelta (4 $Mratio)>
        <–iMod> <–mass $m>
```

```
expElement twoNodeLink $tag $iNode $jNode –dir $dirs –server $ipPort <ipAddr>
        <–ssl> <–dataSize $size> –initStif $Kij … <–orient <$x1 $x2 $x3> $y1 $y2
        $y3> <-pDelta (4 $Mratio)> <–iMod> <–mass $m>
```

$tag	unique element tag
$iNode,$jNode	end node tags
$dir	force directions (1, 1-3 or 1-6)
$siteTag	tag of previously defined site object
$Kij	initial stiffness matrix components (row-wise) of element
$x1, $x2, $x3, $y1, $y2, $y3	orientation vectors for the element. x1, x2, and x3 are vector components in the global coordinates defining the local x-axis. y1, y2, and y3 are the same except that it defines the y vector which lies in the local x-y plane for the element. (optional, default = global X and Y)
$Mratio	P-Delta moment contribution ratio, size of ratio vector is 4 (entries: [My-$iNode, My-$jNode, Mz-$iNode, Mz-$jNode]) My-$iNode + My-$jNode <= 1.0, Mz-$iNode + Mz-$jNode <= 1.0. Remaining P-Delta moments are resisted by shear couples. (optional, default = [0.0 0.0 0.0 0.0])
-iMod	error correction using Nakashima’s initial stiffness modification (optional, default = false)
$m	mass (optional, default = 0.0)
$ipPort	IP port of middle-tier server
ipAddr	IP address of middle-tier server (optional)

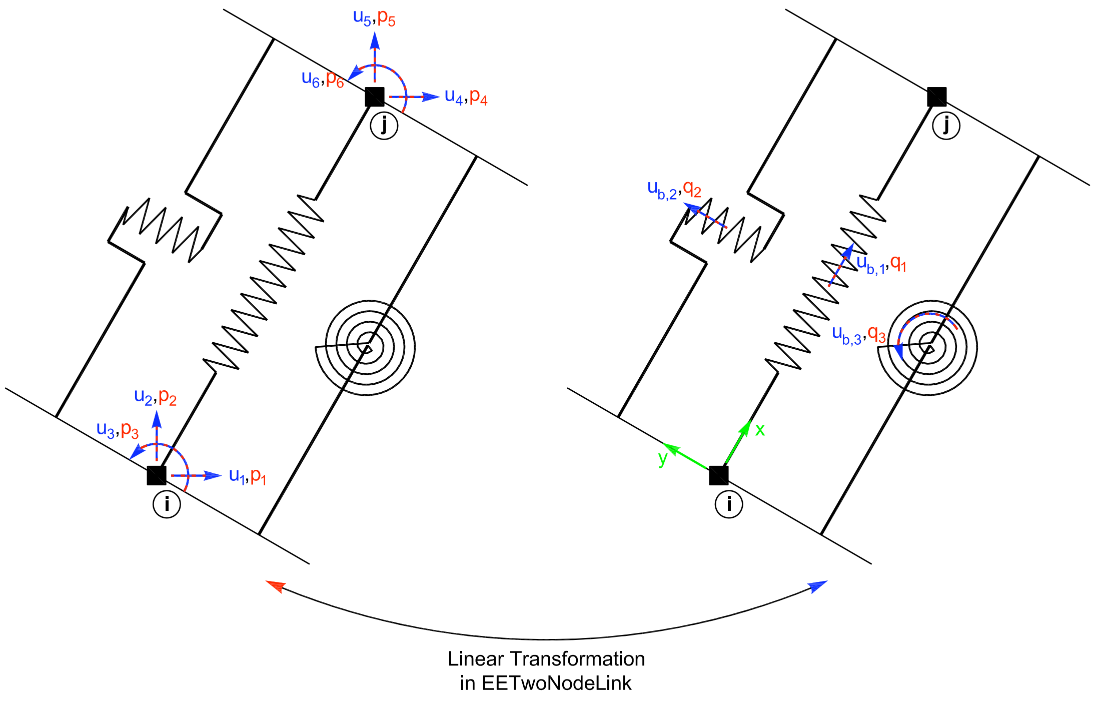
-ssl	secure transactions using OpenSSL (optional)
$size	data size being sent (optional)

EXAMPLE:

```
# Define geometry for model
# ------------------------
set mass3 0.04
set mass4 0.02
# node $tag $xCrd $yCrd $mass
node  1     0.0   0.00
node  2   100.0   0.00
node  3     0.0  54.00  -mass $mass3 $mass3
node  4   100.0  54.00  -mass $mass4 $mass4

# Define experimental site
# ------------------------
# expSite LocalSite $tag $setupTag
expSite  RemoteSite  1  "169.229.203.152"  8090

# Define experimental elements
# ----------------------------
expElement  twoNodeLink  1  1  3  -dir  2  -site  1  -initStif  2.8  -orient  0  1  0  -1  0
0
```

Nodes 1 and 3 connect the above 2D twoNodeLink experimental element. The element is defined using a remote experimental site with an IP address of "169.229.203.152" and a port number of 8090. The initial stiffness of this element is 2.8. The orientation is set such that the element x-axis points in the global Y-axis direction and the element y-axis points in the negative X-direction in the global coordinate system.

$$\mathbf{K}_i = [2.8]$$



$$u_{b,1} = u_4 - u_1$$
$$u_{b,2} = u_5 - u_2$$
$$-0.5L(u_3 + u_6)$$
$$u_{b,3} = u_6 - u_3$$

Figure 15: twoNodeLink Experimental Element

The valid queries to a twoNodeLink experimental element when creating an *ElementRecorder* (see OpenSees Manual) object are:

• global forces:	force, forces, globalForce, globalForces
• local forces:	localForce, localForces
• basic forces:	basicForce, basicForces, daqForce, daqForces
• control displacements:	defo, deformation, deformations, basicDefo, basicDeformation, basicDeformations, ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations
• basic defo and forces:	defoANDforce, deformationANDforce, deformationsANDforces

CHAPTER 4

expSetup Commands

These commands are used to construct the expSetup objects. The expSetup objects transform between the basic experimental element degrees of freedom in OpenFresco and the actuator degrees of freedom in the laboratory. Linear and non-linear transformations are available. The expSetup objects can be defined using the expControl objects.

In This Chapter

Aggregator Experimental Setup

This command is used to construct an Aggregator experimental setup object. The Aggregator experimental setup object combines different experimental setups into one setup.

expSetup Aggregator $tag <–control $ctrlTag> –setup $setupTagi … <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique setup tag
$ctrlTag	tag of previously defined control object (optional)
$setupTag	tags of previously defined setup objects
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental setup
# ------------------------
expSetup  OneActuator  1  2
expSetup  TwoActuators  2  120  120  60

expSetup  Aggregator  3  -control 1  -setup 1 2
```

The above Aggregator experimental setup command combines the previously defined OneActuator setup and TwoActuators setup. In the “-setup” field, the OneActuator setup is listed first and the TwoActuators setup is listed second.

More detailed information about the OneActuator and TwoActuators setups is provided later in this section.

FourActuators Experimental Setup

This command is used to construct a FourActuators experimental setup object. This experimental setup consists of four actuators. The actuators control the two translational and two rotational degrees of freedom of a specimen in 3D. The axial and the torsional DOFs are ignored.

expSetup FourActuators $tag <–control $ctrlTag> $L1 $L2 $L3 $L4 $a1 $a2 $a3 $a4 $h $h1 $h2 $arlN $arlS $LrodN $LrodS $Hbeam <–nlGeom> <–phiLocX $phi> <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique setup tag
$ctrlTag	tag of previously defined control object (optional)
$L1	length of actuator 1
$L2	length of actuator 2
$L3	length of actuator 3
$L4	length of actuator 4
$a1	length of rigid link 1
$a2	length of rigid link 2
$a3	length of rigid link 3
$a4	length of rigid link 4
$h	height of rigid link between actuators
$h1	height of rigid link between lower actuator and pin connection
$h2	height of rigid link between lower actuator and lower beam flange
$arlN	length of rigid link from top pin to lower beam flange at NORTH rod
$arlS	length of rigid link from top pin to lower beam flange at SOUTH rod
$LrodN	pin to pin length of the north side rod
$LrodS	pin to pin length of the south side rod
$Hbeam	height of spreader beam
-nlGeom	nonlinear geometry (optional, default = false)
$phi	angle of actuator 1 with respect to the reaction wall [degree] (optional, default = 0.0)
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental control
# ---------------------------
expControl  xPCtarget  1  1  “192.168.2.20”  22222  HybridControllerD3D3_1Act
"D:/PredictorCorrector/RTActualTestModels/cmAPI-xPCTarget-STS"

# Define experimental setup
# -------------------------
expSetup  FourActuators  1  -control 1  60  60  60  60  72  72  72  72  48  24  24  36  36
12  12  60
```

The above FourActuators experimental setup command uses the previously defined xPCtarget experimental control object. The lengths of Actuators 1, 2, 3, and 4 are 60. The lengths of rigid links 1, 2, 3, and 4 are 72. The length of the rigid link between actuators is 48. The heights of rigid links between the lower actuator and the pin connection and between the lower actuator and the lower beam flange are 24. The lengths of the rigid links between the top pin and the lower beam flange at the NORTH and SOUTH rods are 36. The pin-to-pin lengths of the north side and south side rods are 12. The height of the spreader beam is 60.

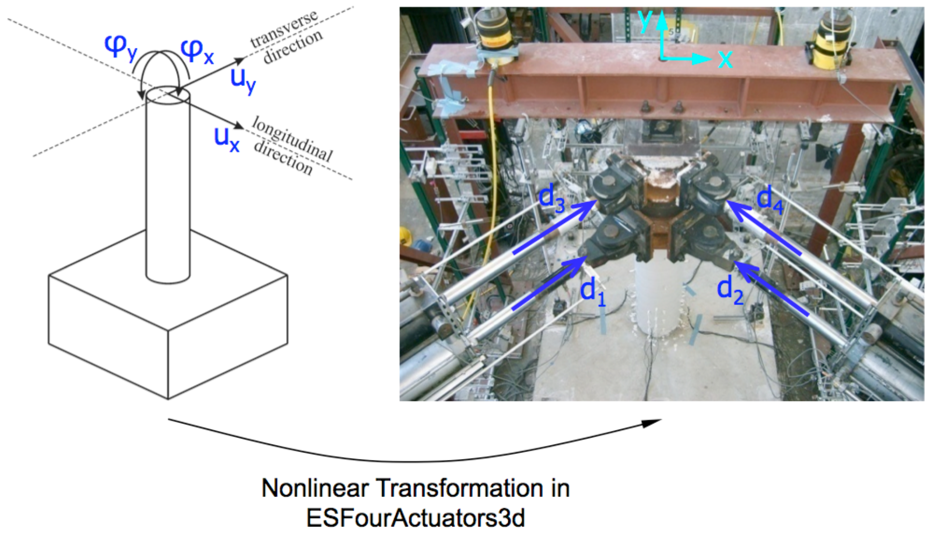


Figure 16: Displacement Transformation in FourActuators Experimental Setup

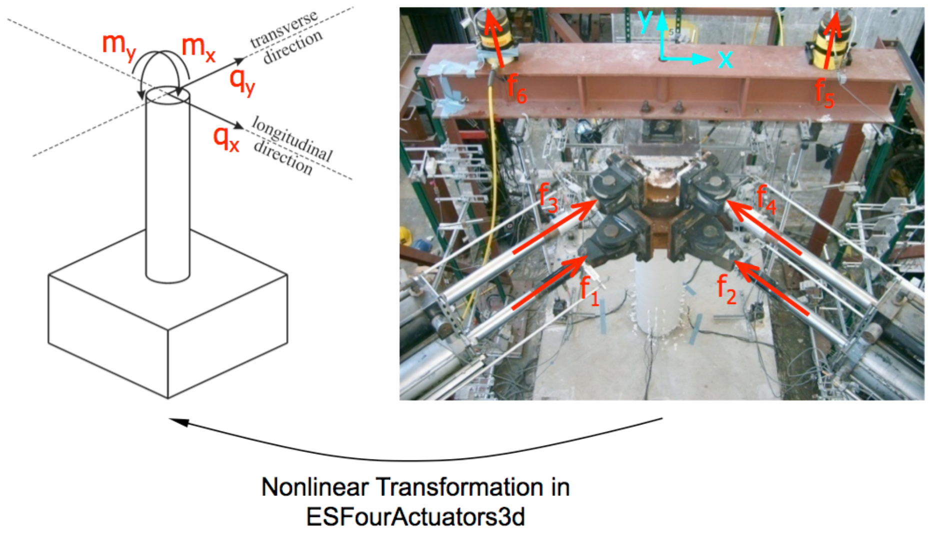


Figure 17: Force Transformation in FourActuators Experimental Setup

InvertedVBrace Experimental Setup

This command is used to construct an InvertedVBrace experimental setup object. This experimental setup consists of three actuators, which control the specimen deformation and two load cells that measure the six support reactions or resisting forces.

expSetup InvertedVBrace $tag <–control $ctrlTag> $La1 $La2 $La3 $L1 $L2 <–nlGeom> <–posAct1 pos> <–phiLocX $phi> <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique setup tag
$ctrlTag	tag of previously defined control object (optional)
$La1	length of actuator 1
$La2	length of actuator 2
$La3	length of actuator 3
$L1	length of rigid link 1
$L2	length of rigid link 2
-nlGeom	nonlinear geometry (optional, default = false)
pos	position of actuator 1, left or right (l or r) (optional, default = left)
$phi	angle from rigid loading beam to local x-axis [degree] (optional, default = 0.0)
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental control
# ---------------------------
# expControl  SimUniaxialMaterials $tag $matTags
expControl  SimUniaxialMaterials  1  1 2 3

# Define experimental setup
# -------------------------
expSetup  InvertedVBrace  1  -control 1  60  72  72  60  60  -posAct1 left
```

The above InvertedVBrace experimental setup command uses the previously defined SimUniaxialMaterials experimental control object. The lengths of Actuators 1, 2, and 3 are 60, 72, and 72, respectively. The total length of the rigid loading beam is 120 (L_1 = 60 and L_2 = 60). Actuator 1 is located on the left side of the beam. The local 1-axis is rotated 0 degree (counterclockwise) from the rigid loading beam.

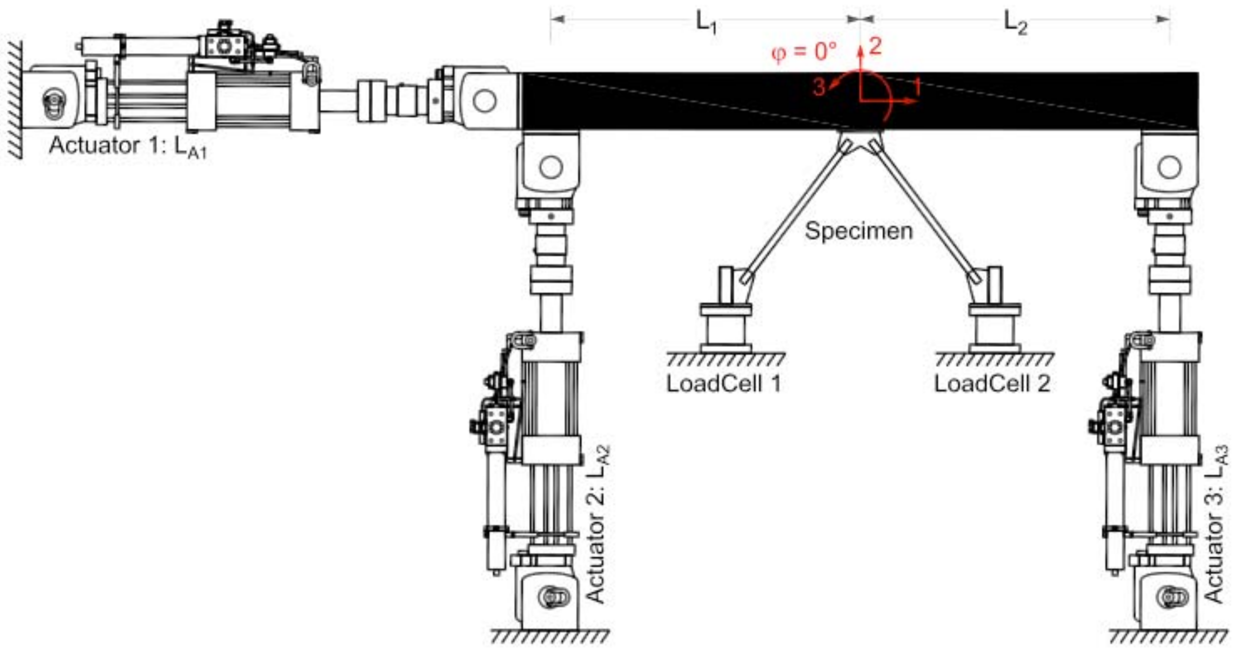


Figure 18: InvertedVBrace Experimental Setup

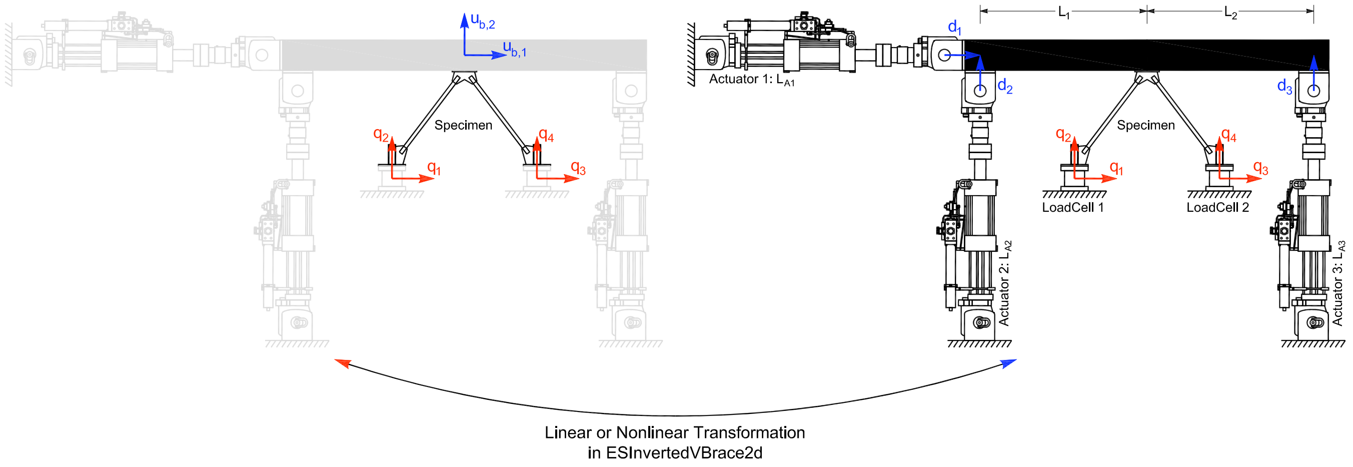


Figure 19: Transformation in InvertedVBrace Experimental Setup

InvertedVBraceJntOff Experimental Setup

This command is used to construct an InvertedVBraceJntOff experimental setup object. This experimental setup consists of three actuators, which control the specimen deformation and two load cells that measure the six support reactions or resisting forces. It accounts for the rigid joint offsets between the actuators.

expSetup InvertedVBraceJntOff $tag <–control $ctrlTag> $La1 $La2 $La3 $L1 $L2 $L3 $L4 $L5 $L6 <–nlGeom> <–posAct1 $pos> <–phiLocX $phi> <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique setup tag
$ctrlTag	tag of previously defined control object (optional)
$La1	length of actuator 1
$La2	length of actuator 2
$La3	length of actuator 3
$L1	length of rigid link 1
$L2	length of rigid link 2
$L3	length of rigid link 3
$L4	length of rigid link 4
$L5	length of rigid link 5
$L6	length of rigid link 6
-nlGeom	nonlinear geometry (optional, default = false)
$pos	position of actuator 1, left or right (l or r) (optional, default = left)
$phi	angle from rigid loading beam to local x-axis [degree] (optional, default = 0.0)
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental control
# ---------------------------
# expControl  SimUniaxialMaterials $tag $matTags
expControl  SimUniaxialMaterials  1  1 2 3

# Define experimental setup
# -------------------------
expSetup  InvertedVBraceJntOff  1  -control  1  60  72  72  24  60  60  24  12  12
-posAct1 left
```

The above InvertedVBraceJntOff experimental setup command uses the previously defined SimUniaxialMaterials experimental control object. The lengths of Actuators 1, 2, and 3 are 60, 72, and 72, respectively. The total length of the rigid loading beam is 120 (L_2 = 60 and L_3 = 60). The lengths of the offsets L_1, L_4, L_5, and L_6 are 24, 24, 12, and 12, respectively. Actuator 1 is located on the left side of the beam. The local 1-axis is rotated 0 degree (counterclockwise) from the rigid loading beam.

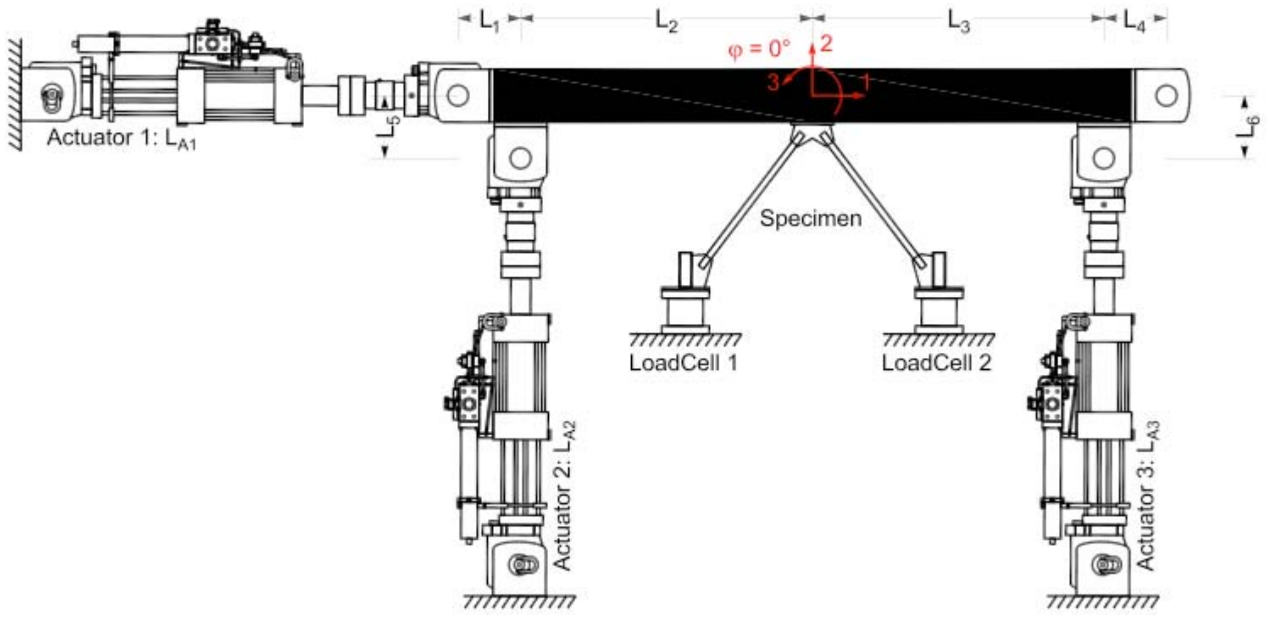


Figure 20: InvertedVBraceJntOff Experimental Setup

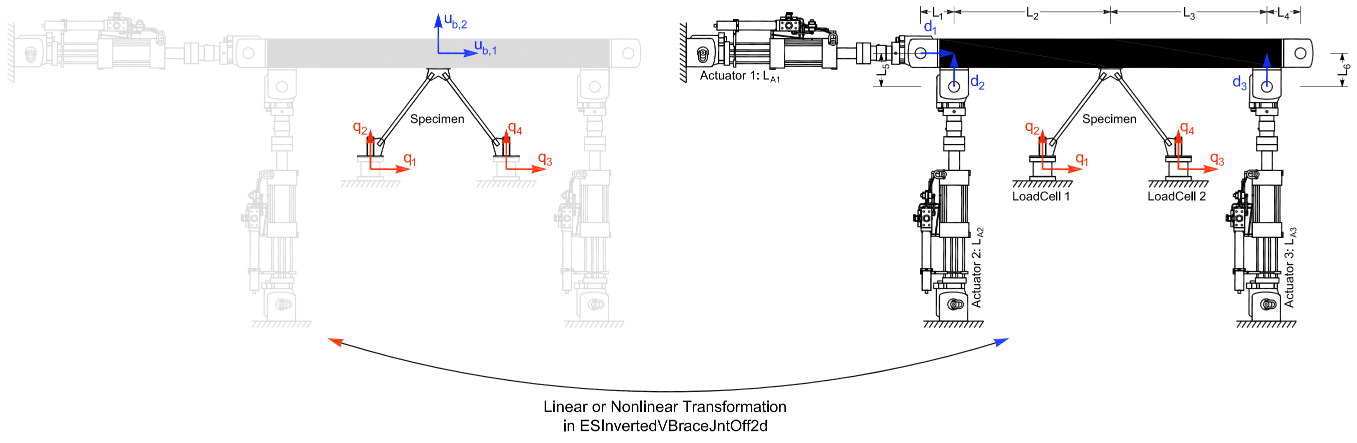


Figure 21: Transformation in InvertedVBraceJntOff Experimental Setup

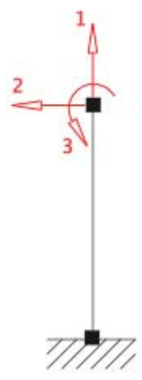
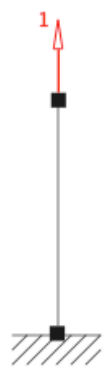
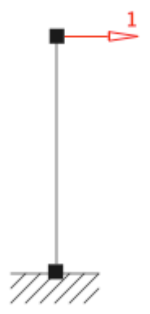
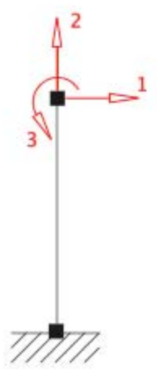
NoTransformation Experimental Setup

This command is used to construct a NoTransformation experimental setup object. This experimental setup consists of up to six actuators, which are set to control any of the basic degrees of freedom of a specimen.

expSetup NoTransformation $tag <–control $ctrlTag> –dir $dirs … –sizeTrialOut $sizeTrial $sizeOut <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique element tag
$ctrlTag	tag of previously defined control object (optional)
$dirs	directions (1-6)
$sizeTrial	size of the trial vector received from the element
$sizeOut	size of the output vector returned to the element
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

2D EXAMPLES:

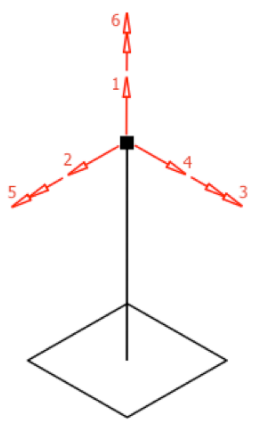
 Element Degrees of Freedom	 Experimental Setup 1	 Experimental Setup 2	 Experimental Setup 3

```
# Define experimental control
# ---------------------------
expControl  SimUniaxialMaterials  1  1
expControl  SimUniaxialMaterials  2  1 2 3

# Define experimental setup
# -------------------------
# Experimental Setup 1
expSetup  NoTransformation  1  -control 1  -dir 1  -sizeTrialOut 3 3
```

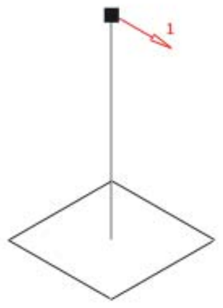
```
# Experimental Setup 2
expSetup  NoTransformation  2  -control 1  -dir 2  -sizeTrialOut 3 3  -trialDispFact -1
-outDispFact -1  -outForceFact  -1

# Experimental Setup 3
expSetup  NoTransformation  3  -control 2  -dir 2 1 3  -sizeTrialOut 3 3  -trialDispFact -1
1 1  -outDispFact -1 1 1  -outForceFact -1 1 1
```

Three examples are provided to show how to define the -dir input. All the examples use the previously defined SimUniaxialMaterials experimental control objects. In the example Experimental Setup 1, the control system degree of freedom 1 points in the same direction as the element degree of freedom 1. In the example Experimental Setup 2, the control system degree of freedom 1 points in the negative direction of element degree of freedom 2. Therefore, the trial displacement, output displacement, and output force are factored by -1. In example 3, the control system degrees of freedom 1, 2, and 3 point in the direction of the element degrees of freedom 2, 1, and 3, respectively. The response quantities for the degree of freedom 1 are factored by -1 since it points in the negative direction of the element degree of freedom 2.

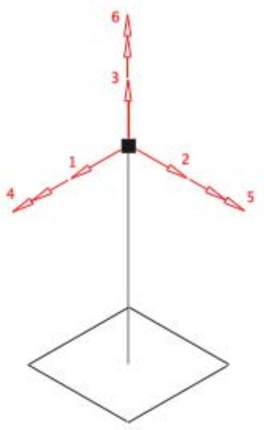
3D EXAMPLES:



Element Degrees of Freedom



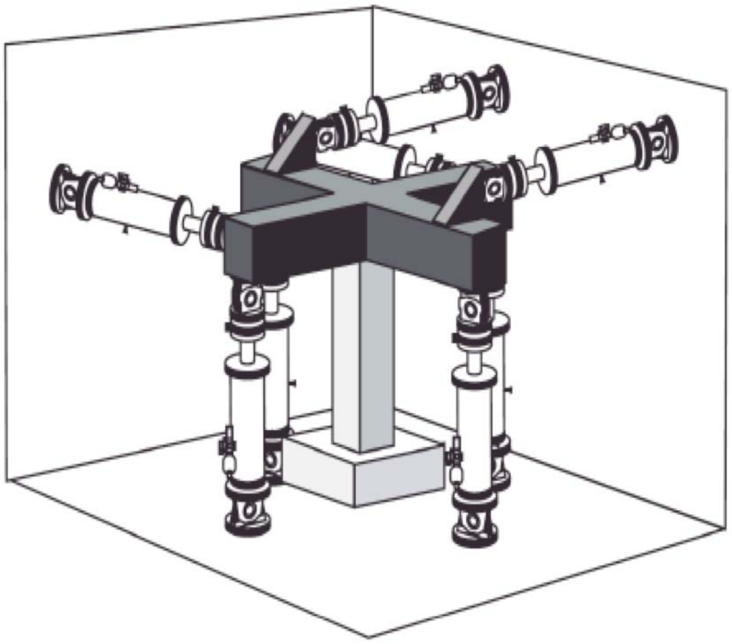
Experimental Setup 1



Experimental Setup 1

```
# Define experimental control
# --------------------------
# expControl  SimUniaxialMaterials $tag $matTags
expControl  SimUniaxialMaterials  1  1
expControl  SimUniaxialMaterials  2  1 2 3 4 5 6

# Define experimental setup
# -------------------------
# Experimental Setup 1
expSetup  NoTransformation  1  -control 1  -dir 4  -sizeTrialOut 6 6

# Experimental Setup 2
expSetup  NoTransformation  2  -control 2  -dir 2 4 1 5 3 6  -sizeTrialOut 6 6
```

The -dir inputs for 3D examples work in the same manner as the 2D examples. Refer to the 2D examples for more information.



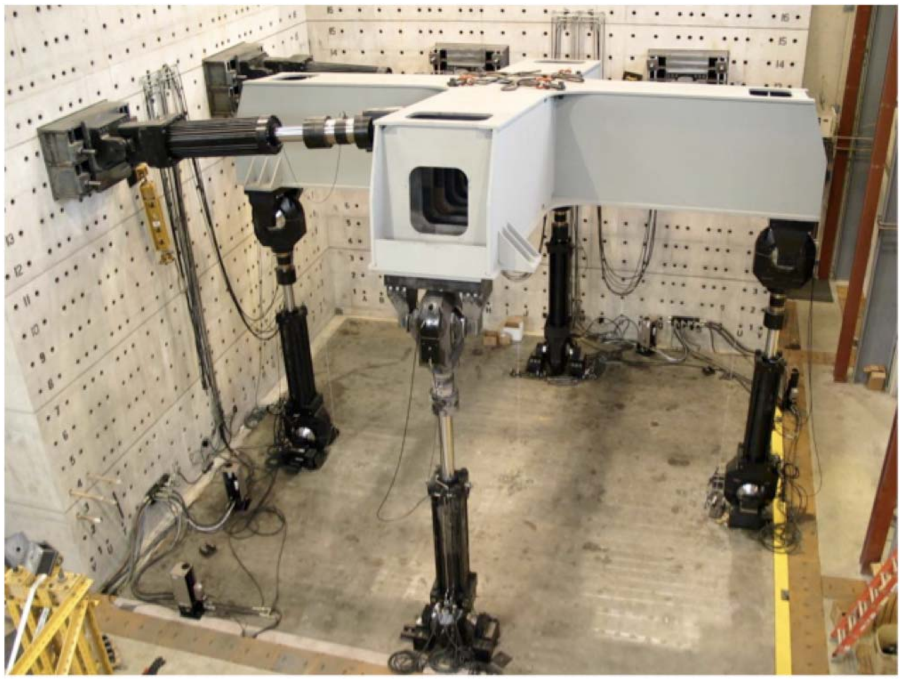


Figure 22: NoTransformation Experimental Setup

OneActuator Experimental Setup

This command is used to construct a OneActuator experimental setup object. This experimental setup consists of only one actuator, which is set in direction to the specimen.

expSetup OneActuator $tag <–control $ctrlTag> $dir –sizeTrialOut $sizeTrial $sizeOut <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique element tag
$ctrlTag	tag of previously defined control object (optional)
$dir	direction of the imposed quantity in the element basic reference coordinate system (1-6)
$sizeTrial	size of the trial vector received from the element
$sizeOut	size of the output vector returned to the element
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental control
# --------------------------
expControl SCRAMNet 1 381020 8

# Define experimental setup
# ------------------------
expSetup  OneActuator  1  –control 1  2  -sizeTrialOut  3  3  -trialDispFact 0.5  -outDispFact
2.0  -outForceFact 2.0
```

The above OneActuator experimental setup uses the previously defined SCRAMNet experimental control object. The imposed quantity is in the 2-direction as shown in Figure 14. The size of the trial vector and the output vector are set to 3. A factor of 0.5 is applied to the trial displacement. A factor of 2.0 is applied to the output displacement and force.

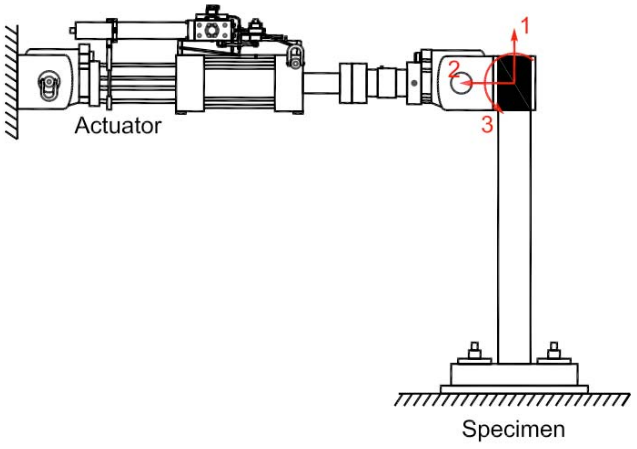


Figure 23: OneActuator Experimental Setup

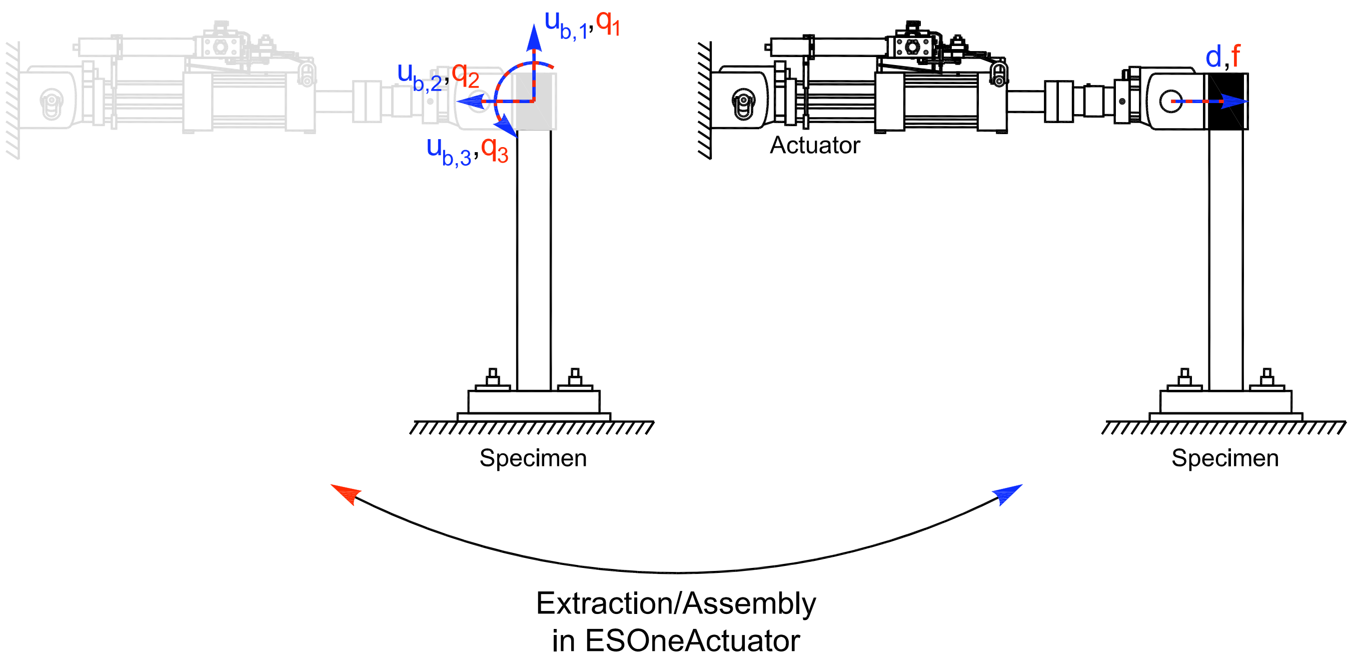


Figure 24: Transformation in OneActuator Experimental Setup

ThreeActuators Experimental Setup

This command is used to construct a ThreeActuators experimental setup object. This experimental setup consists of three actuators. The actuators control the two translational and the rotational degrees of freedom of a specimen.

expSetup ThreeActuators $tag <–control $ctrlTag> $La1 $La2 $La3 $L1 $L2 <–nlGeom> <–posAct1 $pos> <–phiLocX $phi> <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique setup tag
$ctrlTag	tag of previously defined control object (optional)
$La1	length of actuator 1
$La2	length of actuator 2
$La3	length of actuator 3
$L1	length of rigid link 1
$L2	length of rigid link 2
-nlGeom	nonlinear geometry (optional, default = false)
$pos	position of actuator 1, left or right (l or r) (optional, default = left)
$phi	angle from rigid loading beam to local x-axis [degree] (optional, default = 0.0)
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental control
# ----------------------------
# expControl SimUniaxialMaterials $tag $matTags
expControl  SimUniaxialMaterials  1  1 2 3

# Define experimental setup
# --------------------------
expSetup  ThreeActuators  1  -control 1  60  72  72  60  60  -philLocX 90
```

The above ThreeActuators experimental setup command uses the previously defined SimUniaxialMaterials experimental control object. The lengths of Actuators 1, 2, and 3 are 60, 72, and 72, respectively. The total length of the rigid loading beam is 120 (L_1 = 60 and L_2 = 60). Actuator 1 is located on the left side of the beam. The local 1-axis is rotated 90 degrees (counterclockwise) from the rigid loading beam.

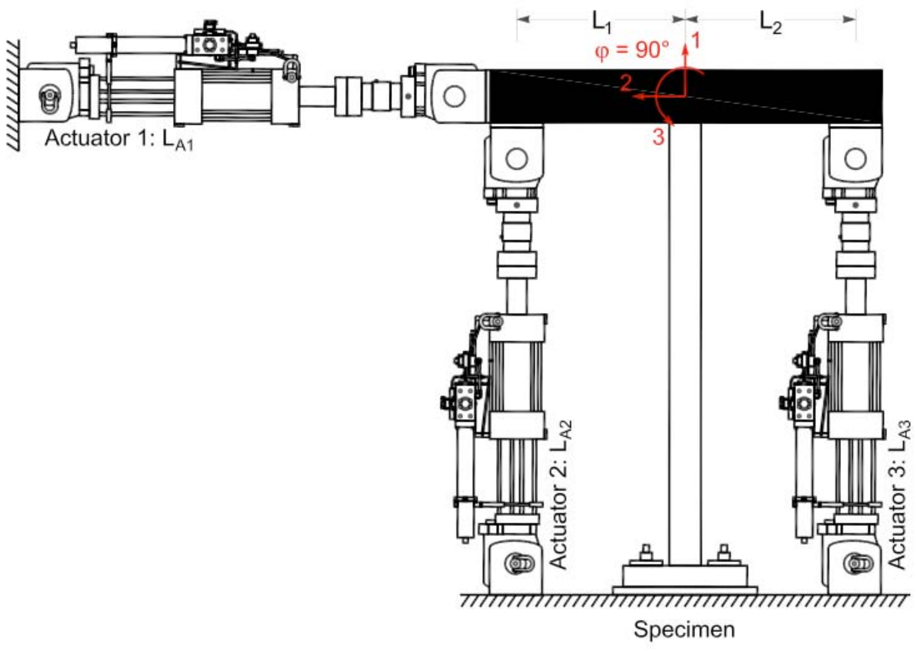
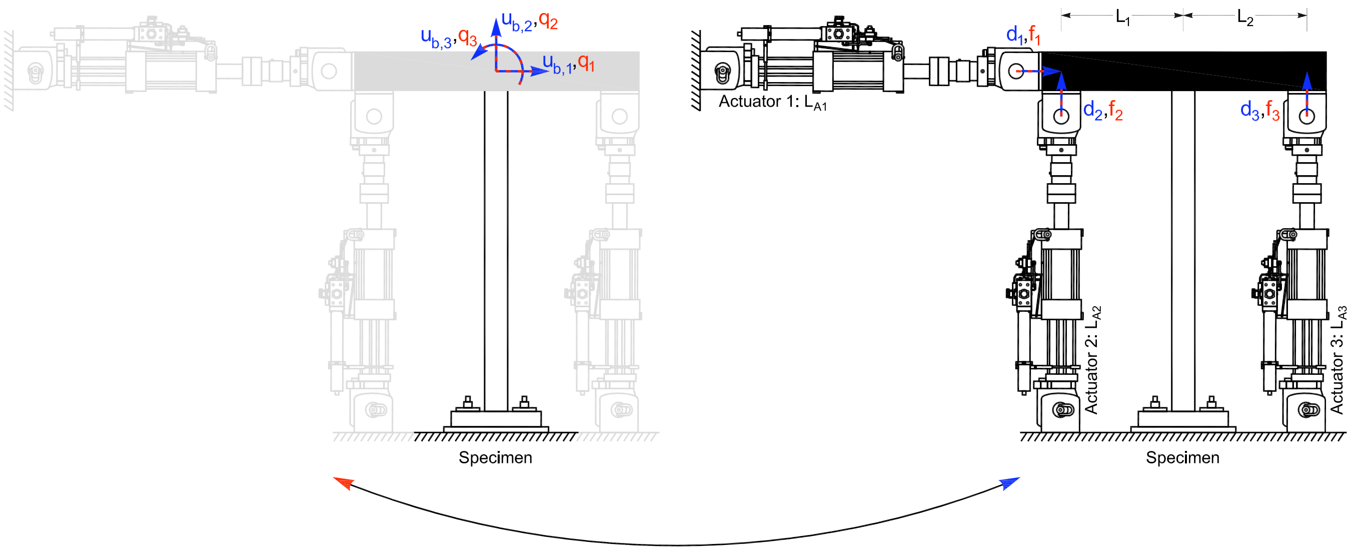


Figure 25: ThreeActuators Experimental Setup



Linear or Nonlinear Transformation
in ESThreeActuators2d

Figure 26: Transformation in ThreeActuators Experimental Setup

ThreeActuatorsJntOff Experimental Setup

This command is used to construct a ThreeActuatorsJntOff experimental setup object. This experimental setup consists of three actuators, which control the two translational and the rotational degrees of freedom of a specimen. It accounts for the rigid joint offsets between the actuators.

expSetup ThreeActuatorsJntOff $tag <–control $ctrlTag> $La1 $La2 $La3 $L1 $L2 $L3 $L4 $L5 $L6 <–nlGeom> <–posAct1 $pos> <–phiLocX $phi> <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique setup tag
$ctrlTag	tag of previously defined control object (optional)
$La1	length of actuator 1
$La2	length of actuator 2
$La3	length of actuator 3
$L1	length of rigid link 1
$L2	length of rigid link 2
$L3	length of rigid link 3
$L4	length of rigid link 4
$L5	length of rigid link 5
$L6	length of rigid link 6
-nlGeom	nonlinear geometry (optional, default = false)
$pos	position of actuator 1, left or right (l or r) (optional, default = left)
$phi	angle from rigid loading beam to local x-axis [degree] (optional, default = 0.0)
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental control
# --------------------------
# expControl  SimUniaxialMaterials $tag $matTags
expControl  SimUniaxialMaterials  1  1 2 3

# Define experimental setup
# -------------------------
expSetup  ThreeActuatorsJntOff  1  -control 1  60  72  72  24  60  60  24  12  12
-philLocX 90
```

The above ThreeActuatorJntOff experimental setup command uses the previously defined SimUniaxialMaterials experimental control object. The lengths of Actuators 1, 2, and 3 are 60, 72, and 72, respectively. The total length of the rigid loading beam is 120 (L_2 = 60 and L_3 = 60). The lengths of the offsets L_1, L_4, L_5, and L_6 are 24, 24, 12, and 12, respectively. Actuator 1 is located on the left side of the beam. The local 1-axis is rotated 90 degrees (counterclockwise) from the rigid loading beam.

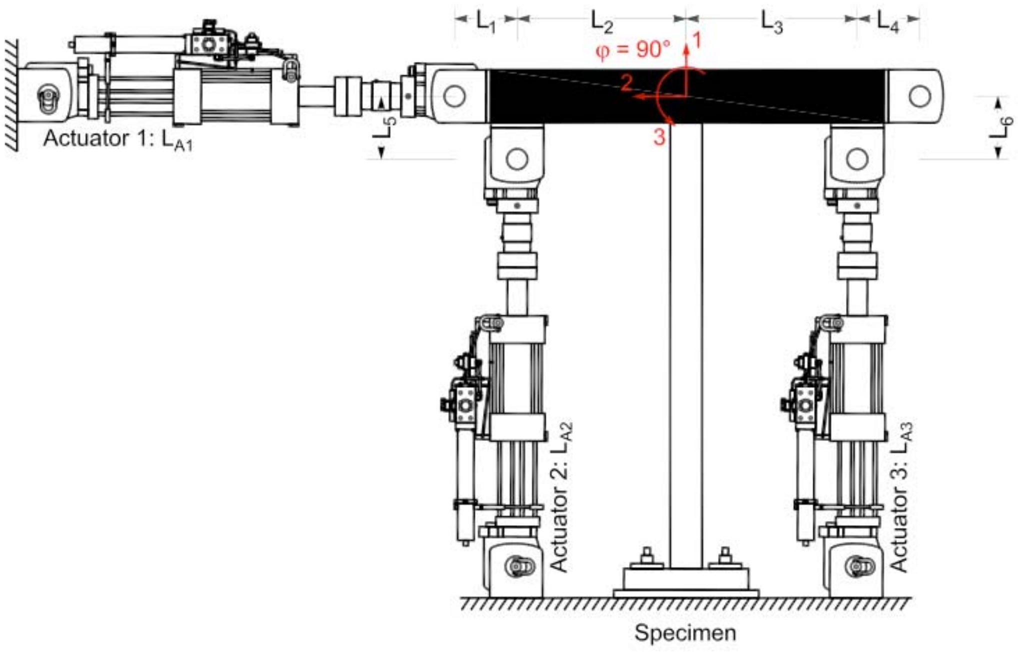


Figure 27: ThreeActuatorsJntOff Experimental Setup

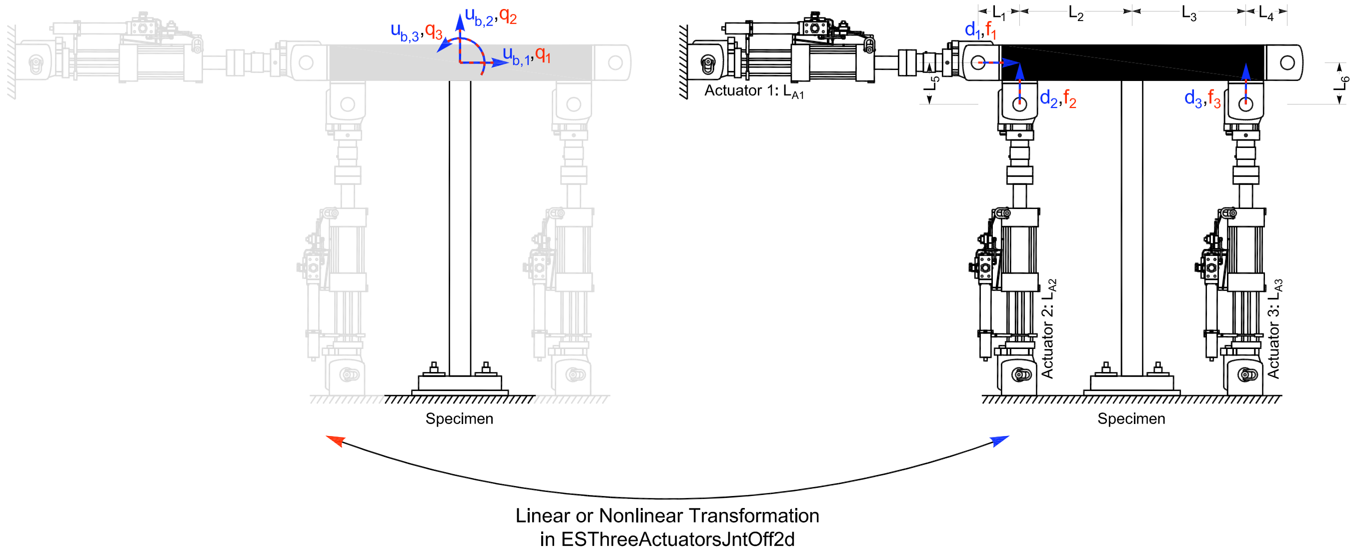


Figure 28: Transformation in ThreeActuatorsJntOff Experimental Setup

TwoActuators Experimental Setup

This command is used to construct a TwoActuators experimental setup object. This experimental setup consists of two actuators, which control the translational and the rotational degrees of freedom of a specimen.

expSetup TwoActuators $tag <–control $ctrlTag> $La1 $La2 $L <–nlGeom> <–posAct $pos> <–phiLocX $phi> <–trialDispFact $f> <–trialVelFact $f> <–trialAccelFact $f> <–trialForceFact $f> <–trialTimeFact $f> <–outDispFact $f> <–outVelFact $f> <–outAccelFact $f> <–outForceFact $f> <–outTimeFact $f>

$tag	unique setup tag
$ctrlTag	tag of previously defined control object (optional)
$La1	length of actuator 1
$La2	length of actuator 2
L	length of rigid link
-nlGeom	nonlinear geometry (optional, default = false)
$pos	position of actuators, left or right (l or r) (optional, default = left)
$phi	angle from horizontal line formed by actuator 1 to local x-axis [degree] (optional, default = 0.0)
$f	factors applied to trial (<-ctrl….Fact $f>) and acquired (<-out….Fact $f>) data before transformation (optional, default = 1.0)

EXAMPLE:

```
# Define experimental control
# ---------------------------
# expControl SimUniaxialMaterials $tag $matTags
expControl  SimUniaxialMaterials  1  1 2

# Define experimental setup
# -------------------------
expSetup  TwoActuators  1  -control 1  72  72  60  -philLocX 90
```

The above TwoActuators experimental setup command uses the previously defined SimUniaxialMaterials experimental control object. The lengths of Actuators 1 and 2 are 72 and 72, respectively. The total length of the rigid loading beam is 60. Actuators are located on the left side of the beam. The local 1-axis is rotated 90 degrees (counterclockwise) from Actuator 1.

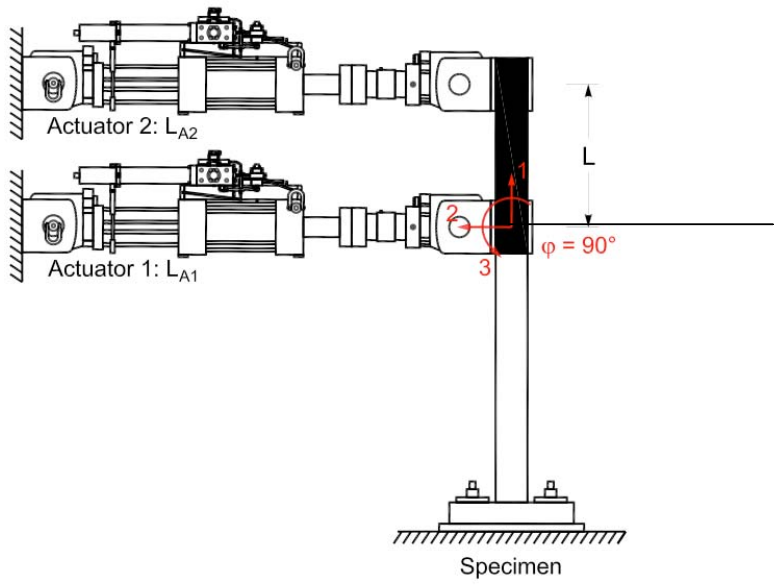


Figure 29: TwoActuators Experimental Setup

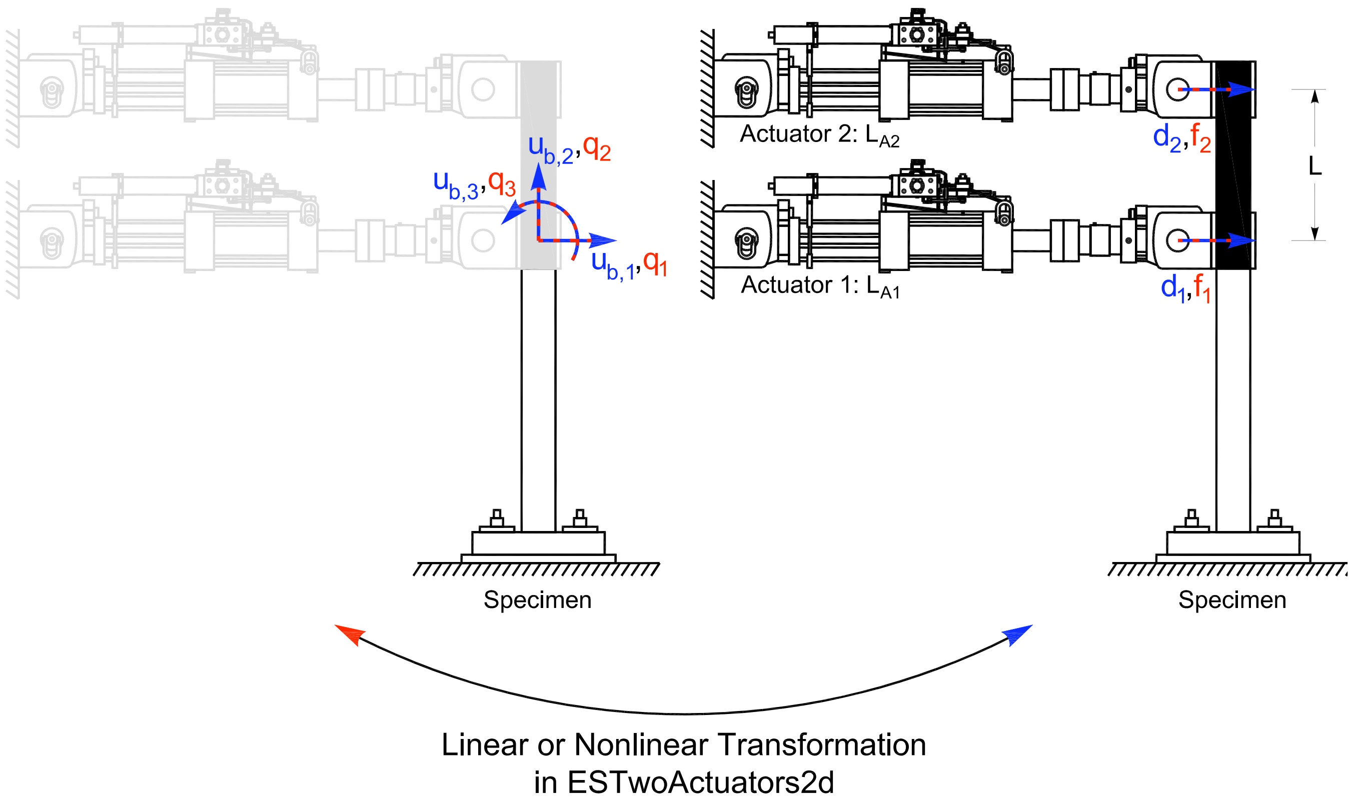


Figure 30: Transformation in TwoActuators Experimental Setup

CHAPTER 5

expSignalFilter Commands

These commands are used to construct the expSignalFilter objects. These objects are used to filter or modify the signals that are both being sent and received from the control system.

In This Chapter

ErrorSimRandomGauss Experimental Signal Filter

This command is used to construct an Error Simulation Random Gauss experimental signal filter object. The Gaussian white noise is generated using the Box-Muller method. The uniform deviates are generated by the special function based on NR::ran3 in Numerical Recipes in C++, 2nd Ed.

expSignalFilter ErrorSimRandomGauss $tag $avg $std

$tag	unique experimental signal filter tag
$avg	average of error
$std	standard deviation of error

The signal at the output port of the filter is calculated as follows:

$$signal = signal + error(avg, std)$$

EXAMPLE:

```
# Define experimental signal filter
# --------------------------------
expSignalFilter  ErrorSimRandomGauss  1  0.0  0.015
```

This example uses an average of 0.0 and a standard deviation of 0.015.

Reference:

Press, William H., et al., “Numerical Recipes in C++: The Art of Scientific Computing”, 2nd ed., Cambridge University Press, 2002.

ErrorSimUndershoot Experimental Signal Filter

This command is used to construct an Error Simulation Undershoot experimental signal filter object.

expSignalFilter ErrorSimUndershoot $tag $error

$tag	unique experimental signal filter tag
$error	undershoot error

The signal at the output port of the filter is calculated as follows:

```
if $(signal_{i+1} > signal_i)$
    $signal_{i+1} = signal_{i+1} - error$
elseif $(signal_{i+1} < signal_i)$
    $signal_{i+1} = signal_{i+1} + error$
else
    $signal_{i+1} = signal_{i+1}$
```

EXAMPLE:

```
# Define experimental signal filter
# --------------------------------
expSignalFilter  ErrorSimUndershoot  1  0.167
```

This example uses an error of 0.167.

CHAPTER 6

expSite Commands

These commands are used to construct the expSite objects. The expSite objects provide communication methods for distributed testing. They utilize communication channels with TCP/IP, NHCP, or UDP communication protocols. In addition they optionally use OpenSSL for secure communication in the case of distributed testing.

In This Chapter

Actor Experimental Site

This command is used to construct an ActorSite experimental site object. An actor experimental site communicates with a remote experimental site and runs on the server program. This object can be defined with either an already defined experimental setup or an experimental control.

To use with already defined experimental setup (in other words, the setup is on the server side):

expSite ActorSite $tag –setup $setupTag $ipPort <–ssl>

To use with already defined experimental control (in other words, the setup is on the client side):

expSite ActorSite $tag –control $ctrlTag $ipPort <–ssl>

$tag	unique experimental site tag
$setupTag	tag of the previously defined setup object if the setup is on the server side
$ipPort	IP port number of the ActorSite
-ssl	secure transactions using OpenSSL (optional)
$ctrlTag	tag of the previously defined control object if the setup is on the client side

EXAMPLE:

```
# Define experimental setup
# -------------------------
# expSetup OneActuator $tag <-control $ctrlTag> $dir <-trialDispFact $f> ...
expSetup  OneActuator  1  -control 2  1

# Define experimental site
# ------------------------
expSite  ActorSite  1  -setup 1  8090
```

This example uses a previously defined OneActuator experimental setup and an IP port of 8090. The experimental setup is defined on the server side.

Local Experimental Site

This command is used to construct a LocalSite experimental site object. It is used to run a local test where no client-server communication is necessary.

expSite LocalSite $tag $setupTag

$tag	unique experimental site tag
$setupTag	tag of the previously defined setup object

EXAMPLE:

```
# Define experimental setup
# -------------------------
# expSetup OneActuator $tag <-control $ctrlTag> $dir <-trialDispFact $f> ...
expSetup  OneActuator  2  -control 2  1

# Define experimental site
# ------------------------
expSite  LocalSite  2  2
```

This example uses a previously defined OneActuator experimental setup to run a local test.

Shadow (or Remote) Experimental Site

This command is used to construct a ShadowSite (or RemoteSite) experimental site object. A shadow (or remote) experimental site communicates with an actor experimental site and runs on the client program.

For Shadow Site:

expSite ShadowSite $tag <–setup $setupTag> ipAddr $ipPort <–dataSize $size>

For Remote Site (this is obsolete and will be removed in future versions):

expSite RemoteSite $tag <–setup $setupTag> ipAddr $ipPort <–dataSize $size>

$tag	unique experimental site tag
$setupTag	tag of the previously defined setup object (optional, needs to be provided if setup is on client side)
ipAddr	IP address of the corresponding ActorSite
$ipPort	IP port number of the corresponding ActorSite
-ssl	secure transactions using OpenSSL (optional)
$size	data size being sent (optional)

EXAMPLE:

```
# Define experimental site
# ------------------------
expSite  ShadowSite  1  "169.229.203.152"  8090
```

This example uses an IP address of 169.229.203.152 and port number of 8090.

CHAPTER 7

expControlPoint Command

This command is used to construct the expControlPoint object. The expControlPoint command is used when a test is conducted using control points. For example, control points are required when using the LabVIEW experimental control command. Since the MTS MiniMost, which is controlled by LabVIEW, cannot set safety limits, the expControlPoint command is used to set such limits.

expControlPoint $tag $nodeTag dir resp <–fact $f> <–lim $l $u> …

$tag	unique control point tag
$nodeTag	unique node tag
dir	direction of response quantity; input parameters are: ▪ ux = X-axis direction ▪ uy = Y-axis direction ▪ uz = Z-axis direction ▪ rx = rotation in the X-axis direction ▪ ry = rotation in the Y-axis direction ▪ rz = rotation in the Z-axis direction
resp	response quantity; input parameters are: ▪ disp = displacement ▪ vel = velocity ▪ accel = acceleration ▪ force = force ▪ time = time
$f	factor applied to response quantity (optional)
$l	lower limit (optional)
$u	upper limit (optional)

A control point represents a logical container of one or more directions (DOF) for sending command signals to a controller or acquiring feedback signals from a data acquisition system. Control-points represent logical groupings of output control channels or input data acquisition channels. To define the control and data acquisition axes for a test configuration, the dir (direction) property of the expControlPoint object is used. Each direction (DOF) is associated with a control channel and response quantity. The set of control-points and the set of directions within these control-points define the control axes and their expected order for the expControl object, which is using the control-points.

Similarly, the set of control-points and the set of directions within these control-points define the feedback axes and their expected order, as they are received from the expControl object. Additionally, the expControlPoint command allows for the definition of scaling factors as well as lower and upper limits for each response quantity. The optional scaling factors can be used to convert between different units and/or to account for similitude laws. Finally, the optional limits can be invoked if no safety limits can be set in the control system.

EXAMPLE:

```
# Define geometry for model
# ------------------------
set mass3 0.04
set mass4 0.02
# node $tag $xCrd $yCrd $mass
node  1     0.0   0.00
node  2   100.0   0.00
node  3     0.0  54.00  -mass $mass3 $mass3
node  4   100.0  54.00  -mass $mass4 $mass4

# Define experimental control points
# ----------------------------------
expControlPoint  1  1  ux disp -fact 0.003 -lim -0.01 0.01
expControlPoint  2  1  ux disp -fact 0.003  ux force -fact [expr 18.0/7.0]

# Define experimental control
# --------------------------
expControl  LabVIEW  1  “130.126.242.175”  44000  –trialCP 1  –outCP 2
```

This example uses node 1 as an experimental control point. The first control point factors the displacement in the x-direction by 0.003 and sets the lower limit to -0.01 and upper limit to 0.01. The second control point factors both the displacement in the x-direction by 0.002 and the force in the x-direction by [expr 18.0/7.0] = 2.57. The first control point is used for the LabVIEW trial command (displacement) and the second for the LabVIEW output feedback (measured displacement and force).

Following are two control-point setups that represent the same physical system. In Setup 1, a single control point has two directions (DOFs), and each direction will be mapped to either a translational or a rotational control channel. The control system will have to transform the control channels correctly to match the physical test rig (the NoTransformation experimental setup is used). Setup 2 receives the correctly transformed response quantities from the appropriate expSetup object so each control-point with a single direction (DOF) is mapped directly to a simple control channel.

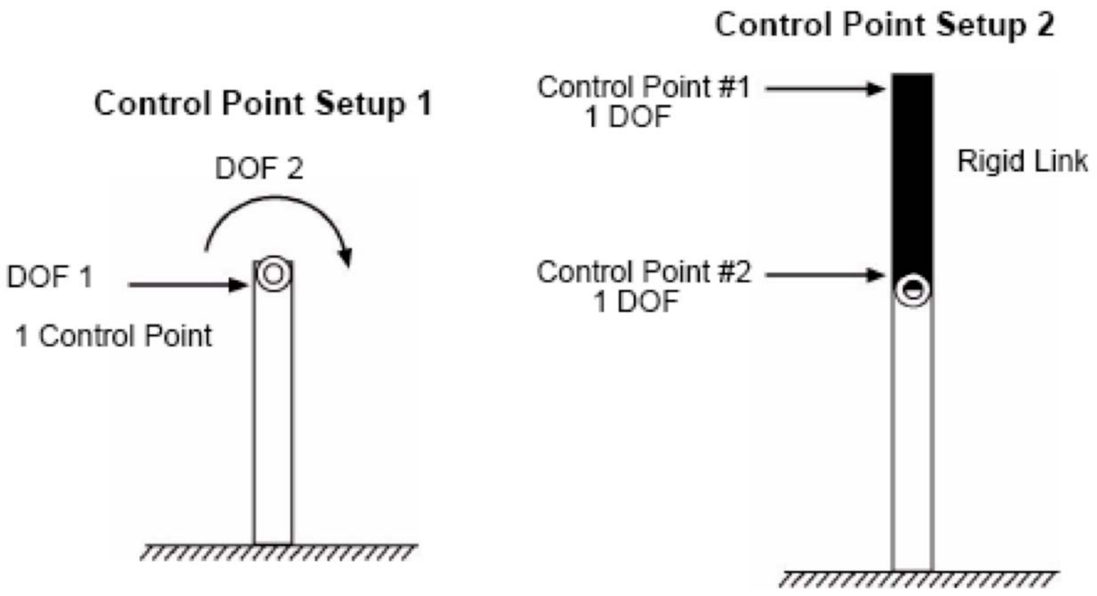


Figure 31: Two fundamentally different control-point setups (courtesy of MTS)

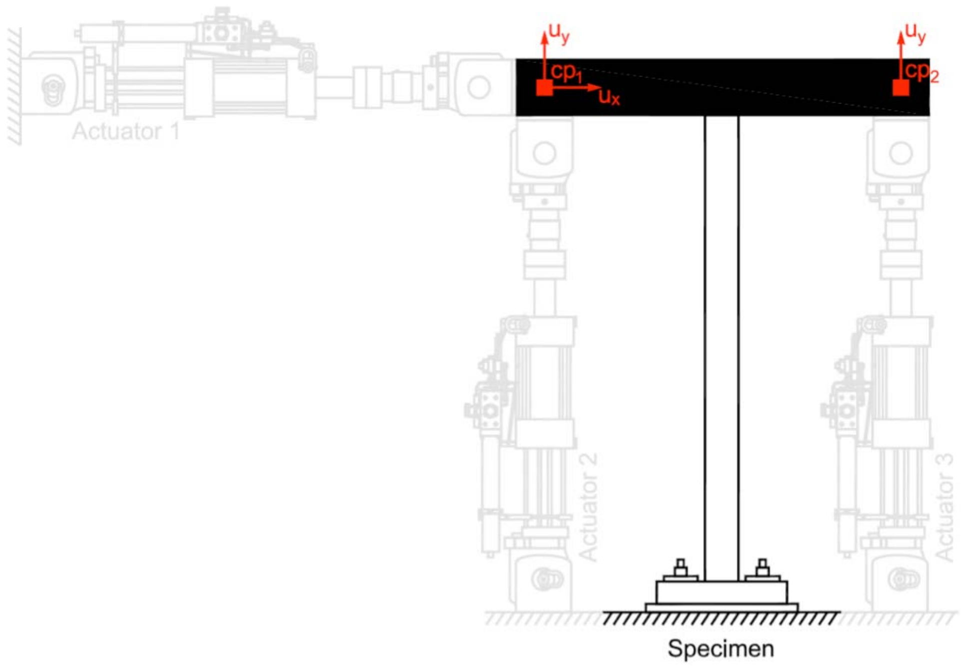


Figure 32: Example with two control-points

C H A P T E R 8

expRecorder Commands

These commands are used to construct the experimental recorder objects. The recorder objects provide means for monitoring and recording response quantities of interest to the experimentalist. Figure 28 illustrates the naming convention for the signals entering and leaving the various OpenFresco modules. The following terminology is used for the data flow: Any data that is flowing from the finite element analysis software towards the laboratory is called 1) “trial data” (if entering an object/class) or 2) “ctrl data” (if leaving an object/class); and any data that is flowing from the laboratory towards the finite element analysis software is called 3) “daq data” (if entering an object/class) or 4) “out data” (if leaving an object/class). The recorder response type strings follow the same naming convention.

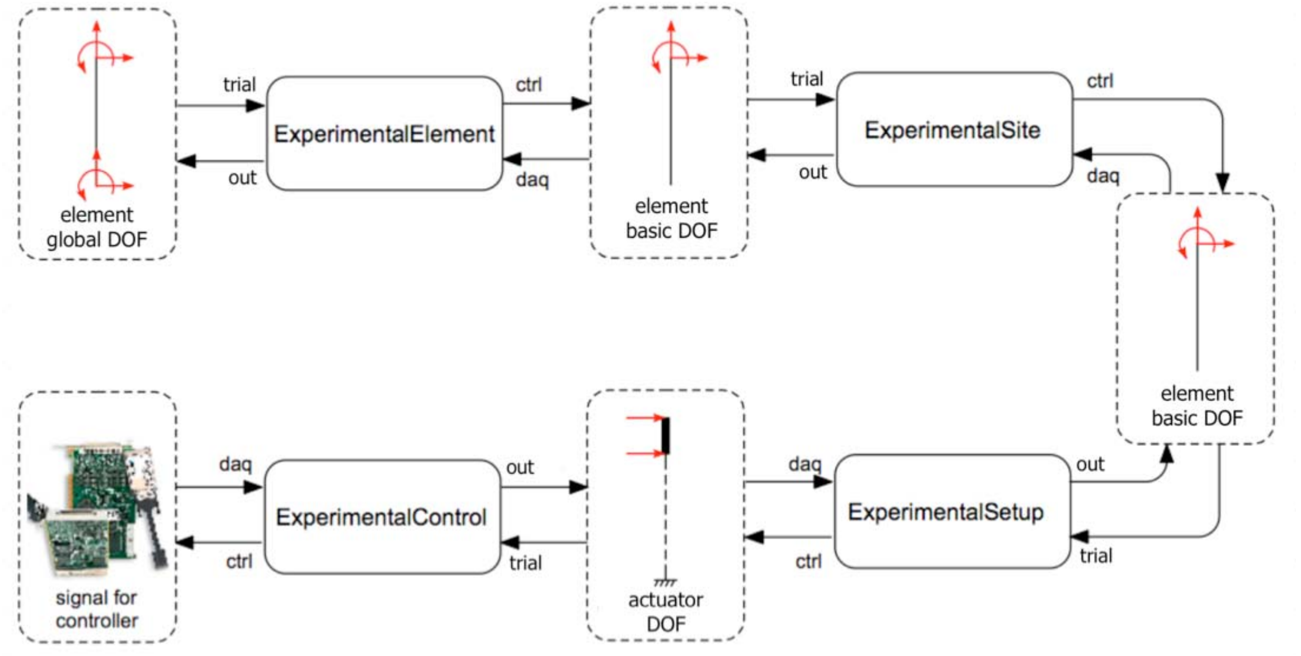


Figure 33: Data flow and transformations for OpenFresco modules

In This Chapter

Control Recorder

This command is used to construct an experimental control recorder object. It is used to record data from the experimental controls such as control (target) and daq (measured) response quantities.

expRecorder Control <–file $fileName> <–csv $fileName> <–xml $fileName> <–binary $fileName> <–database $tableName> <–time> <–dt> <–control $ctrlTag …> <–controlRange $startTag $endTag> <–control all> respType

–file	to record data to a file in ASCII format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–csv	to record data to a file in ASCII comma separated format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–xml	to record data to a file in ASCII format including metadata (optional: use only one of –file, –csv, –xml, –binary, –database)
–binary	to record data to a file in binary format (optional: use only one of –file, –csv, –xml, –binary, –database)
–database	to record data to a database (optional: use only one of –file, –csv, –xml, –binary, –database)
$fileName	file where results are stored. Each line of the file contains the result for a committed state of the experimental test.
$tableName	name of table in database where results are stored. Each line of the table contains the result for a committed state of the experimental test.
–time	to place the time or pseudo-time of the analysis in the first data column (optional)
–dt	to record data at a different time increment than the analysis time step (this should be a multiple of the analysis time step)
$ctrlTag	tag of experimental control to be recorded
$startTag, $endTag	start and end tag of range of experimental controls to be recorded
all	to record all experimental control objects (Note: This option is recommended only for xml file output as OpenFresco might change the order of the experimental control objects)
respType	defines the response type to be recorded. Since this depends on the type of the experimental control the possible arguments are described with the control commands.

EXAMPLE:

```
# Define experimental control
# ----------------------------
# expControl SimUniaxialMaterials $tag $matTags
expControl SimUniaxialMaterials 1 1
expControl SimUniaxialMaterials 2 2
...

# ------------------------------
# Start of recorder generation
# ------------------------------
expRecorder Control -file Control_ctrlDsp.out -time -control 1 2 ctrlDisp
expRecorder Control -file Control_ctrlVel.out -time -control 1 2 ctrlVel
expRecorder Control -file Control_daqDsp.out -time -control 1 2 daqDisp
expRecorder Control -file Control_daqVel.out -time -control 1 2 daqVel
expRecorder Control -file Control_daqFrc.out -time -control 1 2 daqForce
# --------------------------------
# End of recorder generation
# --------------------------------
```

This example creates experimental control recorders that write data to a file in ASCII format without headers. Target (control) displacements and velocities as well as measured (daq) displacements, velocities and forces are recorded for the previously created experimental controls with tags 1 and 2.

Setup Recorder

This command is used to construct an experimental setup recorder object. It is used to record data from the experimental setups such as trial, output, target (control) and measured (daq) response quantities.

expRecorder Setup <–file $fileName> <–csv $fileName> <–xml $fileName> <–binary $fileName> <–database $tableName> <–time> <–dt> <–setup $setupTag …> <–setupRange $startTag $endTag> <–setup all> respType

–file	to record data to a file in ASCII format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–csv	to record data to a file in ASCII comma separated format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–xml	to record data to a file in ASCII format including metadata (optional: use only one of –file, –csv, –xml, –binary, –database)
–binary	to record data to a file in binary format (optional: use only one of –file, –csv, –xml, –binary, –database)
–database	to record data to a database (optional: use only one of –file, –csv, –xml, –binary, –database)
$fileName	file where results are stored. Each line of the file contains the result for a committed state of the experimental test.
$tableName	name of table in database where results are stored. Each line of the table contains the result for a committed state of the experimental test.
–time	to place the time or pseudo-time of the analysis in the first data column (optional)
–dt	to record data at a different time increment than the analysis time step (this should be a multiple of the analysis time step)
$setupTag	tag of experimental setup to be recorded
$startTag, $endTag	start and end tag of range of experimental setups to be recorded
all	to record all experimental setup objects (Note: This option is recommended only for xml file output as OpenFresco might change the order of the experimental setup objects)
respType	defines the response type to be recorded. The possible arguments are described below.

The valid queries to an experimental setup when creating an *ExpSetupRecorder* object are:

• trial displacements:	trialDisp, trialDisplacement, trialDisplacements
• trial velocities:	trialVel, trialVelocity, trialVelocities
• trial accelerations:	trialAccel, trialAcceleration, trialAccelerations
• trial forces:	trialForce, trialForces
• trial time:	trialTime, trialTimes
• output displacements:	outDisp, outDisplacement, outDisplacements
• output velocities:	outVel, outVelocity, outVelocities
• output accelerations:	outAccel, outAcceleration, outAccelerations
• output forces:	outForce, outForces
• output time:	outTime, outTimes
• control displacements:	ctrlDisp, ctrlDisplacement, ctrlDisplacements
• control velocities:	ctrlVel, ctrlVelocity, ctrlVelocities
• control accelerations:	ctrlAccel, ctrlAcceleration, ctrlAccelerations
• control forces:	ctrlForce, ctrlForces
• control time:	ctrlTime, ctrlTimes
• daq displacements:	daqDisp, daqDisplacement, daqDisplacements
• daq velocities:	daqVel, daqVelocity, daqVelocities
• daq accelerations:	daqAccel, daqAcceleration, daqAccelerations
• daq forces:	daqForce, daqForces
• daq time:	daqTime, daqTimes

EXAMPLE:

```
# Define experimental setup
# ------------------------
# expSetup OneActuator $tag <-control $ctrlTag> $dir -sizeTrialOut $t $o <-trialDispFact $f>
expSetup OneActuator 1 -control 1 1 -sizeTrialOut 1 1
expSetup OneActuator 2 -control 2 1 -sizeTrialOut 1 1
...

# -----------------------------
# Start of recorder generation
# -----------------------------
expRecorder Setup -file Setup_trialDsp.out -time -setup 1 2 trialDisp
expRecorder Setup -file Setup_trialVel.out -time -setup 1 2 trialVel
expRecorder Setup -file Setup_trialAcc.out -time -setup 1 2 trialAccel
expRecorder Setup -file Setup_outDsp.out -time -setup 1 2 outDisp
expRecorder Setup -file Setup_outFrc.out -time -setup 1 2 outForce
expRecorder Setup -file Setup_ctrlDsp.out -time -setup 1 2 ctrlDisp
expRecorder Setup -file Setup_ctrlVel.out -time -setup 1 2 ctrlVel
expRecorder Setup -file Setup_ctrlAcc.out -time -setup 1 2 ctrlAccel
expRecorder Setup -file Setup_daqDsp.out -time -setup 1 2 daqDisp
expRecorder Setup -file Setup_daqFrc.out -time -setup 1 2 daqForce
# -------------------------------
# End of recorder generation
# -------------------------------
```

This example creates experimental setup recorders that write data to a file in ASCII format without headers. Trial displacements, velocities and accelerations, output displacements and forces, target (control) displacements, velocities and accelerations as well as measured (daq) displacements and forces are recorded for the previously created experimental setups with tags 1 and 2.

SignalFilter Recorder

This command is used to construct an experimental signal filter recorder object. It is used to record data from the experimental signal filters.

```
expRecorder SignalFilter <–file $fileName> <–csv $fileName> <–xml $fileName>
        <–binary $fileName> <–database $tableName> <–time> <–dt> <–filter
        $filterTag …> <–filterRange $startTag $endTag> <–filter all> respType
```

–file	to record data to a file in ASCII format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–csv	to record data to a file in ASCII comma separated format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–xml	to record data to a file in ASCII format including metadata (optional: use only one of –file, –csv, –xml, –binary, –database)
–binary	to record data to a file in binary format (optional: use only one of –file, –csv, –xml, –binary, –database)
–database	to record data to a database (optional: use only one of –file, –csv, –xml, –binary, –database)
$fileName	file where results are stored. Each line of the file contains the result for a committed state of the experimental test.
$tableName	name of table in database where results are stored. Each line of the table contains the result for a committed state of the experimental test.
–time	to place the time or pseudo-time of the analysis in the first data column (optional)
–dt	to record data at a different time increment than the analysis time step (this should be a multiple of the analysis time step)
$filterTag	tag of experimental signal filter to be recorded
$startTag, $endTag	start and end tag of range of experimental signal filters to be recorded
all	to record all experimental signal filter objects (Note: This option is recommended only for xml file output as OpenFresco might change the order of the experimental signal filter objects)
respType	defines the response type to be recorded. Since this depends on the type of the experimental signal filter the possible arguments are described with the signal filter commands.

Site Recorder

This command is used to construct an experimental site recorder object. It is used to record data from the experimental sites such as trial and output response quantities.

expRecorder Site <–file $fileName> <–csv $fileName> <–xml $fileName> <–binary $fileName> <–database $tableName> <–time> <–dt> <–site $siteTag …> <–siteRange $startTag $endTag> <–site all> respType

–file	to record data to a file in ASCII format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–csv	to record data to a file in ASCII comma separated format without headers (optional: use only one of –file, –csv, –xml, –binary, –database)
–xml	to record data to a file in ASCII format including metadata (optional: use only one of –file, –csv, –xml, –binary, –database)
–binary	to record data to a file in binary format (optional: use only one of –file, –csv, –xml, –binary, –database)
–database	to record data to a database (optional: use only one of –file, –csv, –xml, –binary, –database)
$fileName	file where results are stored. Each line of the file contains the result for a committed state of the experimental test.
$tableName	name of table in database where results are stored. Each line of the table contains the result for a committed state of the experimental test.
–time	to place the time or pseudo-time of the analysis in the first data column (optional)
–dt	to record data at a different time increment than the analysis time step (this should be a multiple of the analysis time step)
$siteTag	tag of experimental site to be recorded
$startTag, $endTag	start and end tag of range of experimental site to be recorded
all	to record all experimental site objects (Note: This option is recommended only for xml file output as OpenFresco might change the order of the experimental site objects)
respType	defines the response type to be recorded. The possible arguments are described below.

The valid queries to an experimental site when creating an *ExpSiteRecorder* object are:

trial displacements:	trialDisp, trialDisplacement, trialDisplacements
trial velocities:	trialVel, trialVelocity, trialVelocities
trial accelerations:	trialAccel, trialAcceleration, trialAccelerations
trial forces:	trialForce, trialForces
trial time:	trialTime, trialTimes
output displacements:	outDisp, outDisplacement, outDisplacements
output velocities:	outVel, outVelocity, outVelocities
output accelerations:	outAccel, outAcceleration, outAccelerations
output forces:	outForce, outForces
output time:	outTime, outTimes

EXAMPLE:

```
# Define experimental site
# ------------------------
# expSite LocalSite $tag $setupTag
expSite LocalSite 1 1
expSite LocalSite 2 2
...

# ------------------------------
# Start of recorder generation
# ------------------------------
expRecorder Site -file Site_trialDsp.out -time -site 1 2 trialDisp
expRecorder Site -file Site_trialVel.out -time -site 1 2 trialVel
expRecorder Site -file Site_trialAcc.out -time -site 1 2 trialAccel
expRecorder Site -file Site_trialTme.out -time -site 1 2 trialTime
expRecorder Site -file Site_outDsp.out -time -site 1 2 outDisp
expRecorder Site -file Site_outVel.out -time -site 1 2 outVel
expRecorder Site -file Site_outAcc.out -time -site 1 2 outAccel
expRecorder Site -file Site_outFrc.out -time -site 1 2 outForce
expRecorder Site -file Site_outTme.out -time -site 1 2 outTime
# --------------------------------
# End of recorder generation
# --------------------------------
```

This example creates experimental site recorders that write data to a file in ASCII format without headers. Trial displacements, velocities, accelerations and time as well as output displacements, velocities, accelerations, forces and time are recorded for the previously created experimental sites with tags 1 and 2.

CHAPTER 9

Server Commands

These commands are used to start the server processes. These are used during local tests using the middle-tier server and during distributed tests.

In This Chapter

startLabServer Command

This command is used to start the laboratory server process. The laboratory (or backend) server is started when running a distributed test, as shown in Figure 1b.

startLabServer $siteTag

$siteTag	tag of previously defined experimental site object

EXAMPLE:

```
# Define experimental site
# ------------------------
# expSite ActorSite $tag -setup $setupTag $ipPort <$dataSize>
expSite  ActorSite  1  -setup 1  8091

# ------------------------------
# Start the server process
# ------------------------------
# startLabServer $siteTag
startLabServer  1
```

This example uses the ActorSite to start the server.

startSimAppElemServer Command

This command is used to start the simulation application element middle-tier server process. The startSimAppElemServer command is used when the simulation application or the finite element software is using a generic-client element as shown in Figures 1a and 1b.

startSimAppElemServer $eleTag $port <–ssl>

$eleTag	tag of previously defined experimental element object
$port	IP port number of the middle-tier server
-ssl	secure transactions using OpenSSL (optional)

EXAMPLE:

```
# Define experimental element
# ---------------------------
# left column
expElement  twoNodeLink  1  1  3 -dir 2  -site 1  -initStif 2.8  -orient 0 1 0 -1 0

# ------------------------------
# Start the server process
# ------------------------------
# startSimAppElemServer $eleTag $port
startSimAppElemServer  1  8090
```

This example uses a twoNodeLink experimental element, and the middle-tier server has a port number of 8090.

startSimAppSiteServer Command

This command is used to start the simulation application site middle-tier server process. The startSimAppSiteServer command is used when the simulation application or the finite element software is using its own experimental element, as shown in Figure 6b.

startSimAppSiteServer $siteTag $port <–ssl>

$siteTag	tag of previously defined experimental site object
$port	IP port number of the middle-tier server
-ssl	secure transactions using OpenSSL (optional)

EXAMPLE:

```
# Define experimental site
# ------------------------
# expSite RemoteSite $tag <-setup $setupTag> $ipAddr $ipPort <$dataSize>
expSite  RemoteSite  1  "127.0.0.1"  8091

# ------------------------------
# Start the server process
# ------------------------------
# startSimAppSiteServer $siteTag $port
#startSimAppSiteServer 1 8090
```

This example uses a RemoteSite, and the middle-tier server has a port number of 8090.

CHAPTER 10

Miscellaneous Commands

This chapter contains miscellaneous OpenFresco commands. These are utility commands that are available in OpenFresco.

In This Chapter

Basic Model Builder

This command is used to construct the BasicBuilder object. This is the same command that is available in OpenSees.

model BasicBuilder -ndm $ndm <-ndf $ndf>

$ndm	dimension of model (1,2 or 3)

$ndf	number of degrees of freedom at node (optional)
	(default value depends on value of ndm:
	ndm=1 -> ndf=1
	ndm=2 -> ndf=3
	ndm=3 -> ndf=6)

EXAMPLE:

```
model BasicBuilder -ndm 2 -ndf 2
```

This example creates a model with two dimensions and 2 degrees of freedom per node.

Node Command

This command is used to construct a Node object. It assigns coordinates to the Node object. This is the same command that is available in OpenSees.

node $nodeTag (ndm $coords)

$nodeTag	integer tag identifying node
$coords	nodal coordinates (ndm arguments)

EXAMPLE:

```
node   1   0.0   0.0
```

This example creates a node with node tag 1 in 2D at the origin.

Load OpenFresco Package

This command is required when OpenSees is used as the finite element software. The DLLs need to be in the same folder as the OpenSees executable. The “OpenFresco Installation and Getting Started Guide” contains a list of required dll’s.

```
loadPackage OpenFresco
```

Reference:

http://neesforge.nees.org/docman/index.php?group_id=36&selected_doc_group_id=36&language_id=1 (http://neesforge.nees.org/docman/index.php?group_id=36&selected_doc_group_id=36&language_id=1)

The OpenFresco Installation and Getting Started Guide is available from the above website.

Index

Notes