1.2.8 VUEL
User subroutine to define an element.

Product: Abaqus/Explicit

> ***Warning:*** *This feature is intended for advanced users only. Its use in all but the simplest test examples will require considerable coding by the user/developer. "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual, should be read before proceeding.*

References

- "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual
- "User-defined element library," Section 27.16.2 of the Abaqus Analysis User's Manual
- "UEL," Section 1.1.22
- *UEL PROPERTY
- *USER ELEMENT

Overview

User subroutine VUEL:

- will be called for each element that is of a general user-defined element type each time element calculations are required; and
- (or subroutines called by user subroutine VUEL) must perform all of the calculations for the element, appropriate to the current activity in the analysis.

User subroutine interface

```
      SUBROUTINE VUEL(nblock,rhs,amass,dtimeStable,svars,nsvars,
     1                energy,
     2                nnode,ndofel,props,nprops,jprops,njprops,
     3                coords,mcrd,u,du,v,a,
     4                jtype,jElem,
     5                time,period,dtimeCur,dtimePrev,kstep,kinc,
     6                lflags,
     7                dMassScaleFactor,
     8                predef,npredef,
     9                jdltyp, adlmag)
C
      include 'vaba_param.inc'

C     operational code keys
      parameter ( jMassCalc            = 1,
     *            jIntForceAndDtStable = 2,
     *            jExternForce         = 3)

C     flag indices
      parameter (iProcedure = 1,
     *           iNlgeom    = 2,
     *           iOpCode    = 3,
     *           nFlags     = 3)

C     energy array indices
      parameter ( iElPd = 1,
     *            iElCd = 2,
     *            iElIe = 3,
     *            iElTs = 4,
     *            iElDd = 5,
     *            iElBv = 6,
     *            iElDe = 7,
     *            iElHe = 8,
     *            iElKe = 9,
     *            iElTh = 10,
     *            iElDmd = 11,
     *            iElDc = 12,
     *            nElEnergy = 12)

C     predefined variables indices
      parameter ( iPredValueNew = 1,
     *            iPredValueOld = 2,
     *            nPred         = 2)

C     time indices
      parameter (iStepTime  = 1,
     *           iTotalTime = 2,
     *           nTime      = 2)

                 dimension rhs(nblock,ndofel),amass(nblock,ndofel,ndofel),
     1          dtimeStable(nblock),
     2          svars(nblock,nsvars),energy(nblock,nElEnergy),
     3          props(nprops),jprops(njprops),
     4          jElem(nblock),time(nTime),lflags(nFlags),
     5          coords(nblock,nnode,mcrd),
     6          u(nblock,ndofel), du(nblock,ndofel),
     7          v(nblock,ndofel), a(nblock, ndofel),
     8          dMassScaleFactor(nblock),
     9          predef(nblock,nnode,npredef,nPred),
     *          adlmag(nblock)


            do kblock = 1,nblock
        user coding to define  rhs, amass, dtimeStable, svars and energy
       end do

      RETURN
      END
```

Variables to be defined

Some of the following arrays depend on the value of the `lflags` array.

`rhs`

An array containing the contributions of each element to the right-hand-side vector of the overall system of equations. Depending on the settings of the `lflags` array, it contains either the internal force from the element or the external load calculated from the specified distributed loads.

`amass`

An array containing the contribution of each element to the mass matrix of the overall system of equations.

All nonzero entries in `amass` should be defined. Moreover, the mass matrix must be symmetric. There are several other requirements that apply depending on the active degrees of freedom specified. These requirements are explained in detail below.

`dtimeStable`

A scalar value defining, for each element, the upper limit of the time increment for stability considerations. This would be the maximum time increment to be used in the subsequent increment for this element to be stable (to satisfy the Courant condition). This value depends strongly on the element formulation, and it is important that is computed appropriately.

`svars`

An array containing the values of the solution-dependent state variables associated with each element. The number of such variables is `nsvars` (see below). You define the meaning of these variables.

This array is passed into `VUEL` containing the values of these variables at the start of the current increment. In most cases they should be updated to be the values at the end of the increment. In rare cases such an update is not required.

`energy`

The array `energy` contains the values of the energy quantities associated with each element. The values in this array when `VUEL` is called are the element energy quantities at the start of the current increment. They should be updated to the correct values at the end of the current increment; otherwise, plots of the energy balance for the entire model will not be accurate. Depending on the element formulation, many of these energies could be zero at all times. The entries in the array are as follows:

`energy(nblock,iElPd )`	Plastic dissipation.
`energy(nblock,iElCd)`	Creep dissipation.
`energy(nblock,iElIe)`	Internal energy.
`energy(nblock,iElTs)`	Transverse shear energy.
`energy(nblock,iElDd)`	Material damping dissipation.
`energy(nblock,iElBv)`	Bulk viscosity dissipation.
`energy(nblock,iElDe)`	Drill energy.
`energy(nblock,iElHe)`	Hourglass energy.
`energy(nblock,iElKe)`	Kinetic energy.
`energy(nblock,iElTh)`	Heat energy.
`energy(nblock,iElDmd)`	Damage dissipation.

`energy(nblock,iElDc )`	Distortion control energy.

Variables passed in for information

Arrays:

`props`

A floating point array containing the `nprops` real property values defined for use with each element processed. `nprops` is the user-specified number of real property values. See "Defining the element properties" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual.

`jprops`

An integer array containing the `njprops` integer property values defined for use with each element processed. `njprops` is the user-specified number of integer property values. See "Defining the element properties" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual.

`coords`

An array containing the original coordinates of the nodes of the element. `coords(kblock,k1,k2)` is the `k2`th coordinate of the `k1`th node of the `kblock` element.

`u, du, v, a`

Arrays containing the basic solution variables (displacements, rotations, temperatures, pressures, depending on the degree of freedom) at the nodes of the element. Values are provided as follows, illustrated below for the `k1`th degree of freedom of the `kblock` element:

`u(kblock,k1)`	Total value of the variables (such as displacements or rotations) at the end of the current increment.
`du(kblock,k1)`	Incremental values of the variables in the current increment.
`v(kblock,k1)`	Time rate of change of the variables (velocities, rates of rotation) at the midpoint of the increment.
`a(kblock,k1)`	Accelerations of the variables at the end of the current increment.

`jElem`

`jElem(kblock)` contains the element number for the `kblock` element.

`adlmag`

`adlmag(kblock)` is the total load magnitude of the load type `jdltyp` (integer identifying the load number for distributed load type U*n*) distributed load at the end of the current increment for distributed loads of type U*n*.

`predef`

An array containing the values of predefined field variables, such as temperature in an uncoupled stress/displacement analysis, at the nodes of the element (“Predefined fields,” Section 28.6.1 of the Abaqus Analysis User's Manual).

The second index, `k2`, indicates the local node number on the `kblock` element. The third index, `k3`, indicates the variable: the temperature is stored if the index is 1, and the predefined field variables are stored if the indices are greater than or equal to 2. The fourth index of the array, `k4`, is either 1 or 2, with 1 indicating the value of the field variable at the end of the increment and 2 indicating the value of the field variable at the beginning of the increment.

`predef(kblock,k2,1,k4)`	Temperature.
`predef(kblock,k2,2,k4)`	First predefined field variable.
`predef(kblock,k2,3,k4)`	Second predefined field variable.
Etc.	Any other predefined field variable.
`predef(kblock,k2,k3,k4)`	Value of the `(k3–1)`th predefined field variable at the `k2`th node of the element at the beginning or the end of the increment.
`predef(kblock,k2,k3,1)`	Values of the variables at the end of the current increment.
`predef(kblock,k2,k3,2)`	Values of the variables at the beginning of the current increment.

`lflags`

An array containing the flags that define the current solution procedure and requirements for element calculations.

`lflags(iProcedure)`	Defines the procedure type. See “Results file output format,” Section 5.1.2 of the Abaqus Analysis User's Manual, for the key used for each procedure.
`lflags(iNlgeom)=0`	Small-displacement analysis.
`lflags(iNlgeom)=1`	Large-displacement analysis (nonlinear geometric effects included in the step; see “General and linear perturbation procedures,” Section 6.1.2 of the Abaqus Analysis User's Manual).
`lflags(iOpCode)=jMassCalc`	Define the mass matrix `amass` in the beginning of the analysis.
`lflags(iOpCode)=jIntForceAndDtStable`	Define the element internal force. Define the stable time increment as well.
`lflags(iOpCode)=jExternForce`	Define the distributed load effect on the external force associated with the element.

`dMassScaleFactor`

An array containing the mass scale factors for each element.

`time(iStepTime)`

Current value of step time.

`time(iTotalTime)`

Current value of total time.

Scalar parameters:

`nblock`

Number of user elements to be processed in this call to `VUEL`.

`dtimeCur`

Current time increment.

`dtimePrev`

Previous time increment.

`period`

Time period of the current step.

`ndofel`

Number of degrees of freedom in the elements processed.

`nsvars`

User-defined number of solution-dependent state variables associated with the element ("Defining the number of solution-dependent variables that must be stored within the element" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual).

`nprops`

User-defined number of real property values associated with the elements processed ("Defining the element properties" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual).

`njprops`

User-defined number of integer property values associated with the elements processed ("Defining the element properties" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual).

`mcrd`

`mcrd` is defined as the maximum of the user-defined maximum number of coordinates needed at any node point ("Defining the maximum number of coordinates needed at any nodal point" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual) and the value of the largest active degree of freedom of the user element that is less than or equal to 3. For example, if you specify that the maximum number of coordinates is 1 and the active degrees of freedom of the user element are 2, 3, and 6 `mcrd` will be 3. If you specify that the maximum number of coordinates is 2 and the active degree of freedom of the user element is 11, `mcrd` will be 2.

`nnode`

User-defined number of nodes on the elements ("Defining the number of nodes associated with the element" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual).

`jtype`

Integer defining the element type. This is the user-defined integer value *n* in element type VU*n* ("Assigning an element type key to a user-defined element" in "User-defined elements," Section 27.16.1 of the Abaqus Analysis User's Manual).

`kstep`

Current step number.

`kinc`

Current increment number.

`npredef`

Number of predefined field variables, including temperature. For user elements Abaqus/Explicit uses one value for each field variable per node.

`VUEL` conventions

The solution variables (displacement, velocity, etc.) are arranged on a node/degree of freedom basis. The degrees of freedom of the first node are first, followed by the degrees of freedom of the second node, etc. The degrees of freedom that will be updated automatically in Abaqus/Explicit are: 1–3 (displacements), 4–6 (rotations), 8 (pressure), and 11 (temperature). Depending on the procedure type (see below), only some of the degrees of freedom listed above will be updated. Other degrees of freedom will not be updated by the time integration procedure in Abaqus/Explicit and, hence, should not be used.

The mass matrix defined in user subroutine `VUEL` must be symmetric. In addition, the following requirements apply:

- The mass matrix entries associated with the translational degrees of freedom for a particular node must be diagonal. Moreover, these diagonal entries must be equal to each other.
- There must be no coupling (off-diagonal) entries specified between degrees of freedom of different kinds. For example, you cannot specify nonzero mass matrix entries to couple the translational degrees of freedom to the rotational degrees of freedom.
- There must be no coupling (off-diagonal) entries specified between degrees of freedom belonging to different nodes.

You must be using appropriate lumping techniques to provide a mass matrix that follows these requirements. For the rotational degrees of freedom at a particular node in three-dimensional analyses, you can specify a fully populated symmetric 3 × 3 inertia tensor.

Usage with general nonlinear procedures

The following illustrates the use in explicit dynamic procedures:

Direct-integration explicit dynamic analysis (`lflags(iProcedure)=17`)

- Automatic updates for degrees of freedom 1–6, 8, and 11.
- The governing equations are as described in “Explicit dynamic analysis,” Section 6.3.3 of the Abaqus Analysis User's Manual.
- Coding for the operational code `lflags(iOpCode)=jExternForce`is optional.

Transient fully coupled thermal-stress analysis (`lflags(iProcedure)=74`)

- Automatic updates for degrees of freedom 1–6 and 8.
- The governing equations are as described in “Fully coupled thermal-stress analysis in Abaqus/Explicit” in “Fully coupled thermal-stress analysis,” Section 6.5.4 of the Abaqus Analysis User's Manual.
- Coding for the operational code `lflags(iOpCode)=jExternForce` is optional.

Example: Structural user element

A structural user element has been created to demonstrate the usage of subroutine `VUEL`. These user-defined elements are applied in a number of analyses. The following excerpt is from the verification problem that invokes the structural user element in an explicit dynamic procedure:

```
*USER ELEMENT, NODES=2, TYPE=VU1, PROPERTIES=4, COORDINATES=3,
VARIABLES=12
1, 2, 3
*ELEMENT, TYPE=VU1
101, 101, 102
*ELGEN, ELSET=VUTRUSS
101, 5
*UEL PROPERTY, ELSET=VUTRUSS
0.002, 2.1E11, 0.3, 7200.
```

The user element consists of two nodes that are assumed to lie parallel to the *x*-axis. The element behaves similarly to a linear truss element. The supplied element properties are the cross-sectional area, Young's modulus, Poisson's ratio, and density, respectively.

The next excerpt shows the listing of the subroutine. The user subroutine has been coded for use in an explicit dynamic analysis. The names of the verification input files associated with the subroutine and these procedures can be found in “`VUEL`,” Section 4.1.30 of the Abaqus Verification Manual.

```
      subroutine vuel(
     *     nblock,
     *     rhs,amass,dtimeStable,
     *     svars,nsvars,
     *     energy,
     *     nnode,ndofel,
     *     props,nprops,
     *     jprops,njprops,
     *     coords,ncrd,
     *     u,du,v,a,
     *     jtype,jElem,
     *     time,period,dtimeCur,dtimePrev,kstep,kinc,lflags,
     *     dMassScaleFactor,
     *     predef,npredef,
     *     ndload,adlmag)

      include 'vaba_param.inc'

c     operation code
      parameter ( jMassCalc            = 1,
     *            jIntForceAndDtStable = 4)

c     flags
      parameter (iProcedure = 1,
     *           iNlgeom    = 2,
     *           iOpCode    = 3,
     *           nFlags     = 3)

c     procedure flags
      parameter ( jDynExplicit = 17 )

c     time
      parameter (iStepTime  = 1,
     *           iTotalTime = 2,
     *           nTime      = 2)

c     energies
      parameter ( iElPd = 1,
     *            iElCd = 2,
     *            iElIe = 3,
     *            iElTs = 4,
     *            iElDd = 5,
     *            iElBv = 6,
     *            iElDe = 7,
     *            iElHe = 8,
     *            iElKe = 9,
     *            iElTh = 10,
     *            iElDmd = 11,
     *            iElDc = 12,
     *            nElEnergy = 12)

      parameter (factorStable = 0.99d0)
      parameter ( zero = 0.d0, half = 0.5d0, one = 1.d0, two=2.d0 )
C
      dimension rhs(nblock,ndofel), amass(nblock,ndofel,ndofel),
     *     dtimeStable(nblock),
     *     svars(nblock,nsvars), energy(nblock,nElEnergy),
     *     props(nprops), jprops(njprops),
     *     jElem(nblock), time(nTime), l(nFlags),
     *     coords(nblock,nnode,ncrd), u(nblock,ndofel),
     *     du(nblock,ndofel), v(nblock,ndofel), a(nblock, ndofel),
     *     predef(nblock, nnode, npredef, nPred), adlmag(nblock),
     *     dMassScaleFactor(nblock)


c     Notes:
c        Define only nonzero entries; the arrays to be defined have
```

```
c         been zeroed out just before this call

      if (jtype .eq. 1001 .and.
     *    lflags(iProcedure).eq.jDynExplicit) then

         area0 = props(1)
         eMod  = props(2)
         anu   = props(3)
         rho   = props(4)

         eDampTra    = zero
         amassFact0  = half*area0*rho

         if ( lflags(iOpCode).eq.jMassCalc ) then
            do kblock = 1, nblock
c              use original distance to compute mass
               alenX0 = (coords(kblock,2,1) - coords(kblock,1,1))
               alenY0 = (coords(kblock,2,2) - coords(kblock,1,2))
               alenZ0 = (coords(kblock,2,3) - coords(kblock,1,3))
               alen0 = sqrt(alenX0*alenX0 + alenY0*alenY0 +
     *                      alenZ0*alenZ0)
               am0   = amassFact0*alen0
               amass(kblock,1,1) = am0
               amass(kblock,2,2) = am0
               amass(kblock,3,3) = am0
               amass(kblock,4,4) = am0
               amass(kblock,5,5) = am0
               amass(kblock,6,6) = am0

            end do
         else if ( lflags(iOpCode) .eq.
     *            jIntForceAndDtStable) then
            do kblock = 1, nblock
               alenX0 = (coords(kblock,2,1) - coords(kblock,1,1))
               alenY0 = (coords(kblock,2,2) - coords(kblock,1,2))
               alenZ0 = (coords(kblock,2,3) - coords(kblock,1,3))
               alen0 = sqrt(alenX0*alenX0 + alenY0*alenY0 +
     *                      alenZ0*alenZ0)
               vol0 = area0*alen0
               amElem0 = two*amassFact0*alen0

               alenX = alenX0
     *              +  (u(kblock,4)        -  u(kblock,1))
               alenY = alenY0
     *              +  (u(kblock,5)        -  u(kblock,2))
               alenZ = alenZ0
     *              +  (u(kblock,6)        -  u(kblock,3))
               alen = sqrt(alenX*alenX + alenY*alenY + alenZ*alenZ)
               area   = vol0/alen
               ak     = area*eMod/alen

c              stable time increment for translations
               dtimeStable(kblock) = factorStable*sqrt(amElem0/ak)
```

```
c               force = E * logarithmic strain *current area
                strainLog = log(alen/alen0)
                fElasTra = eMod*strainLog*area

                forceTra = fElasTra

c               assemble internal load in RHS
                rhs(kblock,1) = -forceTra
                rhs(kblock,4) =  forceTra

c               internal energy calculation
                alenOld  = svars(kblock,1)
                fElasTraOld = svars(kblock,2)

                energy(kblock, iElIe) = energy(kblock, iElIe) +
     *              half*(fElasTra+fElasTraOld)*(alen - alenOld)

c               update state variables
                svars(kblock,1) = alen
                svars(kblock,2) = fElasTra

             end do
          end if

       end if
c
       return
       end
```