ENGINEERING & FEA

Mastering Abaqus User Subroutines: A Technical Guide to UEL and VUEL Custom Element Development

Published: November 04, 2026 | Tags: Abaqus | User Subroutines | Fortran | FEA | VUEL | UEL | Simulation

In the complex landscape of computational mechanics and Finite Element Analysis (FEA), **Abaqus** stands as a premier tool for simulating structural, thermal, and fluid-dynamic behavior. However, the pre-built libraries provided by commercial software packages often fall short when researchers or engineers encounter novel materials, unconventional element formulations, or specific constitutive behaviors that have not yet been standardized. This is where **Abaqus User Subroutines** become indispensable. By leveraging the power of Fortran, users can extend the core functionality of Abaqus, creating bespoke solutions that push the boundaries of virtual prototyping.

The Strategic Role of User Subroutines in Finite Element Analysis

User subroutines are programmable interfaces that allow the user to define specific aspects of a simulation that are not available through the standard Abaqus interface. Whether it is defining a new material model (UMAT/VUMAT), a custom loading condition (DLOAD), or a completely new finite element (UEL/VUEL), these subroutines provide the flexibility required for high-level academic research and advanced industrial applications. The integration of these routines requires a bridge between the Abaqus solver and a **Fortran compiler**, creating a seamless execution environment where custom code is compiled and linked during the analysis runtime.

The Architecture of Custom Subroutines

At its core, a subroutine is a self-contained block of code designed to perform a specific task when called by the main Abaqus executable. In the context of **OOP (Object-Oriented Programming)** principles, though Fortran 77-the traditional language of Abaqus subroutines-is procedural, these subroutines function similarly to methods or operations. They receive input data (such as current strain, time increment, or nodal coordinates), perform calculations, and return updated values (such as stress, internal forces, or state variables) to the solver. This modularity allows for the isolation of complex physics within a structured environment.

Deep Dive into User-Defined Elements: UEL and VUEL

Perhaps the most powerful capability within the Abaqus ecosystem is the ability to define custom elements. While most users are familiar with standard continuum elements (like C3D8R), certain

problems-such as multi-physics coupling, non-local damage models, or specialized beam theories-require a **User-Defined Element (UEL)** for Abaqus/Standard or a **VUEL** for Abaqus/Explicit.

User Subroutine UEL (Abaqus/Standard)

The **UEL** subroutine is called for every element defined as a user element in an Abaqus/Standard analysis. Its primary responsibility is to provide the **stiffness matrix** and the **residual force vector** . Because Abaqus/Standard uses an implicit integration scheme (typically Newton-Raphson), the solver requires the Jacobian (the derivative of the internal force with respect to nodal displacements) to achieve convergence.

- RHS (Right Hand Side): This vector contains the contribution of the element to the global residual equations.
- AMATRX: This is the element Jacobian (stiffness) matrix. For a non-linear problem, the accuracy of this matrix is critical for the convergence rate of the simulation.
- STATEV: An array containing solution-dependent state variables (SDVs), which allow the element to "remember" its internal state (like plastic strain or damage) between increments.

User Subroutine VUEL (Abaqus/Explicit)

In contrast, **VUEL** is the counterpart for Abaqus/Explicit. Unlike UEL, VUEL does not require a stiffness matrix because the explicit method solves the equations of motion directly using a diagonal mass matrix. Instead, the focus of VUEL is on calculating **internal forces** and updating state variables over very small time increments. The subroutine must perform all calculations appropriate to the current activity in the analysis, accounting for material non-linearity and geometric deformations.

Key Differences and Operational Logic

Feature	UEL (Abaqus/Standard)	VUEL (Abaqus/Explicit)
Integration Scheme	Implicit (Newton-Raphson)	Explicit (Central Difference)
Primary Output	Stiffness Matrix & Residual Forces	Internal Forces & Mass Contribution
Convergence	Required (Iterative)	Conditionally Stable (Time-step limited)
State Variables	Maintained via STATEV	Maintained via stateVLocal
Complexity	Higher (due to Jacobian calculation)	Moderate (focus on force/mass)

Technical Framework: Programming User Subroutines in Fortran

To successfully write and run an Abaqus user subroutine, one must adhere to the strict calling conventions defined in the **Abaqus User Subroutines Reference Manual**. A typical subroutine begins with a specific header that defines the arguments passed by the solver. For example, a VUEL interface looks like this:

Data Management and State Variables

One of the most critical aspects of subroutine development is the management of **Solution-Dependent State Variables (SDVs)**. These are used to store data that must persist from one increment to the next. For instance, if you are modeling a material that undergoes hardening, the current level of hardening must be stored in an SDV. In **VUEL**, the argument allows for vectorized processing, meaning the subroutine processes multiple elements in a single call to improve computational efficiency. This requires the programmer to write loops that iterate through the block, applying the same physical logic to each element index.

The Importance of Orientation and Basis Directions

When developing custom elements, the definition of material directions (DIRECT) and the element basis directions is paramount. **User subroutine VUCHARLENGTH** is often used in conjunction with element routines to define characteristic element lengths, which are vital for regularizing damage models and avoiding mesh dependency in localized failure simulations. Proper orientation ensures that the calculated stresses and strains are transformed correctly from the local element system to the global coordinate system.

Step-by-Step Workflow for Implementation

Implementing a user subroutine is a multi-stage process that involves coding, compiling, and validation. Following a structured workflow minimizes the risk of runtime errors and unphysical results.

1. Mathematical Formulation: Deriving the element's internal force vector and, if using UEL, the tangent stiffness matrix using the Principle of Virtual Work.
2. Fortran Coding: Translating the mathematical model into Fortran code. It is highly recommended to use implicit none to avoid variable naming conflicts and to use double precision for all floating-point calculations.
3. Compiler Linking: Configuring the environment. Abaqus requires a specific version of the Intel Fortran Compiler and Microsoft Visual Studio. Use the command abaqus verify -user_std or -user_exp to check the link.

4. Verification: Start with a single-element test. Compare the results of the custom element against a standard Abaqus element (if a comparable one exists) or against an analytical solution.
5. Execution: Run the full model using the command line: abaqus job=myjob user=mysubroutine.f.

Case Study: Implementing a 1D User Element

A common question among researchers is: *Is it possible to write a 1D subroutine in ABAQUS?* The answer is yes. Even though Abaqus is primarily used for 2D and 3D simulations, a **UEL** or **VUEL** can be formulated for 1D behavior (such as a specialized truss or a fiber in a composite). The user defines the nodal coordinates in the input file and the subroutine interprets the distance between these nodes as the 1D length, calculating axial force based on the defined constitutive law.

The Extended Subroutine Library: Beyond Elements

While UEL and VUEL represent the peak of customization, Abaqus offers a vast array of other subroutines for different simulation needs. Understanding these can prevent the "reinventing of the wheel" during model development.

- UMAT / VUMAT: The most widely used subroutines. They allow for custom constitutive material laws (e.g., hyperelasticity, viscoplasticity).
- USDFLD / VUSDFLD: Used to redefine field variables at material points, often used for temperature-dependent or moisture-dependent properties.
- DLOAD / VDLOAD: For defining complex, non-uniform distributed loads that may change with position or time.
- FRIC / VFRIC: Used to define sophisticated friction laws between contact surfaces, going beyond the standard Coulomb friction model.
- GAPCON: Specifically for thermal analysis, defining the conductance between surfaces based on clearance or pressure.

Troubleshooting and Performance Optimization

Writing subroutines is prone to technical pitfalls. Debugging requires a systematic approach to identify whether an issue is numerical, logical, or environmental.

Common Error Modes

1. **Compilation Errors:** Usually caused by syntax errors in Fortran or a mismatch between the Abaqus version and the compiler version. Always check the and files for error messages.2. **Zero**

Pivot / Divergence: In UEL, this often indicates an error in the calculation of the **AMATRX** (Jacobian). If the stiffness matrix is not consistent with the residual force vector, the Newton-Raphson iterations will fail.3. **Excessive Distortion:** In VUEL, if the internal forces are calculated incorrectly, the nodes may move unphysically, leading to element inversion. This is often solved by checking the unit consistency and the time-step stability limit.4. **Memory Access Violations:** Often caused by indexing an array (like) out of its bounds. Fortran does not always provide clear warnings for this, so manual checks are necessary.

Performance Tuning

Since user subroutines are called millions of times in a large simulation, their efficiency is critical. Using vectorized loops in **VUEL** (exploiting the argument) and minimizing expensive operations (like trigonometric functions or matrix inversions) within the inner loops can significantly reduce simulation time. Additionally, using **Common Blocks** or **Modules** can help in sharing data across different subroutines without the overhead of passing large argument lists.

The Impact of Custom Subroutines on Engineering Innovation

The ability to customize the finite element environment through subroutines like **VUEL** and **UEL** transforms Abaqus from a static tool into a dynamic research platform. It allows engineers to simulate the failure of aerospace composites with high fidelity, model the bio-mechanical response of human tissue, and explore the behavior of smart materials in real-time. By mastering the interface between numerical methods and Fortran programming, technical professionals can ensure that their simulations are not limited by software constraints, but rather only by their understanding of the underlying physics. As the field of computational engineering continues to evolve, the proficiency in writing robust, efficient, and accurate user subroutines will remain a hallmark of a senior technical practitioner.

Ultimately, the successful deployment of a user subroutine hinges on a deep understanding of the **Finite Element Method**, the specific mechanics of the problem at hand, and the rigorous application of programming best practices. Whether you are defining a new frictional coefficient via **GAPCON** or implementing a complex 3D element in **VUEL**, the principles of data integrity, numerical stability, and clear documentation are the keys to producing reliable and impactful simulation data.