



**ZAMINEX**  
**(User Manual)**  
Geophysical Modeling and Inversion Software

Version: 1.01  
Last updated: January 2026

*Copyright © 2025-2026 Geotexera Inc.*

This manual provides a guide to using the ZAMINEX graphical user interface (GUI). It focuses on how to use the software interface, create workflows, manage projects, and execute programs. For specific program and application details, please refer to the user manuals for MAGNUM & PODIUM, TETRIUM, and ParaView.

CONTENTS

Introduction.....	2
Understanding The Interface.....	3
The Menu Bar.....	3
Working With Programs.....	3
Creating Workflows.....	5
Running Programs And Workflows.....	5
Project Management.....	6
Best Practices.....	6
Appendix A: Quick Reference.....	8
Appendix B: File Formats.....	9
Appendix C: Example.....	14

## 1. INTRODUCTION

ZAMINEX is a graphical workflow management system designed for geophysical modeling and inversion programs. Rather than running programs individually from the command line, ZAMINEX provides a visual interface that allows you to create, manage, and execute sequences of data processing programs. This approach makes complex workflows more intuitive and manageable.

ZAMINEX consists of three main components:

- **MAGNUM:** Programs for geophysical modeling and inversions.
- **PODIUM:** Programs for data and model preparation, organized into categories.
- **TETRIUM:** Mesh generation application for unstructured meshes.

To visualize the models and data, we convert them to VTU format so they can be opened and viewed in ParaView. ParaView is an open-source, multi-platform data analysis and model visualization application based on Visualization Toolkit (VTK), and can be downloaded from its official website: [www.paraview.org](http://www.paraview.org).

ZAMINEX interface has three main areas: Category Panel (left) for browsing programs, Work Panel (center) for building workflows, and Log Panel (bottom) for execution feedback. The menu bar provides access to file operations, edit functions, settings, and auxiliary applications.

*NOTE: Input/output file names must be in English. We also recommend using English for all folder names in the file path, as some non-English characters may cause file-not-found errors.*

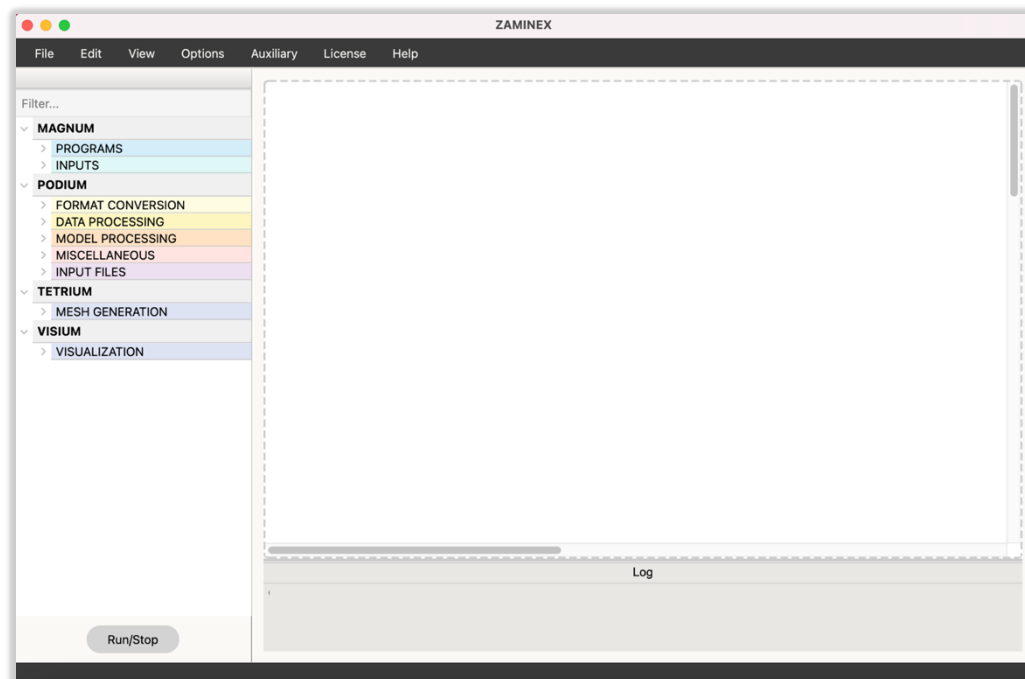


Figure 1: ZAMINEX interface.

## 2. UNDERSTANDING THE INTERFACE

### Category Panel (Left Side)

The Category Panel organizes programs into a hierarchical tree with PODIUM and MAGNUM sections. Programs are grouped into categories and subsections. A filter box at the top allows you to search for programs by name. Double-click a program name to add it to your canvas. Each category has a distinct background color that helps identify program types.

### Work Panel (Center)

The Work Panel is your workspace for building workflows. The canvas extends beyond the visible area. You can zoom with the mouse wheel and pan by dragging the background. Program widgets appear as rounded rectangles with connection dots on left (inputs) and right (outputs) sides. Widget borders indicate state: thin (ready), blinking (running), thick (completed), red (failed).

Right-clicking on the work panel background shows a "Refresh" option. It resets all widget borders to thin and black, as if they haven't been run yet. This clears thick black borders (completed) and red borders (failed), returning widgets to their initial state.

### Log Panel (Bottom)

The Log Panel displays real-time execution messages, program output, and errors. It's scrollable and automatically updates during execution. It can be visible by checking Log in the menu View.

## 3. THE MENU BAR

File Menu: New Project (Ctrl+N), Open Project (Ctrl+O), Save Project (Ctrl+S), Exit. Project files use .gtxr extension and store all widgets, connections, parameters, and canvas state.

Edit Menu: Undo (Ctrl+Z) and Redo (Ctrl+Y) for reversing actions. Stacks are cleared when saving or loading projects.

View Menu: Toggle Log Panel visibility, Zoom In/Out/Reset controls. Works with mouse wheel zooming.

Options Menu: Local Runs (configure running jobs on local machine), Cluster Runs (submit jobs to High-Performance Computing 'HPC' clusters).

Auxiliary Menu: Launch TETRIUM (mesh generation), AI Assistant.

License Menu: View license information and status.

Help Menu: About dialog, Docs and Examples (opens web browser).

## 4. WORKING WITH PROGRAMS

### Understanding Widgets and Programs

Each widget in ZAMINEX represents an independent program that typically runs from the command line. These programs have their own command-line interfaces with specific inputs, outputs, and parameters. The widget's configuration dialog provides a graphical interface to these command-line parameters. When you configure a widget, you're essentially setting up the command-line arguments that will be passed to the underlying program when it executes.

### Adding Programs

Double-click a program name in the Category Panel to add it to the canvas. You can add multiple instances of the same program; they're automatically numbered (e.g., "Mesh 01", "Mesh 02"). Drag widgets to reposition them.

### Configuring Programs

Double-click a widget to open its configuration dialog. Dialogs are organized into logical sections. Common parameters are visible immediately; advanced parameters are in a "More..." dialog. Input controls include file paths (with Browse buttons and optional "Ext" button), dropdowns, spin boxes, and checkboxes. Some file inputs have an "Ext" button that controls whether file extensions are preserved or removed when selecting files. Click "OK" to save or "Cancel" to discard. Settings are automatically saved per widget instance. Each dialog includes a "Help" button for program documentation.

### Context Menu (Right-Click)

Regular program widgets: Open (configuration dialog), Rename, Duplicate, Delete, Run, Stop.

Run/Stop widgets: Open, Delete. Options are enabled/disabled based on widget state (e.g., Run disabled when already running).

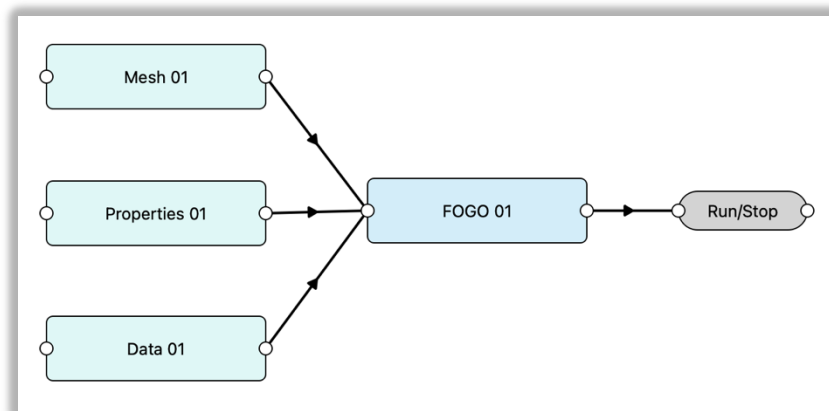


Figure 2: A workflow for forward modeling.

## 5. CREATING WORKFLOWS

### Creating Connections

A workflow is a sequence of connected programs. Connections (arrows) show data flow direction. To create a connection, drag from the right dot (output) of one widget to the left dot (input) of another, or click both dots sequentially.

### Connection States

Active connections appear as black lines with black arrowheads; inactive (deactivated) connections are gray. Right-click a connection to Deactivate, Activate (if deactivated), or Delete. Double-click a connection to delete it. Only active (black) connections are followed during execution.

### Workflow Patterns

You can branch (one output to multiple inputs) or merge (multiple outputs to one input). ZAMINEX prevents circular connections. Programs with no input connections run first. Programs wait for all input programs to complete before executing. ZAMINEX automatically determines execution order based on dependencies.

### Design Tips

Place input programs on the left, output programs on the right. Use vertical space for parallel branches. Avoid crossing connections when possible. Use descriptive names for multiple instances (e.g., "Mesh\_Topo" instead of "Mesh 01").

## 6. RUNNING PROGRAMS AND WORKFLOWS

### Three Ways to Run/Stop

1. Program Dialog: Open widget dialog, use "Run" and "Stop" buttons at bottom.
2. Context Menu: Right-click widget, select "Run" or "Stop".
3. Run/Stop Widget: For workflow sequences (see below).

When you run a program in ZAMINEX (either a single widget or a connected series), ZAMINEX is effectively running the program exactly like you would in a terminal: ``program input(s) output(s) [options]``. The full executing command is printed in the Log Panel, so you can see precisely what is being run and verify that it is correct by comparing it with the command-line usages described in the PODIUM and MAGNUM documentation (or the Help section of dialog window).

### Visual Feedback

The Log Panel displays execution messages and errors. And, the widget border signs show the stage of runs. During execution, widget borders blink. On completion, borders become thick. If failed, borders become red.

NOTE that these widget border signs are for essential executions and do not support the advanced ones. Thus, in addition to them, after each program execution, validation must be performed to ensure outputs are generated correctly. Validation consists of: (1) checking the ZAMINEX log panel for errors, (2) verifying the generated output file in its designated folder, and (3) inspecting the .vtu file in ParaView if one was created.

### **Using Run/Stop Widget**

Add "Run/Stop" from Category Panel. Connect it to final program(s) in your workflow. Double-click to open dialog with "Run" and "Stop" buttons.

Clicking "Run" traces back through all active connections to find every program in the workflow. It identifies starting points (programs with no inputs) and executes them first. Programs with dependencies wait for all inputs to complete. The "Stop" button halts all running programs in the workflow.

You can have multiple Run/Stop widgets controlling different workflow sequences.

## **7. PROJECT MANAGEMENT**

### **Saving Projects**

Save workflows as .gtxr files (File > Save Project or Ctrl+S). Project files store all widgets, positions, connections, parameters, and canvas state. Files are portable but may require path updates when moved between computers.

### **Opening Projects**

Open saved projects (File > Open Project or Ctrl+O). ZAMINEX restores everything exactly as saved. Use Options > Program Paths to configure executable paths if needed when moving projects.

### **New Projects**

Create new projects (File > New Project or Ctrl+N). ZAMINEX prompts to save unsaved changes first.

## **8. BEST PRACTICES**

### **Workflow Design**

Design for clarity: place inputs on left, outputs on right. Use descriptive names for multiple instances. Avoid crossing connections. Group related programs.

### **File Management**

Use consistent folder structures. Keep related files together. Create project-specific folders. Clean up temporary files regularly.

### **Saving and Performance**

Save frequently. Save incremental versions with version numbers or dates. For large workflows, break into smaller chunks and save intermediate results. Monitor system resources.

### **Troubleshooting**

Check Log Panel for error messages (View > Log or Ctrl+L). Verify required parameters are filled and input files exist. Check file paths, especially after moving files. Review connections to ensure correct execution order. Red borders indicate errors; right-click widget for details.

We recommend running each widget independently and reviewing its outputs before using it within a workflow!

## APPENDIX A: QUICK REFERENCE

### Keyboard Shortcuts:

New Project: Ctrl+N (Cmd+N Mac) | Open: Ctrl+O (Cmd+O Mac) | Save: Ctrl+S (Cmd+S Mac)  
| Undo: Ctrl+Z (Cmd+Z Mac) | Redo: Ctrl+Y (Cmd+Y Mac) | Toggle Log: Ctrl+L (Cmd+L Mac)  
| Zoom In: Ctrl++ | Zoom Out: Ctrl+- | Reset Zoom: Ctrl+0

### Mouse Operations:

- Double-click widget: Open dialog | Right-click widget: Context menu
- Drag widget: Move | Drag background: Pan | Mouse wheel: Zoom
- Drag right dot to left dot: Create connection | Right-click arrow: Connection menu
- Double-click arrow: Delete connection

### Visual Indicators:

Thin border: Ready | Blinking border: Running | Thick border: Completed | Red border: Failed



## APPENDIX B: FILE FORMATS

Some of the most commonly used files are brought here.

### ➤ **NODE FILES (.node):**

Node files contain point data with coordinates and optional attributes. They are used for two main purposes: (1) “**standalone point data**” (e.g., survey stations, data points, topography points) where each node represents a physical location with associated measurements or properties, and (2) “**mesh nodes**” where nodes define the vertices of a mesh (tetrahedra, triangles, etc.) used for numerical modeling and inversion.

#### **Format:**

```
n_nodes n_dim n_attr n_boundary
node_id x y z [attributes] [boundary_markers]
node_id x y z [attributes] [boundary_markers]
...
```

#### **Header Line:**

- ``n_nodes``: Total number of nodes in the file
- ``n_dim``: Spatial dimension (typically ``3`` for 3D data with X, Y, Z coordinates, or ``2`` for 2D data)
- ``n_attr``: Number of attribute columns per node (can be 0 or more)
- ``n_boundary``: Number of boundary marker columns per node (can be 0 or more)

#### **Data Rows:**

- ``node_id``: Unique identifier for the node (typically sequential integers starting from 1)
- ``x y z``: Spatial coordinates (X, Y, Z in 3D; for 2D, only X and Y are used)
- ``[attributes]``: One or more attribute values associated with the node
- ``[boundary_markers]``: One or more boundary marker values (used to identify boundary conditions, regions, or material properties)

**Attributes** are scalar or vector values associated with each node. Common examples include:

- Physical properties: Magnetic susceptibility, density, electrical conductivity, etc.
- Observations: Measured magnetic field, gravity, electromagnetic responses
- Model parameters: Inverted property values, predicted responses
- Auxiliary data: Elevation, temperature, or any other node-associated quantity

**Boundary markers** are integer or floating-point values used to:

- Identify boundary conditions (e.g., Dirichlet, Neumann boundaries)
- Mark different regions or materials in the model

- Label nodes belonging to specific surfaces or interfaces
- Distinguish between interior and boundary nodes

**Example 1:** Standalone Point Data (Survey Stations):

```
1000 3 1 0
1 568607.037 6876589.392 1044.6 55071.193
2 568607.287 6876589.964 1044.65 55071.343
3 568607.538 6876590.537 1044.7 55071.497
...
```

This example shows 1000 survey stations in 3D space, each with 1 attribute (magnetic field value). No boundary markers are used (`n\_boundary` = 0).

**Example 2:** Mesh Nodes with Attributes and Boundary Markers:

```
5000 3 2 1
1 568514.65 6875797.68 900.0 0.01 0.5 1
2 568520.12 6875800.45 901.2 0.015 0.6 1
3 568515.89 6875798.23 900.5 0.012 0.55 0
...
```

This example shows 5000 mesh nodes in 3D space, each with 2 attributes (e.g., magnetic susceptibility and density) and 1 boundary marker (e.g., `1` for boundary nodes, `0` for interior nodes).

**Example 3:** Topography Points (No Attributes):

```
2000 3 0 0
1 568600.0 6876500.0 1000.0
2 568601.0 6876501.0 1001.5
3 568602.0 6876502.0 1002.3
...
```

This example shows 2000 topography points with only coordinates, no attributes or boundary markers.

**Notes:**

- Node IDs must be unique and typically start from 1
- Coordinates are usually in meters (or consistent units)
- Attributes can be any floating-point values
- Boundary markers are often integers but can be floating-point
- When used with element files (`.ele`), node IDs must match between files
- For visualization, nodes can be converted to VTU format

## ➤ ELEMENT FILES (.ele):

Element files define the connectivity of mesh elements, specifying which nodes form each element. For 3D meshes, elements are typically tetrahedra (4-node elements), which are the fundamental building blocks of unstructured 3D meshes used in geophysical modeling and inversion. Each tetrahedron connects four nodes to form a 3D volume element.

### Format:

```
n_elements n_nodes_per_element n_attr
element_id node1 node2 node3 node4 [attributes]
element_id node1 node2 node3 node4 [attributes]
...
```

### Header Line:

- `n\_elements`: Total number of elements in the mesh
- `n\_nodes\_per\_element`: Number of nodes per element (typically `4` for tetrahedra in 3D, `3` for triangles in 2D)
- `n\_attr`: Number of attribute columns per element (can be 0 or more)

### Data Rows:

- `element\_id`: Unique identifier for the element (typically sequential integers starting from 1)
- `node1 node2 node3 node4`: Indices of the four nodes that form the tetrahedron (these must correspond to node IDs in the corresponding `.node` file)
- `[attributes]`: One or more attribute values associated with the element

### Tetrahedron Structure:

A tetrahedron is a 3D polyhedron with four triangular faces, four vertices (nodes), and six edges. In the element file, the four node indices define the vertices of the tetrahedron. The order of nodes is important for:

- Volume calculation: The volume is positive if nodes are ordered correctly (right-hand rule)
- Face orientation: Determines which side is "inside" vs "outside"
- Neighbor relationships: Used in `.neigh` files to identify adjacent elements

**Attributes** in element files typically represent:

- Physical properties: Material properties assigned to each element (e.g., magnetic susceptibility, density, electrical conductivity)
- Model parameters: Inverted property values from geophysical inversions
- Auxiliary data: Quality metrics, refinement indicators, or other element-associated quantities

**Example 1: Tetrahedral Mesh with Element Attributes:**

```

5000 4 1
1 1 2 3 4 0.01
2 5 6 7 8 0.02
3 2 9 10 11 0.015
...
```

This example shows 5000 tetrahedral elements, each with 1 attribute (e.g., magnetic susceptibility). Element 1 connects nodes 1, 2, 3, and 4 from the corresponding `.node` file and has a property value of 0.01.

**Example 2: Tetrahedral Mesh with Multiple Attributes:**

```

10000 4 3
1 1 2 3 4 0.01 2670.0 0.001
2 5 6 7 8 0.02 2700.0 0.002
...
```

This example shows 10000 tetrahedra, each with 3 attributes (e.g., magnetic susceptibility, density, electrical conductivity).

**Example 3: Mesh Without Element Attributes:**

```

3000 4 0
1 1 2 3 4
2 5 6 7 8
3 2 9 10 11
...
```

This example shows 3000 tetrahedra with only connectivity information, no attributes.

**Relationship with Node Files:**

- Element files must reference node IDs that exist in the corresponding `.node` file
- Node IDs in element files are 1-indexed (first node is ID 1, not 0)
- The same node can be shared by multiple elements (this is normal and expected)
- Elements define the volume discretization, while nodes define the spatial coordinates

**Notes:**

- Element IDs must be unique and typically start from 1
- Node indices must be valid (exist in the corresponding `.node` file)
- Elements should form a valid mesh (no overlapping volumes, proper connectivity)
- For visualization, elements are often converted to VTU format where they become "cells"

Besides the **.node** and **.ele** files, there are additional mesh file formats, such as **.neigh**, **.face**, and **.edge**, whose internal structures are not necessary to understand in order to use the software.

➤ **VTU FILES (.vtu):**

VTU (VTK Unstructured Grid) files are XML-based format for visualization.

- Can be opened in ParaView (download from [www.paraview.org](http://www.paraview.org)), or other VTK-based tools
- Contains coordinates, connectivity, and attributes
- Supports scalar and vector data

➤ **INPUT FILES (.inp):**

MAGNUM input files are text-based configuration files.

**Format:** Key-value pairs with quoted strings

```
key "value"
key "value"
...
```

**Example:** (`data.inp`)

```
datatype "gz"
datafile "/path/to/gravity_data.node"
chifact "1.0"
```

➤ **REGION BOUNDARY FILES (.txt):**

Region boundary files define polygonal boundaries for mesh generation and clipping.

**Format:**

```
Zmin Zmax
x1 y1 z1
x2 y2 z2
x3 y3 z3
...
```

**Notes:**

- First line defines vertical extent (Zmin, Zmax)
- Subsequent lines define polygon vertices (X, Y, Z)
- Z coordinate in vertices is ignored (only X-Y polygon is used)
- Polygon should have at least 3 points and is closed (last point connects to first)

## APPENDIX C: EXAMPLE

### PRACTICAL EXAMPLE: FORWARD MODELING OF GRAVITY DATA

This example demonstrates creating a workflow for forward modeling using FOGO.

#### Step 1: Add Input Programs

Double-click "Mesh", "Properties", and "Data" in the Category Panel to position them on the canvas.

#### Step 2: Configure Input Programs

Double-click each widget to configure parameters. Set mesh files, property files, and data files. Each program writes a file for FOGO.

#### Step 3: Add FOGO

Add "FOGO" from Category Panel and position it to the right of the input programs. Configure FOGO to specify the three input files (from Mesh, Properties, Data) and the output file.

#### Step 4: Create Connections

Drag from right dots of Mesh 01, Properties 01, and Data 01 to the left dot of FOGO 01. This creates three arrows converging on FOGO, indicating it needs all three inputs.

#### Step 5: Add Run/Stop Widget

Add "Run/Stop" and connect FOGO 01's right dot to Run/Stop's left dot.

#### Step 6: Run Workflow

Double-click Run/Stop widget and click "Run". ZAMINEX traces back to find all four programs. Mesh 01, Properties 01, and Data 01 run first (in parallel if configured). Once all complete, FOGO 01 runs automatically. Check Log Panel for execution messages.

#### What This Demonstrates

- Multiple inputs merge into one program
- Programs with no inputs are starting points and can run in parallel
- Programs wait for all dependencies before executing
- Run/Stop traces connections to identify complete workflow
- Visual organization (left-to-right flow) improves clarity

Save this workflow as a .gtxr file for reuse with different parameters or files.