

**RS3**

# **Automating Sublevel Stoping with CSV and Scripting**

Example Manual

# Table of Contents

|                                             |          |
|---------------------------------------------|----------|
| <b>1. Sublevel Stopping Scripting .....</b> | <b>3</b> |
| 1.1. Introduction.....                      | 3        |
| 1.2. Model Description.....                 | 3        |
| 1.3. Script Overview .....                  | 4        |
| 1.3.1. Import libraries .....               | 4        |
| 1.3.2. Define Model Parameters .....        | 4        |
| 1.3.3. Connect to RS3 .....                 | 5        |
| 1.3.4. Read CSV File .....                  | 5        |
| 1.3.5. Create Stages.....                   | 6        |
| 1.3.6. Find Model Volumes .....             | 6        |
| 1.3.7. Process Each Stage .....             | 7        |
| 1.3.8. Save the Model.....                  | 7        |
| 1.4. Discussion .....                       | 8        |

# 1. Sublevel Stoping Scripting

## 1.1. Introduction

Sublevel stoping analyses often require a large number of excavation and backfilling stages. As a result, manually assigning the sequential excavation and backfill stages is both time-consuming and prone to modelling errors. Furthermore, even a minor change to the mining schedule can affect multiple subsequent stages, requiring updates throughout the excavation sequence.

This example demonstrates how RS3 scripting eliminates this long process by using a single CSV file to define the excavation and backfill schedule. The CSV file serves as the controlling schedule for the model, allowing engineers to update the excavation sequence by modifying the input file rather than manually editing the staged model. The purpose of this workflow is to simplify model preparation, improve consistency, and provide a flexible method for evaluating different sublevel stoping schedules in RS3.

## 1.2. Model Description

The model represents an underground sublevel stoping operation consisting of a host rock mass and a series of production stopes distributed across multiple mining levels. Each stope is modeled as a separate volume to enable independent excavation and backfilling during the staged analysis. The initial model also includes the required material definitions for the rock mass, excavated regions, and backfill.

Figure 1-1 shows the initial model before applying the excavation sequence. The model contains the undeveloped rock mass, access ramp, production drifts, and stope volumes prior to excavation and backfilling.

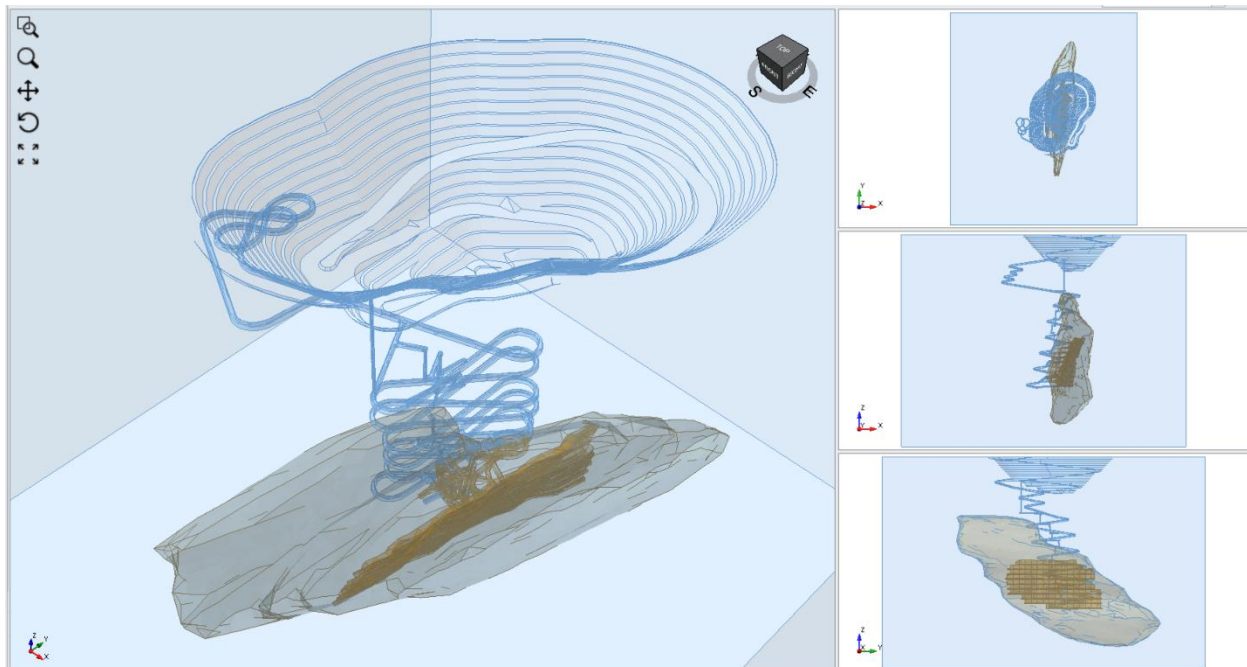


Figure 1-1: Initial Model

## 1.3. Script Overview

The script automates the generation of a staged sublevel stoping excavation sequence using a CSV file as the input schedule. The workflow consists of two main parts. The first part processes the CSV file by reading the excavation schedule, creating the analysis stages, and storing the excavation and backfill actions for each stope. The second part processes the RS3 model by locating the corresponding stope volumes, matching them with the CSV data, and updating the material assignments at each analysis stage. Together, these two workflows eliminate the need to manually create stages and assign excavation or backfill materials, allowing the excavation sequence to be controlled entirely through the input CSV file. The following sections describe each step of the script in detail.

The initial model contains pre-divided volumetric pieces representing the underground developments and ore zones. In this exercise, the appropriate material properties will be assigned to the stopes, drifts, and ramps throughout the mining sequence, with excavation and backfilling performed at the intended timeframes represented by the RS3 modelling stages.

The mining sequence is defined in an input CSV file, with the header row specifying the analysis stage names and each subsequent row defining the excavation (EXC) or backfill (BACKFILL) action for each stope. This structure allows the excavation schedule to be updated without modifying the script. Figure 1-2 illustrates the stage creation workflow used in the model.

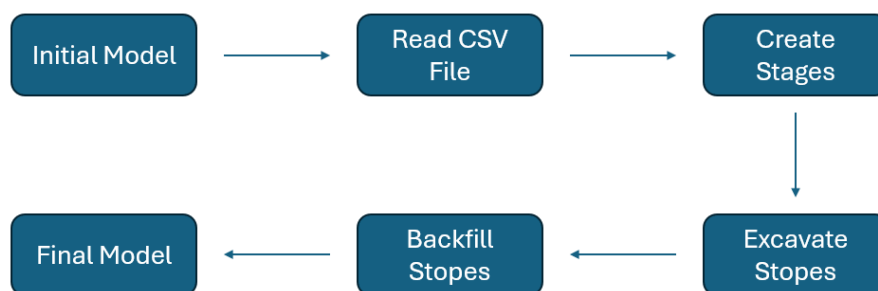


Figure 1-2: Workflow Diagram

### 1.3.1. Import libraries

Import the required modules from RS3 Python API library for connecting to the module, selecting geometry, and modifying model properties.

```
from rs3.RS3Modeler import RS3Modeler
import os
import csv
```

### 1.3.2. Define Model Parameters

Before running the script, ensure that RocScript is opened with the working directory set to the folder containing both the script and the RS3 model. The script uses the current working directory to locate the input CSV file, so opening RocScript from a different location may result in invalid file paths. Once the

working directory has been set correctly, the script can locate the required input files and execute as intended.

The model location is specified under `model_path`. The excel file will open from the same folder directory.

```
# Get the current folder path
current_dir = os.path.abspath(os.path.dirname(__file__))

# Input both model and csv file path here
model_path = rf"{current_dir}\sublevel-stoping-initial.rs3v3"
csv_path = rf"{current_dir}\Stope Excavation Sequence.csv"
```

### 1.3.3. Connect to RS3

Choose an available port in the program and put it under `port`. The script connects with the RS3 Modeler through the same port number specified earlier and opens the model from the model path.

```
# Set the port number and connect to RS3 modeler
port = 60064
modeler = RS3Modeler(port)

# Open the RS3 model
model = modeler.openFile(model_path)
```

### 1.3.4. Read CSV File

The excavation schedule is read directly from the input CSV file. The CSV file is structured with the first column containing the stope volume names and the following columns representing the stages of mining sequence. These stages will be directly used for the modeling stages. The header row defines the stage names used for the staged analysis, while each cell under a stage indicates the required action for that stope, either excavation (EXC) or backfilling (BACKFILL).

The script reads this information and organizes it into a “volume\_action” dictionary that stores the required action for each stope at each analysis stage. The volume\_action dictionary will consists of elements with keys being the name of each volume and store another dictionary holding actions between excavation or backfill for the respective stage index (key) (e.g. {“Stope-318-6”: {3: “EXC”, 4: “BACKFILL”}, “Stope-318-11”: {4: EXC, 5: BACKFILL}}). This data structure allows the script to identify which volumes require excavation or backfilling for different stages.

```
# Read the CSV file
with open(csv_path, 'r') as f:
    reader = csv.reader(f)
    rows = list(reader)

# Row 2 (index 1) contains stage names: "", "initial", "2027-1", "2027-2", ...
stage_row = rows[1]
stage_names = [s.strip() for s in stage_row[1:] if s.strip()]
```

```
# Parse volume actions: {volume_name: {stage_idx: action}}
# stage_idx is 0-based (0 = first stage "initial")
volume_actions = {}

for row in rows[2:]:
    if len(row) >= 1 and row[0].strip():
        volume_name = row[0].strip()
        volume_actions[volume_name] = {}

        # Convert the schedule to a dictionary
        for stage_num, action in enumerate(row[1:], start=1):
            action_upper = action.strip().upper()
            if action_upper == "EXC" or action_upper == "BACKFILL":
                volume_actions[volume_name][stage_num] = action_upper
```

### 1.3.5. Create Stages

Using the total number of stage names in the CSV file, the script creates the required number of analysis stages in RS3 and assigns the corresponding stage names. This ensures that the staged analysis follows the sequence specified in the input schedule.

```
# Create the same number of stages as the schedule listed
total_stages = len(stage_names)

stages = model.ProjectSettings.Stages
stages.setTotalNumberOfStages(total_stages)

# Update the stage names corresponding to the schedule
for stage_num, stage_name in enumerate(stage_names, start=1):
    stages.setName(stage_num, stage_name)
```

### 1.3.6. Find Model Volumes

This section identifies all stoep volumes within the model and organizes them into a dictionary, "volume\_dict", using their volume names and the respective volume objects as keys and values, respectively. This allows fast matching between the volumes defined in the CSV file and the corresponding geometry in the RS3 model.

```
# Build dictionary of volumes by name
volume_dict = {}
for vol in model.getAllExternalVolumes():
    vol_name = vol.getName()
    # Since External will not be changed
    if vol_name != "External":
        volume_dict[vol_name] = vol
```

### 1.3.7. Process Each Stage

The script processes each analysis stage defined in the CSV file and uses the information stored in the two dictionaries created previously. The “volume\_actions” dictionary contains the excavation schedule from the CSV file, where each stoep name is linked to its assigned action (EXC or BACKFILL) at each stage. The “volume\_dict” dictionary contains the corresponding external volume objects from the RS3 model, where each volume name is linked to its geometry.

During stage processing, the script compares the volume names stored in both dictionaries to match each CSV entry with the corresponding RS3 volume. Once a match is found, the assigned action from the CSV file is converted into the corresponding RS3 material property using the material\_dict dictionary. Volumes marked as EXC are assigned the *No Material* property to represent excavation, while volumes marked as BACKFILL are assigned the *BackFill* material. This process is repeated for each analysis stage until the complete mining sequence has been generated.

The active stage update is included for demonstration purposes to allow users to view the progression of the excavation sequence during script execution and is not required for the material assignment process.

```
# Create a dictionary that links actions to material properties
material_dict = {"EXC": "No Material", "BACKFILL": "BackFill"}
for stage_num, stage_name in enumerate(stage_names, start=1):
    # Find all volumes with actions at this stage
    for volume_name, actions in volume_actions.items():
        if stage_num in actions:
            action = actions[stage_num]

            if volume_name in volume_dict:
                # Get the volume object from the dictionary
                volume = volume_dict[volume_name]

                # Set the corresponding material based on the schedule
                volume.setAppliedMaterialProperty(stage_num, material_dict[action])

    # Set active stage
    model.setActiveStage(stage_num)
```

### 1.3.8. Save the Model

After all stages have been processed, the completed model can then be saved, producing an RS3 project that contains the full staged excavation and backfilling sequence defined by the CSV file.

```
# Save the model and close it
model.saveAs(rf"{current_dir}\sublevel-stoping-final.rs3v3")
model.close(False)
```

## 1.4. Discussion

This example demonstrates how RS3 scripting can be used to automate a sublevel stoping excavation sequence by generating analysis stages and updating material assignments from an external CSV file. Instead of manually creating each stage and assigning excavation or backfill materials, the script reads the predefined mining schedule and applies the corresponding changes automatically.

In this workflow, the CSV file acts as the controller of the excavation sequence, allowing the mining schedule to be modified independently from the scripting logic. Different mining scenarios can be evaluated by updating the CSV file while using the same script, eliminating the need to modify the code for each sequence. This provides a flexible approach for generating alternative excavation strategies and automating repetitive staged analyses in RS3.