

RS3

Automating Cut and Fill Mining Sequence Application with Scripting

Example Manual

Table of Contents

1. Cut and Fill Scripting	3
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 Path and Connect to RS3	4
1.3.3. Define Model Parameters.....	5
1.3.4. Identify and Rename Stopes	5
1.3.5. Identify Development Excavations	5
1.3.6. Define Helper Function	6
1.3.7. Generate Mining Sequence Plan.....	7
1.3.8. Apply Stage Plan to RS3 Model	8
1.3.9. Save the Model.....	9
1.4. Discussion	9

1. Cut and Fill Scripting

1.1. Introduction

This example shows how RS3 scripting can be used to automate a cut and fill mining operation. The script renames the stopes based on their elevation, creates excavation stages, excavates and backfills all stopes at a given elevation, and repeats the process for each elevation.

1.2. Model Description

The model represents an underground cut-and-fill mining operation consisting of a host rock mass, development excavations, and stopes distributed across nine mining levels. The development excavations include a main access ramp and nine production drifts. The main ramp provides access between all mining levels, while each production drift extends from the ramp to provide access to the surrounding stopes at its corresponding elevation.

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.

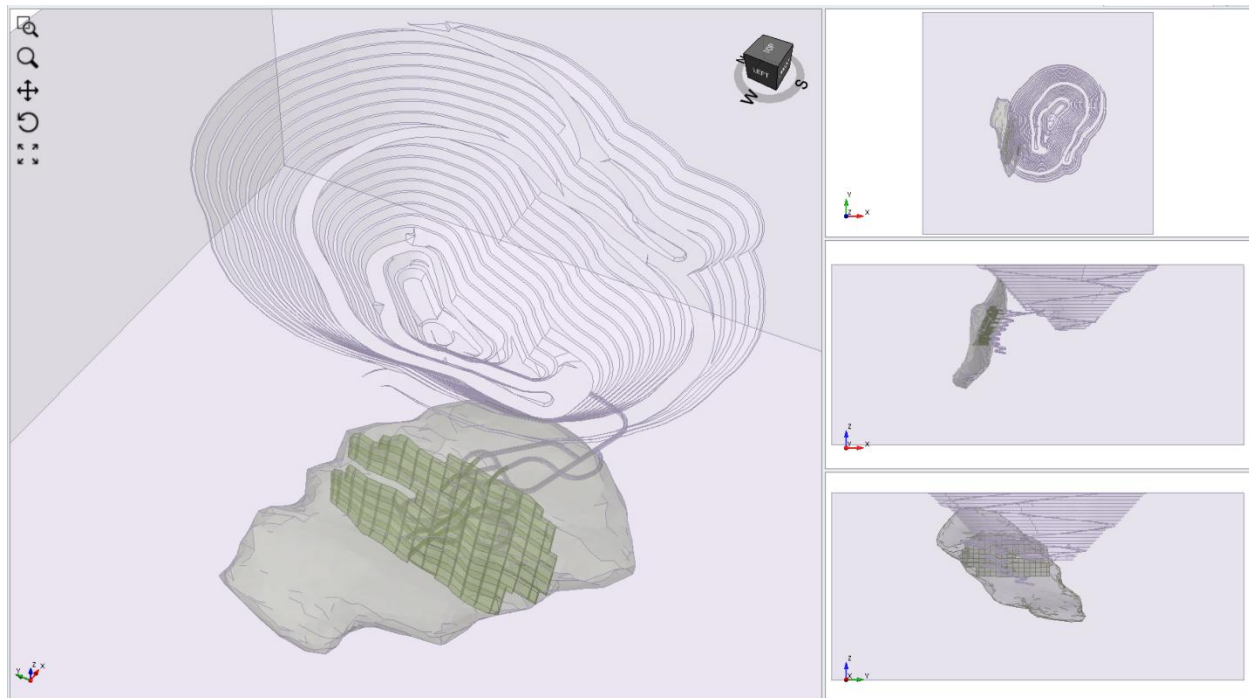


Figure 1-1: Initial Model

The objective of the scripting in this example is to automate the multi-stage excavation and backfilling sequence for this cut-and-fill layout. Starting from the prepared initial model, the script identifies the development excavations and stope volumes, organizes them by mining level, and builds a complete stage plan that progresses from the lowest elevation to the highest.

At each mining level, the script excavates the corresponding production drift, then excavates connected stopes in successive stages based on shared boundaries between volumes. After all stopes at that elevation have been excavated, they are assigned backfill material before the sequence advances to the next level. Figure 1-2 illustrates the stage creation workflow used in the model.

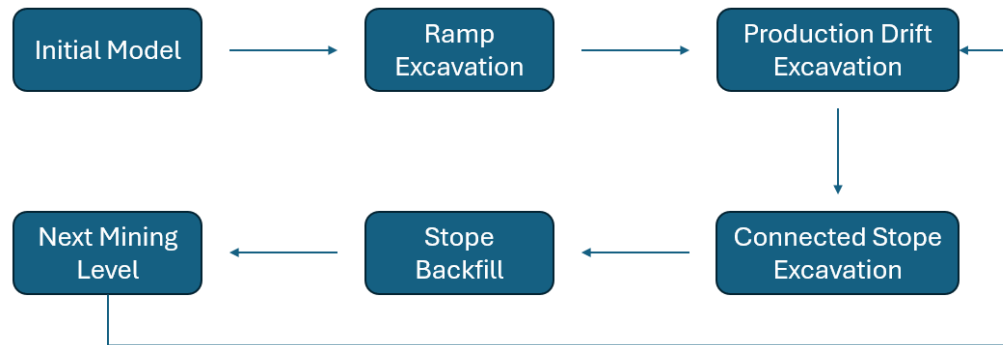


Figure 1-2: Workflow Diagram

1.3. Script Overview

The script is organized into a series of sections that initialize the required parameters, retrieve model entities, and automate the creation of excavation and backfilling stages. Functions are used to simplify repetitive tasks, while the main execution loop processes each drift according to the specified mining sequence. The following sections describe the purpose and implementation of each part of the script.

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. The os library is also imported to support file path handling and script information retrieval.

```

from rs3.RS3Modeler import RS3Modeler
from rs3.properties.PropertyEnums import *
from rs3.Geometry import Cube, Point
import os

```

1.3.2. Define Model Path and Connect to RS3

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 model path is automatically generated based on the location of the Python script. The RS3Modeler object is then created using the specified port and used to open the RS3 model.

```

current_dir = os.path.abspath(os.path.dirname(__file__))

model_path = rf"{current_dir}\cut-and-fill-initial.rs3v3"

port = 60064

```

```
modeler = RS3Modeler(port)
model = modeler.openFile(model_path)
```

1.3.3. Define Model Parameters

Specify the bounding box limits and mining elevations used throughout the script. These parameters define the regions used to identify stopes, drifts, and development excavations.

```
# Define the X, Y, Z boundaries
X_min, X_max = 4650, 4800
Y_min, Y_max = 3830, 4220
Z_elevations = [33, 53, 73, 93, 113, 133, 153, 173, 193]
```

1.3.4. Identify and Rename Stopes

The script identifies stope volumes at each mining elevation using a bounding box. Volumes named *Convex Hull* are selected and renamed according to their elevation and position. The renamed stopes are stored in a dictionary for later access during the excavation sequence.

```
# Rename the stopes at each elevation and store them in a dictionary, all_stopess.
# Their names are formatted as "Stope_<elevation>_<id>".
# Its keys match the assigned name and value as a corresponding stope volume.
all_stopess = {}
for Z_min in Z_elevations:
    # Get the stope objects at each elevation
    bounding_box = Cube(Point(X_min, Y_min, Z_min), Point(X_max, Y_max, Z_min + 2.0))
    convex_hulls = [v for v in model.Selection.getExternalVolumes(bounding_box)
                     if v.getName().startswith("Convex Hull")]

    for i, stope in enumerate(convex_hulls, start=1):
        # Rename all the stopes
        name = f"Stope_{Z_min}_{i}"
        stope.setName(name)

        # Store stopes in a dictionary
        all_stopess[name] = stope
```

1.3.5. Identify Development Excavations

Two data structures are then initialized to store the identified development excavations. The `ramp_volumes` list stores all main ramp volumes, while the `drift_map` dictionary groups drift volumes according to their drift number.

```
# Collect ramp and drift volumes in ramp_volumes array and drift_map dictionary, respectively.
ramp_volumes = []
drift_map = {} # {drift_num: [volumes]}
```

The script then searches all external volumes and separates the development excavations into two groups. Main ramp volumes are stored separately, while drift volumes are organized according to their drift number for use during stage generation.

```
for v in model.getAllExternalVolumes():
    name = v.getName()
    if name.startswith("Main Ramp"):
        ramp_volumes.append(v)
    elif name.startswith("Drift-"):
        try:
            # Names of drifts follows the format "Drift-<level id>_<subnumber>", e.g., "Drift-1_468", "Drift-2_475", etc.
            # We're extracting the level id as an integer to use as a key in the drift_map dictionary.
            drift_num = int(name.split("-")[1].split("_")[0])
            # store the drifts (within the ore and footwall) in a list under the corresponding drift_num key in the drift_map dictionary.
            drift_map.setdefault(drift_num, []).append(v)
        except ValueError:
            pass
```

1.3.6. Define Helper Function

A helper function is defined to identify unexcavated stopes connected to a given set of volumes. This function is used to determine the next group of stopes to excavate at each mining level based on shared boundaries between volumes.

The `stope_frontier()` function finds stopes connected to the current excavation area. Only stopes at the current elevation that have not been excavated are returned.

```
def stope_frontier(source_vols, elevation, excavated):
    """
    Identify unexcavated stopes connected to a set of volumes at a specific elevation.
    It returns a set of names of unexcavated stopes attached to any volume in source_vols.
    """
    return {
        av.getName()
        for v in source_vols
        for av in model.Selection.getAttachedExternalVolumes(v)
        if av.getName().startswith(f"Stope_{elevation}_")
        and av.getName() not in excavated
    }
```

1.3.7. Generate Mining Sequence Plan

Instead of modifying the RS3 model immediately, the script first creates a list of excavation and backfill operations. Each planned stage contains a stage name and the volumes with their assigned material properties. This separates the process of generating the mining sequence from applying the changes to the RS3 model.

The organized volume data from the previous steps is used to build the stage plan. The `ramp_volumes` list provides the volumes for the initial ramp excavation, while `drift_map` and `all_stopes` provide access to the production drifts and stopes. The script begins by adding the ramp excavation stage to the plan and then processes each mining elevation sequentially.

For each mining level, the corresponding drift volumes are retrieved directly from `drift_map`. Because the drifts are already named with the mining level included, all drift pieces for that level can be collected without additional renaming. These volumes are then added as a drift excavation stage.

After the drift excavation stage is created, `stope_frontier()` identifies stopes connected to the excavated drift. A while loop continues the excavation sequence by finding additional connected stopes until no unexcavated stopes remain at that elevation. The names of excavated stopes are stored in `excavated_stopes` to prevent them from being included again. Once all stopes at the current elevation have been added to the excavation sequence, a backfill stage is created by assigning the `BackFill` material property to those stopes.

```
# No model changes are made at this point; the plan list is constructed.
# The stage_plans list will be used to store the excavation and backfill stages for the ramp,
drifts, and stopes.
# Each entry in the list is a tuple containing the stage name and a list of (volume, material
property name) pairs.
# The material property name is set to "No Material" for excavation stages and "BackFill" for
backfill stages.
# For its first entry, the ramp excavation stage is added to the stage_plans list, with all
ramp volumes assigned "No Material".
stage_plans = [("RampExcavation", [(v, "No Material") for v in ramp_volumes])]
# The excavated_stopes set is initialized to keep track of the names of stopes that have been
excavated.
excavated_stopes = set()
for drift_num, elevation in enumerate(Z_elevations, start=1):
    # Collect all drift volumes at the given elevation
    drift_volumes = drift_map.get(drift_num, [])
    # Update drift excavation stage plan
    stage_plans.append((
        f"Drift_{drift_num}_Excavation",
        [(v, "No Material") for v in drift_volumes]
    ))

    # Collect the names of all unexcavated stopes at the given elevation connected to the
drift_volumes
    frontier = stope_frontier(drift_volumes, elevation, excavated_stopes)
```

```

if not frontier:
    print(f" No stopes connected to Drift-{drift_num}, skipping...")
    continue

# Initialize a list to keep track of the stopes excavated at this elevation
stopes_this_elevation = []

# Update the stope excavation stage plan for the current elevation, iterating through the
frontier of unexcavated stopes connected to the volumes, initially the drift_volumes.
# The frontier is updated after each batch of stopes is excavated, until no unexcavated
stopes remain connected to the drift_volumes.
while frontier:
    sorted_names = sorted(frontier)
    stope_ids = [n.split("_")[-1] for n in sorted_names]
    label = "_".join(stope_ids[:5]) + ("_etc" if len(stope_ids) > 5 else "")

    # Adds a new stage plan for the excavation of the stopes in the current batch, with
the stage name reflecting the stope numbers included in that batch.
    # E.g. "Drift-1-Stopes_1_2"
    stage_plans.append((
        f"Drift-{drift_num}-Stopes_{label}",
        [(all_stopes[n], "No Material") for n in sorted_names if n in all_stopes],
    ))
    excavated_stopes.update(sorted_names)
    stopes_this_elevation.extend(sorted_names)

    # Update unexcavated stopes list by looking for the external volumes attached to the
stopes in the current batch,
    # filtering for those that are unexcavated and at the same elevation.
    frontier = stope_frontier(
        [all_stopes[n] for n in sorted_names if n in all_stopes],
        elevation, excavated_stopes,
    )

# Update stope backfill stage plan after all excavations
stage_plans.append((
    f"Drift-{drift_num}-Backfill",
    [(all_stopes[n], "BackFill") for n in stopes_this_elevation if n in all_stopes],
))

```

1.3.8. Apply Stage Plan to RS3 Model

After generating the complete mining sequence, the script determines the total number of stages required and updates the RS3 model accordingly. Since the initial model already contains Stage 1, the total number of stages is equal to one plus the number of planned excavation and backfill stages.

```
total_stages = 1 + len(stage_plans) # model already has stage 1
stages = model.ProjectSettings.Stages
stages.setTotalNumberOfStages(total_stages)
```

The script then iterates through the stage plan, activates each stage, assigns its name, and applies the corresponding material properties to the associated volumes.

```
for stage_num, (name, assignments) in enumerate(stage_plans, start=2):
    model.setActiveStage(stage_num)
    stages.setName(stage_num, name)

    # Update the material assignment of each volume at each stage
    for volume, material in assignments:
        volume.setAppliedMaterialProperty(stage_num, material)

    print(f" Stage {stage_num}: {name} ({len(assignments)} volume(s))")
```

1.3.9. Save the Model

After the mining sequence has been applied, the completed model is saved to a new file and then closed.

```
model.saveAs(rf"{current_dir}\cut-and-fill-final.rs3v3")
model.close(False)
```

1.4. Discussion

This example demonstrates how RS3 scripting can be used to automate a cut-and-fill mining sequence by controlling excavation stages and material assignments. Instead of manually creating each excavation and backfilling stage, the script automatically generates the complete mining sequence based on the model geometry and the shared boundaries between stopes.

The script uses volume selection and shared boundaries between stopes to determine the excavation order at each mining level. By organizing the stopes by elevation and tracking previously excavated volumes, the script ensures that each mining level is excavated and backfilled before progressing to the next level.

The workflow can be adapted for different mining layouts by modifying the model geometry, mining elevations, or stage generation logic. This provides a flexible approach for automating repetitive excavation sequences and reducing manual stage creation in RS3.