VonSourceTools Technical Specifications

Document Information

- **Project Name**: Vona's Blender Source Tools (VonSourceTools)
- **Version**: 0.0.2 (Development)
- **Author**: Vona
- **Target Platform**: Blender 4.2.10+
- **Programming Language**: Python 3.x
- **Purpose**: Streamline Blender to Source/Garry's Mod workflow

---

1. Executive Summary

VonSourceTools is a Blender addon designed to significantly improve the workflow for creating Source Engine and Garry's Mod content. The addon automates the generation of QC files, manages complex armature transformations through the delta animation trick, and handles material/texture conversions between Blender and Source formats.

Key Features

- Automated QC file generation with sequences, bodygroups, and material folders
- Delta Animation Trick automation for Garry's Mod player models
- VMT/VTF file management and generation
- Batch image conversion (TGA, PNG, VTF)
- Jigglebone support
- Integration with existing Blender Source Tools

---

2. System Architecture

2.1 Module Structure

```
VonSourceTools/
├── __init__.py                    # Addon registration and metadata
├── von_common.py                  # Shared utilities and data structures
├── von_ui.py                      # UI panel definitions
├── von_ui_operators.py            # Operator implementations
├── von_qcbuilder.py               # QC file generation logic
├── von_deltaanimtrick.py          # Delta animation trick automation
├── von_batchvtfconversions.py     # Image/texture batch conversion
├── von_armaturenamingstandardization.py # Armature naming utilities (planned)
├── credits.txt                    # Attribution and credits
└── storeditems/
    └── qcgenerator/
        └── templates/
            ├── commands/          # QC command templates
            ├── qc_section_order.json  # QC command ordering
            └── surfaceprops.json      # Surface property definitions
```

2.2 Core Dependencies

- **Blender Python API (bpy)**: Core Blender functionality
- **Standard Library**: json, pathlib, collections, os, subprocess
- **mathutils**: Vector operations for bone manipulation
- **bmesh**: Mesh data manipulation (future features)

2.3 Integration Points

- **Blender Source Tools**: Expected to be installed alongside VonSourceTools
- **External Tools**: VTFCmd/VTFEdit for texture conversions (via subprocess)
- **Crowbar**: Decompilation support (future integration)

---

3. Detailed Feature Specifications

3.1 QC Generator System

3.1.1 Overview

Generates complete, valid QC files for Source Engine model compilation based on Blender scene data.

3.1.2 Supported QC Commands

Core Commands:

- `$modelname` - Model output path
- `$cdmaterials` - Material folder paths
- `$include` - Include external QC files
- `$body` / `$bodygroup` - Model bodygroups
- `$sequence` - Animation sequences
- `$collisionmodel` - Physics collision mesh

Advanced Commands:

- `$surfaceprop` - Physics surface properties
- `$attachment` - Bone attachment points
- `$hboxset` - Hitbox definitions
- `$illumposition` - Illumination center
- `$origin` - Model origin offset
- `$includemodel` - Include external models
- `$staticprop` - Static prop optimization
- `$jigglebone` - Jiggle bone physics

3.1.3 Model Types

The system supports multiple model type categories:

```python
MODEL_TYPE_CATEGORY_MAP = {
    "NPC": [
        "Basic", "Movement", "Combat", "Gestures",
        "NPC_Reactions", "Signals_Commands", "Interaction",
        "Swimming", "Vehicles", "Scripted_Sequences"
    ],
    "CHARACTER": [
        "Basic", "Movement", "Combat", "Gestures",
        "NPC_Reactions", "Signals_Commands", "Interaction",
        "Swimming", "Vehicles", "Scripted_Sequences"
    ],
    "VIEWMODEL": ["Viewmodel"],
    "PROP": [],
    "WORLDMODEL": []
}
```

3.1.4 Data Structures

QC_PrimaryData (Scene Property):

- `vmt_filepaths`: Collection of VMT file paths
- `bodygroup_boxes`: Bodygroup definitions with associated collections
- `sequence_objectdata`: Per-armature sequence export data

Sequence Export Data Structure:

```python
{
  armatureName: {
    originalSequenceName: {
      "sequenceName": str,
      "shouldExport": bool,
      "qcPath": str,
      "customTag": str,
      "activityCategory": str,  # Enum value
      "activity": str           # Activity name
    }
  }
}
```

Bodygroup Data Structure:

```python
{
  "BodygroupName": [
    "CollectionName1",
    "CollectionName2",
    ...
  ]
}
```

3.1.5 Template System

QC commands are generated using a template-based system:

1. Templates stored in `storeditems/qcgenerator/templates/commands/`
2. Each command has a corresponding template file
3. Templates use Python format strings with named placeholders
4. `populate_template()` function handles template loading and population

Example Template Flow:

```python
populate_template(
    "modelname.template",
    replacements={"mdlModelName": "models/player/mymodel.mdl"}
)
```

3.1.6 QC Generation Workflow

1. **Data Collection Phase**:
    - Gather bodygroup data from scene collections
    - Collect sequence data from armature NLA tracks
    - Retrieve VMT file paths
    - Get surface property selection
2. **Validation Phase**:
    - Verify all commands are valid for selected model type
    - Check command order against `qc_section_order.json`
    - Report invalid commands
3. **Generation Phase**:
    - Write commands in proper order
    - Populate templates with collected data
    - Handle special cases (multiple materials, bodygroups, etc.)
4. **Export Phase**:
    - Write final QC file to specified output path
    - Generate associated SMD files
    - Create material folder structure

3.1.7 Future Enhancements

- Automatic collision mesh generation from simplified geometry
- Attachment point detection from empties/bones
- Hitbox generation from mesh bounds
- Template customization UI
- Multi-QC export for LOD models

3.2 Delta Animation Trick System

3.2.1 Overview

Automates the "proportion trick" technique for creating Garry's Mod player models with custom proportions while maintaining Source Engine compatibility.

3.2.2 Reference Implementation

Based on: https://github.com/sksh70/proportion_trick_script (Blender 2.9 version)

3.2.3 Technical Background

The delta animation trick allows custom-proportioned models to use Garry's Mod's player animation system by:

1. Creating a "proportions" armature that maintains Valve Biped skeleton structure
2. Constraining the proportions armature to match custom character proportions
3. Applying constraints as rest pose
4. Merging non-ValveBiped bones from custom rig to proportions armature
5. Transferring mesh skinning to proportions armature

3.2.4 Required Armatures

The system requires three reference armatures:

1. **proportions**: Modified Valve Biped armature (main output)
2. **reference_male**: Standard male proportions reference
3. **reference_female**: Standard female proportions reference

Storage Location: Stored in external blend file, imported on-demand **Import Function**:
`von_common.importitemfromdict()`

3.2.5 Valve Biped Bone List

The system maintains two hardcoded bone lists:

valvebipeds_1 (52 bones):

- Core deformation bones (pelvis, spine, limbs, fingers, toes)
- Used for constraint application and validation

valvebipeds_2 (46 bones):

- Bone pairs for locked track constraints
- Format: [TargetBone, SubtargetBone, TargetBone, SubtargetBone, ...]

3.2.6 Process Flow

Part One: Constraint Setup

```
python

def delta_anim_trick_one(sourceArmature: bpy.types.Object):
```

1. **Validation**:
    - Verify 'proportions' armature exists in scene
    - Check both source and target are ARMATURE type
    - Validate bone existence
2. **Copy Location Constraints**:
    - Add to all ValveBiped bones in proportions armature
    - Target: Source armature
    - Subtarget: Matching bone name
3. **Locked Track Constraints**:
    - Add two constraints per bone in targetBones list
    - Constraint 1: Track X axis, Lock Z axis
    - Constraint 2: Track X axis, Lock Y axis
    - Subtarget: Corresponding bone from subBones list
4. **Constraint Cleanup**:
    - Remove non-COPY_LOCATION constraints from bones with no constrained children
    - Prevents over-constraining leaf bones
5. **Finalization**:
    - Hide source armature
    - Show proportions armature
    - Enter POSE mode on proportions armature

Part Two: Merge and Transfer

```
python

def delta_anim_trick_two(imported_name="gg", proportions_name="proportions"):
```

1. **Safety Checks**:
    - Verify both armatures exist
    - Confirm ARMATURE type
    - Get ValveBiped bone list
2. **Armature Duplication**:
    - Create data-level copy of source armature
    - Link to current collection
3. **Bone Merging** (EDIT mode):
    - Enter edit mode on proportions armature
    - For each non-ValveBiped bone in source:
        - Create new bone in proportions armature
        - Copy head, tail, roll from source
        - Parent to matching bone (default: ValveBiped.Bip01_Pelvis)
4. **Mesh Transfer**:
    - Find all MESH objects in scene
    - Update/add Armature modifier
    - Set target to proportions armature
5. **Cleanup**:
    - Remove temporary duplicate armature
    - Restore previous mode

Full Automation Operator:

```
python

class VonPanel_DeltaAnimTrick_Full(bpy.types.Operator):
```

Combines both parts with additional features:

1. **Multi-Armature Support**: Processes all selected armatures
2. **Validation**: Checks similarity threshold for Valve Biped structure
3. **Toe Adjustment**: Calls `toevertical()` to align toe bones
4. **Constraint Application**: Applies pose as rest using `bpy.ops.pose.armature_apply()`
5. **Constraint Cleanup**: Removes all constraints after application

3.2.7 Helper Functions

Validation:

```python
def is_valve_biped(armature, context) -> bool:
```

- Counts matching ValveBiped bones
- Calculates match percentage
- Compares against user-defined threshold
- Returns True if threshold met

Toe Bone Adjustment:

```python
def toevertical(bone):
```

- Sets toe bone tail to vertical above head
- X and Y match head position
- Prevents toe bone rotation issues

Constraint Removal:

```python
def clearposeboneconstraints(bone, armature):
```

- Requires POSE mode
- Removes all constraints from specified bone
- Used after applying constraints as rest pose

Reference Import:

```python
def has_properties() -> tuple[bool, bool, bool]:
```

- Checks for required reference armatures
- Imports missing armatures
- Re-imports 'proportions' if already exists (ensures clean state)
- Returns availability flags

3.2.8 User-Configurable Parameters

Similarity Threshold:

- Property: `toolBox.float_deltaAnim_simmilarityThreshold`
- Default: TBD (likely 70-80%)
- Purpose: Determines minimum percentage of ValveBiped bones required
- Used by: `is_valve_biped()` validation

3.2.9 Error Handling

Common Errors:

1. Missing 'proportions' armature → Import automatically
2. Invalid armature type → Exception with type info
3. Below similarity threshold → Report with threshold value and failed armatures
4. Wrong object mode → Exception in constraint removal

Error Messages:

```
python

# Missing armature
"No armature named 'proportions' found in the scene."

# Type validation
"Target armature 'proportions' must be an ARMATURE, not {type}"

# Similarity failure
"{armature.name} has failed to meet the current similarity threshold of {threshold}.
Please either lower threshold or ensure the armature uses valve biped armature on
core deformation bones."
```

3.2.10 UI Integration

Simple Panel:

- Single button: "Delta Anim Trick (Full)"
- Operator: `von.deltaanimtrick_full`
- Icon: 'PLAY'

Advanced Panel (Collapsed by default):

- Import Reference Armatures button
- Part One button (constraint setup)
- Part Two button (merge and transfer)
- Similarity threshold slider
- Individual control for debugging/custom workflows

3.2.11 Known Limitations and Future Work

Current Limitations:

1. Toe bone check has duplicate condition (Line 166, 186)
2. Hard-coded armature names ('proportions', 'gg')
3. No undo support for full automation
4. Limited error recovery

Planned Improvements:

1. Customizable reference armature names
2. Save/load proportion presets
3. Batch processing for multiple characters
4. Integration with weight painting tools
5. Automatic IK bone generation
6. Face rig preservation options

3.3 VMT/VTF Material System

3.3.1 Overview

Manages Source Engine material files (VMT) and Valve Texture Format (VTF) files, with batch conversion capabilities.

3.3.2 VMT File Generation

Supported Shader Types (Future Implementation):

- `VertexLitGeneric` - Standard world materials
- `LightmappedGeneric` - Lightmapped world materials
- `UnlitGeneric` - Unlit materials (sprites, overlays)
- `Skin` - Character skin shader
- `Eyes` - Eye shader with parallax
- `Teeth` - Specialized teeth shader

VMT Parameters (Common):

```
$basetexture "path/to/diffuse"
$bumpmap "path/to/normal"
$phong 1
$phongexponent 20
$phongboost 0.5
$phongtint "[1 1 1]"
$rimlight 1
$rimlightexponent 2
$rimlightboost 1
$halflambert 1
$nocull 1
$alphatest 1
$translucent 1
```

3.3.3 VTF Batch Conversion

Supported Input Formats:

- PNG
- TGA
- BMP
- JPG
- TIFF

Supported Output Formats:

- VTF (Valve Texture Format)
- TGA (Targa)
- PNG (Portable Network Graphics)

Conversion Module: `von_batchvtfconversions.py`

```python
def batch_convert(input_folder, output_folder, source_type, target_type):
```

External Tool Integration:

- Uses VTFCmd for VTF ↔ other format conversion
- Subprocess calls to command-line tools
- Error handling for missing tools

UI Properties:

- `string_vtfbatch_inputfolder`: Source folder path
- `string_vtfbatch_outputfolder`: Destination folder path
- `enum_vtfbatch_sourcefiletype`: Input format selection
- `enum_vtfbatch_targetfiletype`: Output format selection

Performance Considerations:

- Warning: "Blender might hang if converting to or from VTF"
- Note: "May take a few minutes"
- Processes all files in folder recursively
- No progress feedback (blocking operation)

3.3.4 Material Path Management

VMT Filepath Collection:

```python
def get_all_vmt_filepaths() -> list:
```

- Retrieves from `scene.QC_PrimaryData.vmt_filepaths`
- Each item has `filepath` property
- Filters out empty paths
- Returns list of valid filepath strings

QC Integration:

- VMT paths added to `$cdmaterials` commands
- Supports multiple material folders
- Relative path handling

3.3.5 Future Enhancements

- Auto-generate VMT from Blender materials
- Material property mapping (Principled BSDF → Source shader)
- Texture baking for unsupported features
- Normal map conversion (OpenGL ↔ DirectX)
- VTF compression settings UI
- Mipmap generation control
- Texture atlas support

3.4 Batch SMD Export

3.4.1 Overview

Handles batch export of Static Mesh Data (SMD) files for Source Engine model compilation.

3.4.2 Collection Management

Split Objects Operator:

```
python

class OBJECT_OT_split_objects(bpy.types.Operator):
```

- Organizes scene objects into collections
- Prepares for separate SMD exports
- Likely groups by bodygroup or LOD

Restore Objects Operator:

```
python

class OBJECT_OT_restore_objects(bpy.types.Operator):
```

- Reverts collection organization
- Restores original scene structure
- Cleanup after export

3.4.3 Export Process

Export Operator:

```
python

class OBJECT_OT_export_smd(bpy.types.Operator):
```

- Exports collections as individual SMD files
- Uses Blender Source Tools export functionality
- Configurable output folder

UI Properties:

- `string_export_folder`: Export destination path

3.4.4 Integration with QC Generator

- Exported SMD files referenced in QC bodygroups
- Sequence SMD files for animations
- Collision SMD for physics

3.5 Jigglebone System (Planned)

3.5.1 Overview

Automated generation of jigglebone QC commands for physics-based secondary motion (hair, cloth, accessories).

3.5.2 QC Jigglebone Syntax

```
$jigglebone "bone_name" {
  is_flexible {
    length 10
    tip_mass 50
    pitch_stiffness 100
    pitch_damping 5
    yaw_stiffness 100
    yaw_damping 5
  }
}
```

3.5.3 Blender Integration (Proposed)

Bone Tagging:

- Custom bone properties to mark jigglebone
- UI panel for jigglebone parameters
- Per-bone physics settings

Parameter Mapping:

- Bone length → `length`
- Custom properties → stiffness/damping values
- Bone constraints → physics limits

Validation:

- Check bone chain continuity
- Verify mass distribution
- Warn about extreme values

3.5.4 Presets

- Hair (light, flexible)
- Cloth (medium stiffness)
- Heavy accessories (stiff, damped)
- Custom (user-defined)

---

4. Data Structures and Property Groups

4.1 Scene Properties

4.1.1 ToolBox (Primary Settings)

```
python

bpy.types.Scene.toolBox: PropertyGroup
```

QC Generator Settings:

- `enum_qcGen_modelType`: Enum("NPC", "CHARACTER", "VIEWMODEL", "PROP", "WORLDMODEL")
- `string_qcGen_outputPath`: StringProperty (QC output location)
- `int_qcGen_scale`: IntProperty (Model scale multiplier)
- `bool_qcGen_generateCollission`: BoolProperty (Auto-generate collision)
- `string_qcGen_existingCollissionCollection`: StringProperty (Use existing collision mesh)

Surface Properties:

- `enum_surfaceprop_category`: Enum (Material categories from surfaceprops.json)
- `enum_surfaceprop_item`: Enum (Specific surface property)

Delta Animation Settings:

- `float_deltaAnim_simmilarityThreshold`: FloatProperty (ValveBiped match percentage, 0-100)

VTF Batch Conversion:

- `string_vtfbatch_inputfolder`: StringProperty (Source folder)
- `string_vtfbatch_outputfolder`: StringProperty (Destination folder)
- `enum_vtfbatch_sourcefiletype`: Enum("PNG", "TGA", "VTF", etc.)
- `enum_vtfbatch_targetfiletype`: Enum("PNG", "TGA", "VTF", etc.)

SMD Export:

- `string_export_folder`: StringProperty (SMD output folder)

4.1.2 QC_PrimaryData

```python
bpy.types.Scene.QC_PrimaryData: PropertyGroup
```

VMT File Paths:

```python
vmt_filepaths: CollectionProperty(type=VMT_FilepathItem)

class VMT_FilepathItem(PropertyGroup):
    filepath: StringProperty
```

Bodygroups:

```python
bodygroup_boxes: CollectionProperty(type=BodygroupBox)

class BodygroupBox(PropertyGroup):
    name: StringProperty
    collections: CollectionProperty(type=BodygroupCollection)

class BodygroupCollection(PropertyGroup):
    name: StringProperty
    enabled: BoolProperty
```

Sequences:

```python
sequence_objectdata: CollectionProperty(type=SequenceObjectData)

class SequenceObjectData(PropertyGroup):
    armatureName: StringProperty
    sequences: CollectionProperty(type=SequenceData)

class SequenceData(PropertyGroup):
    originalName: StringProperty          # NLA track name
    sequenceName: StringProperty          # Export name
    shouldExport: BoolProperty            # Export flag
    qcPath: StringProperty                # SMD file path
    customTag: StringProperty             # Additional QC tags
    enum_activity_category: EnumProperty # Activity category
    enum_activity: EnumProperty           # Specific activity
```

4.2 Activity System

Activity Categories: Based on `MODEL_TYPE_CATEGORY_MAP`

- Dynamically populated based on model type
- Different activities available per category

Example Activities (NPC/Character):

```python
"Basic": ["ACT_IDLE", "ACT_WALK", "ACT_RUN"]
"Combat": ["ACT_IDLE_ANGRY", "ACT_RANGE_ATTACK1", "ACT_MELEE_ATTACK1"]
"Gestures": ["ACT_GESTURE_WAVE", "ACT_GESTURE_SALUTE"]
# etc.
```

---

5. User Interface Design

5.1 Panel Hierarchy

5.2 Panel Specifications

Location: 3D Viewport > Sidebar (N-Panel) **Tab Name**: "VonSourceTools" **Region**: UI (Sidebar) **Space**: VIEW_3D

5.3 UI Conventions

Button Sizing:

- Primary action buttons: `row.scale_y = 2` (double height)
- Standard buttons: Default sizing

Icons:

- Export/Generate: 'CHECKMARK'
- Play/Execute: 'PLAY'
- Collections: 'OUTLINER_OB_GROUP_INSTANCE'
- Refresh: 'FILE_REFRESH'
- Export: 'EXPORT'

Layout Patterns:

- Two-column split for settings: `split(factor=0.5)`
- Boxes for grouped settings: `layout.box()`
- Separators for visual breaks: `layout.separator()`

---

6. File I/O and External Data

6.1 Template System

Location: `storeditems/qcgenerator/templates/`

Command Templates (`commands/`):

- Individual template file per QC command
- Python format string syntax
- Named placeholders for value substitution

Section Ordering (`qc_section_order.json`):

```json
[
  "modelname",
  "staticprop",
  "surfaceprop",
  "cdmaterials",
  "include",
  "body",
  "bodygroup",
  "attachment",
  "hboxset",
  "sequence",
  "collisionmodel",
  "illumposition",
  "origin",
  "includemodel"
]
```

Surface Properties (`surfaceprops.json`):

```json
{
  "category1": {
    "item1": "surfaceprop_value1",
    "item2": "surfaceprop_value2"
  },
  "category2": {
    ...
  }
}
```

6.2 External Armature Files

Storage: External .blend file(s) containing reference armatures **Location**: Determined by `deltaanimtrick_armaturefilelocations()` **Contents**:

- 'proportions' armature
- 'reference_male' armature
- 'reference_female' armature

Import Method:

```python
von_common.importitemfromdict(
    object_name: str,
    target_collection: str,
    file_location_dict: dict
)
```

6.3 JSON Data Loading

Utility Function:

```python
def load_json_dict_to_var(relative_path, filename):
    scriptDir = Path(__file__).parent
    jsonPath = scriptDir / relative_path / filename
    with open(jsonPath, 'r', encoding='utf-8') as f:
        return json.load(f)
```

Usage:

- Load surface properties
- Load QC section ordering
- Load activity definitions
- Load preset configurations

---

7. Workflow Examples

7.1 Complete Player Model Workflow

1. **Import Custom Character Model**:
    - Import FBX/DAE with custom armature
    - Ensure meshes are parented to armature
2. **Run Delta Animation Trick**:
    - Select custom armature
    - Click "Delta Anim Trick (Full)"
    - Addon validates, imports references, applies constraints
    - Mesh transferred to 'proportions' armature
3. **Set Up Animations**:
    - Use NLA editor on 'proportions' armature
    - Import/create animation strips
    - Name strips according to desired sequence names
4. **Configure QC Generator**:
    - Set Model Type to "CHARACTER"
    - Set output path
    - Configure surface property
    - Select animations for export
    - Set activities for each sequence
5. **Set Up Bodygroups** (if applicable):
    - Organize alternate meshes into collections
    - Configure bodygroup boxes
    - Enable desired collections per bodygroup
6. **Generate QC File**:
    - Click "QC Generator - CHARACTER"
    - Review generated QC
    - Adjust as needed
7. **Export SMDs**:
    - Configure export folder
    - Click "Export SMD"
    - Verify exported files
8. **Compile Model**:
    - Run studiomdl.exe with generated QC
    - Test in Garry's Mod

7.2 Simple Prop Workflow

1. **Model Prop**:
    - Create/import prop mesh
    - No armature needed
2. **Configure Materials**:
    - Apply Blender materials
    - Note material names
3. **Configure QC Generator**:
    - Set Model Type to "PROP"
    - Set output path
    - Configure surface property
4. **Generate QC**:
    - Click "QC Generator - PROP"
    - Review for `$staticprop` command
5. **Export and Compile**:
    - Export SMD
    - Compile with studiomdl.exe

---

8. Error Handling and Validation

8.1 User Input Validation

QC Generator:

- Verify output path exists and is writable
- Check for invalid QC command combinations
- Warn if no sequences selected but model type expects animations
- Validate collection references in bodygroups

Delta Animation Trick:

- Require armature selection
- Validate ValveBiped similarity threshold
- Check for conflicting bone names
- Warn if reference armatures can't be imported

Batch Conversion:

- Verify folders exist
- Check for external tool availability
- Validate file format compatibility
- Warn about large batch operations

8.2 Error Reporting

Operator Return Values:

```python
return {'FINISHED'}  # Success
return {'CANCELLED'}  # User cancelled or validation failed
```

User Notifications:

```python
self.report({'ERROR'}, "Error message")
self.report({'WARNING'}, "Warning message")
self.report({'INFO'}, "Info message")
```

Console Logging:

```python
print(f"DEBUG: {debug_info}")
print(f"ERROR Command: {issue} is not applicable to this type of qc.")
```

8.3 Recovery Mechanisms

Undo Support:

- Respect Blender's undo system where possible
- Document operations that can't be undone
- Provide "Restore Objects" for collection changes

Backup Strategies:

- Don't modify original imported armatures
- Create copies for delta animation trick
- Preserve original mesh skinning data

---

9. Performance Considerations

9.1 Optimization Strategies

Data Caching:

- Load JSON files once at registration
- Cache surface property enums
- Store ValveBiped bone lists as constants

Batch Operations:

- Process all armatures in single operator call
- Minimize mode switches (OBJECT ↔ EDIT ↔ POSE)
- Use data-level operations where possible (avoid bpy.ops)

UI Responsiveness:

- Avoid blocking operations in draw() methods
- Use modal operators for long tasks (future)
- Provide progress feedback (future)

9.2 Known Performance Issues

VTF Batch Conversion:

- Blocking operation, UI hangs
- No progress indication
- Subprocess overhead for each file

Delta Animation Trick:

- Multiple mode switches
- Constraint evaluation overhead
- Not optimized for multiple armatures

---

10. Testing Strategy

10.1 Unit Testing

Test Coverage (Proposed):

- QC command template population
- Bodygroup data gathering
- Sequence data gathering
- ValveBiped bone validation
- JSON data loading
- Path validation

Test Framework: Python unittest module

10.2 Integration Testing

Test Scenarios:

1. Complete player model workflow
2. Multi-bodygroup model
3. Viewmodel with animations
4. Static prop
5. Delta animation trick with various armatures
6. Batch VTF conversion

Test Data:

- Sample blend files
- Reference models
- Test textures

10.3 User Acceptance Testing

Key Metrics:

- QC compiles without errors
- Model displays correctly in Garry's Mod
- Animations play correctly
- Bodygroups switch properly
- Materials appear correctly

---

11. Documentation Requirements

11.1 User Documentation

Installation Guide:

- Blender version requirements
- Dependency installation (Blender Source Tools)
- External tool setup (VTFCmd)
- Addon installation steps

Feature Tutorials:

- Delta animation trick step-by-step
- QC generation for each model type
- Bodygroup configuration
- Material setup and conversion
- Jigglebone setup (future)

Troubleshooting Guide:

- Common errors and solutions
- Compatibility issues
- Export problems

11.2 Developer Documentation

Code Documentation:

- Docstrings for all functions
- Module-level documentation
- Inline comments for complex logic

API Reference:

- Public functions and classes
- Property descriptions
- Operator specifications

Architecture Documentation:

- Module dependencies
- Data flow diagrams
- UI hierarchy

11.3 Credits and Attribution

Referenced Projects:

- Blender Source Tools
- sksh70/proportion_trick_script
- VTFCmd/VTFEdit

File: `credits.txt`

---

12. Future Roadmap

12.1 Phase 1 (Current Development)

- ✅ Delta animation trick automation
- ✅ Basic QC generation framework
- ✅ UI foundation
- 🚧 Complete QC command implementation
- 🚧 VMT generation
- 🚧 Sequence export

12.2 Phase 2 (Near-term)

- Jigglebone support
- Attachment point automation
- Hitbox generation
- Collision mesh automation
- Material property mapping
- QC validation and error checking

12.3 Phase 3 (Mid-term)

- Face flexes/phonemes support
- IK setup automation
- Weight painting tools
- Texture baking pipeline
- LOD generation
- Physics constraint setup

12.4 Phase 4 (Long-term)

- Integration with Crowbar
- Model viewer integration
- Batch model processing
- Preset system for common model types
- Cloud-based model library
- Collaborative workflow tools

---

13. Known Issues and Limitations

13.1 Current Issues

Delta Animation Trick:

- Toe bone check has duplicate condition (lines 166, 186)
- Hard-coded armature names
- Limited error recovery
- No progress feedback for multi-armature processing

QC Generator:

- Many commands have placeholder implementations
- Template system not fully integrated
- No validation for complex command combinations
- Path handling may not work cross-platform

VTF Conversion:

- UI hangs during batch conversion
- No progress indication
- Requires external tools (not bundled)
- Error handling for missing tools incomplete

13.2 Limitations

Platform Support:

- Developed for Windows (subprocess calls may fail on Linux/Mac)
- VTFCmd paths hard-coded

Blender Version:

- Targets 4.2.10+
- Backward compatibility not tested
- May break with Blender API changes

Source Engine Version:

- Targets Source 2007/2013
- Source 2 not supported
- Some advanced features may not work in older games

Model Complexity:

- Very high poly counts may cause slowdowns
- Extreme bone counts not tested
- Memory usage not optimized

---

14. Contributing Guidelines

14.1 Code Standards

Python Style:

- Follow PEP 8 where reasonable
- Use type hints where beneficial
- Meaningful variable names
- Comment complex logic

Blender Conventions:

- Use `bl_idname` naming: "von.operation_name"
- Panel IDs: "VONPANEL_PT_panel_name"
- Property naming: prefix with type (bool_, string_, enum_, etc.)

14.2 Submission Process

1. Fork repository
2. Create feature branch
3. Implement changes
4. Test thoroughly
5. Update documentation
6. Submit pull request
7. Code review
8. Merge

14.3 Priority Areas

High Priority:

- Complete QC command implementations
- Error handling improvements
- Cross-platform compatibility
- Documentation

Medium Priority:

- Performance optimizations
- UI polish
- Additional model types
- Preset system

Low Priority:

- Advanced features
- Integration with other tools
- Cloud features

---

15. Technical Dependencies

15.1 Required Software

Blender: 4.2.10 or newer

- Python 3.x (bundled with Blender)
- bpy module

Blender Source Tools: Latest version

- SMD import/export
- QC syntax reference

15.2 Optional Software

VTFCmd / VTFEdit:

- VTF file creation and conversion
- Batch texture processing

Crowbar:

- Model decompilation
- QC extraction

studiomdl.exe:

- Source SDK tool
- Model compilation

15.3 Python Modules

Standard Library:

- json (JSON parsing)
- pathlib (Path handling)
- collections (OrderedDict)
- os (File operations)
- subprocess (External tool execution)

Blender Bundled:

- mathutils (Vector operations)
- bmesh (Mesh data access)

---

16. Licensing and Legal

16.1 Addon License

TBD - To be determined by author

16.2 Third-Party Code

proportion_trick_script:

- Author: sksh70 (B L Λ Z Ξ)
- License: TBD
- URL: https://github.com/sksh70/proportion_trick_script
- Usage: Delta animation trick implementation

Attribution Required:

- CaptainBigButt: Proportion trick discovery
- sksh70: Original script implementation

16.3 Valve Intellectual Property

Source Engine:

- QC format: Valve Corporation
- SMD format: Valve Corporation
- VTF format: Valve Corporation
- ValveBiped skeleton: Valve Corporation

Usage: Educational and modding purposes only

---

17. Changelog

Version 0.0.2 (Current - Development)

- Delta animation trick full automation
- QC builder framework
- UI foundation
- Batch VTF conversion
- Basic SMD export

Version 0.0.1 (Initial)

- Project setup
- Module structure
- Basic operators

---

18. Glossary

QC (QC File): QuakeC-like script file used by Source Engine's studiomdl compiler to define model properties, sequences, bodygroups, etc.

SMD (Static Mesh Data): ASCII-based file format used by Source Engine for geometry and animation data.

VMT (Valve Material Type): Text-based material definition file for Source Engine.

VTF (Valve Texture Format): Binary texture format used by Source Engine.

ValveBiped: Standard skeleton structure used by Source Engine for humanoid characters, specifically for Garry's Mod player models.

Delta Animation Trick / Proportion Trick: Technique for creating custom-proportioned player models that work with Garry's Mod's player animation system.

Bodygroup: Mechanism in Source Engine for swapping different mesh parts (e.g., different hats, accessories).

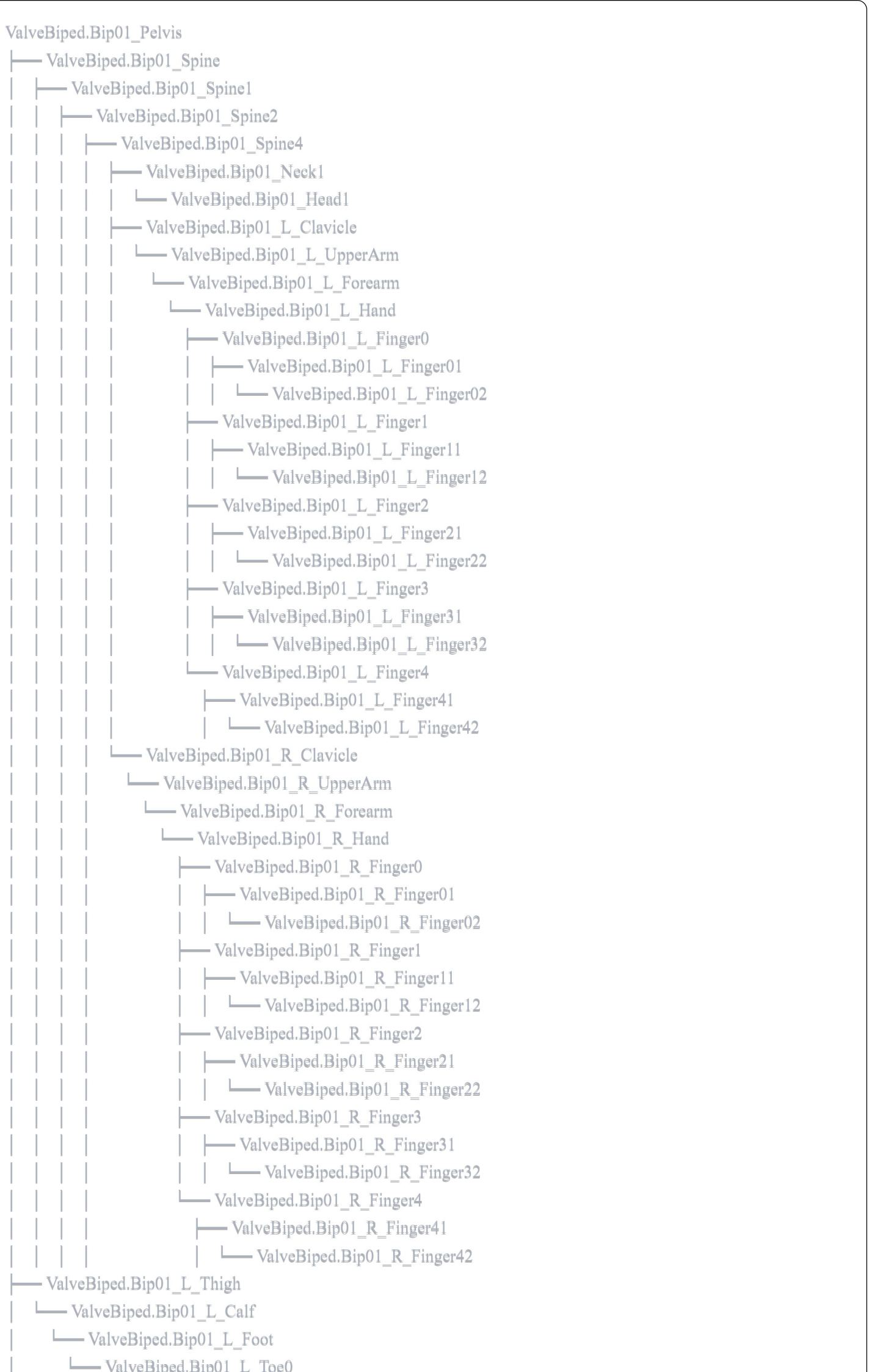
Jigglebone: Source Engine physics-based bone system for secondary motion (hair, cloth, etc.).

Activity: Named animation state in Source Engine (e.g., ACT_IDLE, ACT_RUN).

Surface Property: Physics material property in Source Engine defining sound, friction, etc.

NLA (Non-Linear Animation): Blender's animation system for layering and mixing animation clips.

Appendix A: Complete ValveBiped Bone Hierarchy


ValveBiped.Bip01_Pelvis
ValveBiped.Bip01_Spine
ValveBiped.Bip01_Spine1
ValveBiped.Bip01_Spine2
ValveBiped.Bip01_Spine4
ValveBiped.Bip01_Neck1
ValveBiped.Bip01_Head1
ValveBiped.Bip01_L_Clavicle
ValveBiped.Bip01_L_UpperArm
ValveBiped.Bip01_L_Forearm
ValveBiped.Bip01_L_Hand
ValveBiped.Bip01_L_Finger0
ValveBiped.Bip01_L_Finger01
ValveBiped.Bip01_L_Finger02
ValveBiped.Bip01_L_Finger1
ValveBiped.Bip01_L_Finger11
ValveBiped.Bip01_L_Finger12
ValveBiped.Bip01_L_Finger2
ValveBiped.Bip01_L_Finger21
ValveBiped.Bip01_L_Finger22
ValveBiped.Bip01_L_Finger3
ValveBiped.Bip01_L_Finger31
ValveBiped.Bip01_L_Finger32
ValveBiped.Bip01_L_Finger4
ValveBiped.Bip01_L_Finger41
ValveBiped.Bip01_L_Finger42
ValveBiped.Bip01_R_Clavicle
ValveBiped.Bip01_R_UpperArm
ValveBiped.Bip01_R_Forearm
ValveBiped.Bip01_R_Hand
ValveBiped.Bip01_R_Finger0
ValveBiped.Bip01_R_Finger01
ValveBiped.Bip01_R_Finger02
ValveBiped.Bip01_R_Finger1
ValveBiped.Bip01_R_Finger11
ValveBiped.Bip01_R_Finger12
ValveBiped.Bip01_R_Finger2
ValveBiped.Bip01_R_Finger21
ValveBiped.Bip01_R_Finger22
ValveBiped.Bip01_R_Finger3
ValveBiped.Bip01_R_Finger31
ValveBiped.Bip01_R_Finger32
ValveBiped.Bip01_R_Finger4
ValveBiped.Bip01_R_Finger41
ValveBiped.Bip01_R_Finger42
ValveBiped.Bip01_L_Thigh
ValveBiped.Bip01_L_Calf
ValveBiped.Bip01_L_Foot
ValveBiped.Bip01_L_Toe0

```
└── ValveBiped.Bip01_R_Thigh
    └── ValveBiped.Bip01_R_Calf
        └── ValveBiped.Bip01_R_Foot
            └── ValveBiped.Bip01_R_Toe0
```

Appendix B: Example Generated QC

qc

```
// Generated by VonSourceTools v0.0.2

$modelname "models/player/custom/mycharacter.mdl"

$surfaceprop "flesh"

$cdmaterials "models/player/custom/mycharacter/"

// Reference pose
$body "body" "reference.smd"

// Bodygroups
$bodygroup "head"
{
    studio "head_default.smd"
    studio "head_hat.smd"
    studio "head_helmet.smd"
}

$bodygroup "torso"
{
    studio "torso_default.smd"
    studio "torso_armored.smd"
}

// Animation sequences
$sequence "idle" "anims/idle.smd" fps 30 loop ACT_IDLE 1

$sequence "walk_all" "anims/walk.smd" fps 30 loop ACT_WALK 1

$sequence "run_all" "anims/run.smd" fps 30 loop ACT_RUN 1

$sequence "jump" "anims/jump.smd" fps 30 ACT_JUMP 1

// Collision model
$collisionmodel "phys.smd"
{
    $mass 80
    $inertia 10
    $damping 0.01
    $rotdamping 1.5
}

$illumposition 0 0 40
```

Appendix C: Contact and Support

Author: Vona **GitHub**: https://github.com/1294Angel/VonSourceTools (dev branch) **Blender Artists Thread**: TBD **Discord**: TBD

Bug Reports: GitHub Issues **Feature Requests**: GitHub Discussions **General Support**: Discord / Blender Artists

---

Document Revision History

Version	Date	Author	Changes
1.0	2026-01-27	Claude (AI Assistant)	Initial comprehensive technical specification based on prototype analysis

---

End of Technical Specifications