```
/**
  Copyright (C) 2012-2014 by Autodesk, Inc.
  All rights reserved.

  RS-274D post processor configuration.

  $Revision: 38310 $
  $Date: 2014-12-22 10:46:18 +0100 (ma, 22 dec 2014) $
  POSTTICKET#366

  FORKID {2EECF092-D7C3-4aca-BFE6-377B72950FE9}
*/

description = "RS-274D";
vendor = "Autodesk, Inc.";
vendorUrl = "http://www.autodesk.com";
legal = "Copyright (C) 2012-2014 by Autodesk, Inc.";
certificationLevel = 2;
minimumRevision = 24000;

extension = "nc";
setCodePage("ascii");

capabilities = CAPABILITY_MILLING;
tolerance = spatial(0.002, MM);

minimumChordLength = spatial(0.01, MM);
minimumCircularRadius = spatial(0.01, MM);
maximumCircularRadius = spatial(1000, MM);
minimumCircularSweep = toRad(0.01);
maximumCircularSweep = toRad(180);
allowHelicalMoves = true;
allowedCircularPlanes = undefined; // allow any circular motion



// user-defined properties
properties = {
  writeMachine: true, // write machine
  writeTools: false, // writes the tools
  preloadTool: false, // preloads next tool on tool change if any
  showSequenceNumbers: true, // show sequence numbers
  sequenceNumberStart: 10, // first sequence number
  sequenceNumberIncrement: 5, // increment for sequence numbers
  optionalStop: false, // optional stop
  separateWordsWithSpace: true, // specifies that the words should be
separated with a white space
  singleToolMode: true // post without G28 G43 G54
};

var numberOfToolSlots = 9999;
```

```
var mapCoolantTable = new Table(
  [9, 8, 7],
  {initial:COOLANT_OFF, force:true},
  "Invalid coolant mode"
);

var gFormat = createFormat({prefix:"G", decimals:0});
var mFormat = createFormat({prefix:"M", decimals:0});
var hFormat = createFormat({prefix:"H", decimals:0});
var dFormat = createFormat({prefix:"D", decimals:0});

var xyzFormat = createFormat({decimals:(unit == MM ? 3 : 4)});
var abcFormat = createFormat({decimals:3, forceDecimal:true, scale:DEG});
var feedFormat = createFormat({decimals:(unit == MM ? 1 : 2)});
var toolFormat = createFormat({decimals:0});
var rpmFormat = createFormat({decimals:0});
var secFormat = createFormat({decimals:3, forceDecimal:true}); // seconds
- range 0.001-1000
var taperFormat = createFormat({decimals:1, scale:DEG});

var xOutput = createVariable({prefix:"X"}, xyzFormat);
var yOutput = createVariable({prefix:"Y"}, xyzFormat);
var zOutput = createVariable({prefix:"Z"}, xyzFormat);
var aOutput = createVariable({prefix:"A"}, abcFormat);
var bOutput = createVariable({prefix:"B"}, abcFormat);
var cOutput = createVariable({prefix:"C"}, abcFormat);
var feedOutput = createVariable({prefix:"F"}, feedFormat);
var sOutput = createVariable({prefix:"S", force:true}, rpmFormat);
var dOutput = createVariable({}, dFormat);

// circular output
var iOutput = createReferenceVariable({prefix:"I"}, xyzFormat);
var jOutput = createReferenceVariable({prefix:"J"}, xyzFormat);
var kOutput = createReferenceVariable({prefix:"K"}, xyzFormat);

var gMotionModal = createModal({force:true}, gFormat); // modal group 1
// G0-G3, ...
var gPlaneModal = createModal({onchange:function ()
{gMotionModal.reset();}}, gFormat); // modal group 2 // G17-19
var gAbsIncModal = createModal({}, gFormat); // modal group 3 // G90-91
var gFeedModeModal = createModal({}, gFormat); // modal group 5 // G93-94
var gUnitModal = createModal({}, gFormat); // modal group 6 // G20-21
var gCycleModal = createModal({}, gFormat); // modal group 9 // G81, ...
var gRetractModal = createModal({}, gFormat); // modal group 10 // G98-99

var WARNING_WORK_OFFSET = 0;

// collected state
var sequenceNumber;
var currentWorkOffset;

/**
  Writes the specified block.
*/
```

```
function writeBlock() {
  if (properties.showSequenceNumbers) {
    writeWords2("N" + sequenceNumber, arguments);
    sequenceNumber += properties.sequenceNumberIncrement;
  } else {
    writeWords(arguments);
  }
}

function formatComment(text) {
  return "(" + String(text).replace(/[\(\)]/g, "") + ")";
}

/**
  Output a comment.
*/
function writeComment(text) {
  writeln(formatComment(text));
}

function onOpen() {
  if (!machineConfiguration.isMachineCoordinate(0)) {
    aOutput.disable();
  }
  if (!machineConfiguration.isMachineCoordinate(1)) {
    bOutput.disable();
  }
  if (!machineConfiguration.isMachineCoordinate(2)) {
    cOutput.disable();
  }

  if (!properties.separateWordsWithSpace) {
    setWordSeparator("");
  }

  sequenceNumber = properties.sequenceNumberStart;
  writeln("%");

  if (programName) {
    writeComment(programName);
  }
  if (programComment) {
    writeComment(programComment);
  }

  // dump machine configuration
  var vendor = machineConfiguration.getVendor();
  var model = machineConfiguration.getModel();
  var description = machineConfiguration.getDescription();

  if (properties.writeMachine && (vendor || model || description)) {
    writeComment(localize("Machine"));
    if (vendor) {
      writeComment("  " + localize("vendor") + ": " + vendor);
```

```
    }
    if (model) {
      writeComment("  " + localize("model") + ": " + model);
    }
    if (description) {
      writeComment("  " + localize("description") + ": "  + description);
    }
  }

  // dump tool information
  if (properties.writeTools) {
    var zRanges = {};
    if (is3D()) {
      var numberOfSections = getNumberOfSections();
      for (var i = 0; i < numberOfSections; ++i) {
        var section = getSection(i);
        var zRange = section.getGlobalZRange();
        var tool = section.getTool();
        if (zRanges[tool.number]) {
          zRanges[tool.number].expandToRange(zRange);
        } else {
          zRanges[tool.number] = zRange;
        }
      }
    }

    var tools = getToolTable();
    if (tools.getNumberOfTools() > 0) {
      for (var i = 0; i < tools.getNumberOfTools(); ++i) {
        var tool = tools.getTool(i);
        var comment = "T" + toolFormat.format(tool.number) + " " +
          "D=" + xyzFormat.format(tool.diameter) + " " +
          localize("CR") + "=" + xyzFormat.format(tool.cornerRadius);
        if ((tool.taperAngle > 0) && (tool.taperAngle < Math.PI)) {
          comment += " " + localize("TAPER") + "=" +
taperFormat.format(tool.taperAngle) + localize("deg");
        }
        if (zRanges[tool.number]) {
          comment += " - " + localize("ZMIN") + "=" +
xyzFormat.format(zRanges[tool.number].getMinimum());
        }
        comment += " - " + getToolTypeName(tool.type);
        writeComment(comment);
      }
    }
  }

  // absolute coordinates and feed per min
  writeBlock(gAbsIncModal.format(90), gFeedModeModal.format(94));
  writeBlock(gPlaneModal.format(17));

  switch (unit) {
  case IN:
    writeBlock(gUnitModal.format(20));
```

```
    break;
  case MM:
    writeBlock(gUnitModal.format(21));
    break;
  }
}

function onComment(message) {
  writeComment(message);
}

/** Force output of X, Y, and Z. */
function forceXYZ() {
  xOutput.reset();
  yOutput.reset();
  zOutput.reset();
}

/** Force output of A, B, and C. */
function forceABC() {
  aOutput.reset();
  bOutput.reset();
  cOutput.reset();
}

/** Force output of X, Y, Z, A, B, C, and F on next output. */
function forceAny() {
  forceXYZ();
  forceABC();
  feedOutput.reset();
}

var currentWorkPlaneABC = undefined;

function forceWorkPlane() {
  currentWorkPlaneABC = undefined;
}

function setWorkPlane(abc) {
  if (!machineConfiguration.isMultiAxisConfiguration()) {
    return; // ignore
  }

  if (!((currentWorkPlaneABC == undefined) ||
        abcFormat.areDifferent(abc.x, currentWorkPlaneABC.x) ||
        abcFormat.areDifferent(abc.y, currentWorkPlaneABC.y) ||
        abcFormat.areDifferent(abc.z, currentWorkPlaneABC.z))) {
    return; // no change
  }

  onCommand(COMMAND_UNLOCK_MULTI_AXIS);

  // NOTE: add retract here
```

```
  writeBlock(
    gMotionModal.format(0),
    conditional(machineConfiguration.isMachineCoordinate(0), "A" +
abcFormat.format(abc.x)),
    conditional(machineConfiguration.isMachineCoordinate(1), "B" +
abcFormat.format(abc.y)),
    conditional(machineConfiguration.isMachineCoordinate(2), "C" +
abcFormat.format(abc.z))
  );

  onCommand(COMMAND_LOCK_MULTI_AXIS);

  currentWorkPlaneABC = abc;
}

var closestABC = false; // choose closest machine angles
var currentMachineABC;

function getWorkPlaneMachineABC(workPlane) {
  var W = workPlane; // map to global frame

  var abc = machineConfiguration.getABC(W);
  if (closestABC) {
    if (currentMachineABC) {
      abc = machineConfiguration.remapToABC(abc, currentMachineABC);
    } else {
      abc = machineConfiguration.getPreferredABC(abc);
    }
  } else {
    abc = machineConfiguration.getPreferredABC(abc);
  }

  try {
    abc = machineConfiguration.remapABC(abc);
    currentMachineABC = abc;
  } catch (e) {
    error(
      localize("Machine angles not supported") + ":"
      + conditional(machineConfiguration.isMachineCoordinate(0), " A" +
abcFormat.format(abc.x))
      + conditional(machineConfiguration.isMachineCoordinate(1), " B" +
abcFormat.format(abc.y))
      + conditional(machineConfiguration.isMachineCoordinate(2), " C" +
abcFormat.format(abc.z))
    );
  }

  var direction = machineConfiguration.getDirection(abc);
  if (!isSameDirection(direction, W.forward)) {
    error(localize("Orientation not supported."));
  }

  if (!machineConfiguration.isABCSupported(abc)) {
    error(
```

```
      localize("Work plane is not supported") + ":"
      + conditional(machineConfiguration.isMachineCoordinate(0), " A" +
abcFormat.format(abc.x))
      + conditional(machineConfiguration.isMachineCoordinate(1), " B" +
abcFormat.format(abc.y))
      + conditional(machineConfiguration.isMachineCoordinate(2), " C" +
abcFormat.format(abc.z))
    );
  }

  var tcp = true;
  if (tcp) {
    setRotation(W); // TCP mode
  } else {
    var O = machineConfiguration.getOrientation(abc);
    var R = machineConfiguration.getRemainingOrientation(abc, W);
    setRotation(R);
  }

  return abc;
}

function onSection() {
  var insertToolCall = isFirstSection() ||
    currentSection.getForceToolChange &&
currentSection.getForceToolChange() ||
    (tool.number != getPreviousSection().getTool().number);

  var retracted = false; // specifies that the tool has been retracted to
the safe plane
  var newWorkOffset = isFirstSection() ||
    (getPreviousSection().workOffset != currentSection.workOffset); //
work offset changes
  var newWorkPlane = isFirstSection() ||
    !isSameDirection(getPreviousSection().getGlobalFinalToolAxis(),
currentSection.getGlobalInitialToolAxis());
  if (insertToolCall || newWorkOffset || newWorkPlane) {

    // stop spindle before retract during tool change
    if (insertToolCall && !isFirstSection()) {
      onCommand(COMMAND_STOP_SPINDLE);
    }

    // retract to safe plane
    retracted = true;
    if (!properties.singleToolMode) {
    writeBlock(gFormat.format(28), gAbsIncModal.format(91), "Z" +
xyzFormat.format(0)); // retract
    writeBlock(gAbsIncModal.format(90));
    }
    zOutput.reset();

  }
```

```
  writeln("");

  if (hasParameter("operation-comment")) {
    var comment = getParameter("operation-comment");
    if (comment) {
      writeComment(comment);
    }
  }

  if (insertToolCall) {
    forceWorkPlane();

    retracted = true;
    onCommand(COMMAND_COOLANT_OFF);

    if (!isFirstSection() && properties.optionalStop) {
      onCommand(COMMAND_OPTIONAL_STOP);
    }

    if (tool.number > numberOfToolSlots) {
      warning(localize("Tool number exceeds maximum value."));
    }

    writeBlock("T" + toolFormat.format(tool.number), mFormat.format(6));
    if (tool.comment) {
      writeComment(tool.comment);
    }
    var showToolZMin = false;
    if (showToolZMin) {
      if (is3D()) {
        var numberOfSections = getNumberOfSections();
        var zRange = currentSection.getGlobalZRange();
        var number = tool.number;
        for (var i = currentSection.getId() + 1; i < numberOfSections;
++i) {
          var section = getSection(i);
          if (section.getTool().number != number) {
            break;
          }
          zRange.expandToRange(section.getGlobalZRange());
        }
        writeComment(localize("ZMIN") + "=" + zRange.getMinimum());
      }
    }

    if (properties.preloadTool) {
      var nextTool = getNextTool(tool.number);
      if (nextTool) {
        writeBlock("T" + toolFormat.format(nextTool.number));
      } else {
        // preload first tool
        var section = getSection(0);
        var firstToolNumber = section.getTool().number;
        if (tool.number != firstToolNumber) {
```

```
          writeBlock("T" + toolFormat.format(firstToolNumber));
        }
      }
    }
  }

  if (insertToolCall ||
      isFirstSection() ||
      (rpmFormat.areDifferent(tool.spindleRPM, sOutput.getCurrent())) ||
      (tool.clockwise != getPreviousSection().getTool().clockwise)) {
    if (tool.spindleRPM < 1) {
      error(localize("Spindle speed out of range."));
    }
    if (tool.spindleRPM > 99999) {
      warning(localize("Spindle speed exceeds maximum value."));
    }
    writeBlock(
      sOutput.format(tool.spindleRPM), mFormat.format(tool.clockwise ? 3
: 4)
    );
  }

  // wcs
  var workOffset = currentSection.workOffset;
  if (workOffset == 0) {
    warningOnce(localize("Work offset has not been specified. Using G54
as WCS."), WARNING_WORK_OFFSET);
    workOffset = 1;
  }

  if (workOffset > 0 && !properties.singleToolMode) {
    if (workOffset > 6) {
      var code = workOffset - 6;
      if (code > 3) {
        error(localize("Work offset out of range."));
        return;
      }
      if (workOffset != currentWorkOffset) {
        writeBlock(gFormat.format(59) + "." + code);
        currentWorkOffset = workOffset;
      }
    } else {
      if (workOffset != currentWorkOffset) {
        writeBlock(gFormat.format(53 + workOffset)); // G54->G59
        currentWorkOffset = workOffset;
      }
    }
  }

  forceXYZ();

  if (machineConfiguration.isMultiAxisConfiguration()) { // use 5-axis
indexing for multi-axis mode
    // set working plane after datum shift
```

```
    var abc = new Vector(0, 0, 0);
    if (currentSection.isMultiAxis()) {
      forceWorkPlane();
      cancelTransformation();
    } else {
      abc = getWorkPlaneMachineABC(currentSection.workPlane);
    }
    setWorkPlane(abc);
  } else { // pure 3D
    var remaining = currentSection.workPlane;
    if (!isSameDirection(remaining.forward, new Vector(0, 0, 1))) {
      error(localize("Tool orientation is not supported."));
      return;
    }
    setRotation(remaining);
  }

  // set coolant after we have positioned at Z
  {
    var c = mapCoolantTable.lookup(tool.coolant);
    if (c) {
      writeBlock(mFormat.format(c));
    } else {
      warning(localize("Coolant not supported."));
    }
  }

  forceAny();

  var initialPosition =
getFramePosition(currentSection.getInitialPosition());
  if (!retracted) {
    if (getCurrentPosition().z < initialPosition.z) {
      writeBlock(gMotionModal.format(0),
zOutput.format(initialPosition.z));
    }
  }

  if (insertToolCall || retracted) {
    var lengthOffset = tool.lengthOffset;
    if (lengthOffset > numberOfToolSlots) {
      error(localize("Length offset out of range."));
      return;
    }

    gMotionModal.reset();
    writeBlock(gPlaneModal.format(17));

    if (!machineConfiguration.isHeadConfiguration()) {
      writeBlock(
        gAbsIncModal.format(90),
        gMotionModal.format(0), xOutput.format(initialPosition.x),
yOutput.format(initialPosition.y)
```

```
      );
      if (!properties.singleToolMode) {
        writeBlock(gMotionModal.format(0), gFormat.format(43),
zOutput.format(initialPosition.z), hFormat.format(lengthOffset));
      } else {
        writeBlock(gMotionModal.format(0),
zOutput.format(initialPosition.z));
      }
    } else {
      writeBlock(
        gAbsIncModal.format(90),
        gMotionModal.format(0),
        gFormat.format(43), xOutput.format(initialPosition.x),
        yOutput.format(initialPosition.y),
        zOutput.format(initialPosition.z), hFormat.format(lengthOffset)
      );
    }
  } else {
    writeBlock(
      gAbsIncModal.format(90),
      gMotionModal.format(0),
      xOutput.format(initialPosition.x),
      yOutput.format(initialPosition.y)
    );
  }
}

function onDwell(seconds) {
  if (seconds > 99999.999) {
    warning(localize("Dwelling time is out of range."));
  }
  seconds = clamp(0.001, seconds, 99999.999);
  writeBlock(gFormat.format(4), "P" + secFormat.format(seconds));
}

function onSpindleSpeed(spindleSpeed) {
  writeBlock(sOutput.format(spindleSpeed));
}

function onCycle() {
  writeBlock(gPlaneModal.format(17));
}

function getCommonCycle(x, y, z, r) {
  forceXYZ();
  return [xOutput.format(x), yOutput.format(y),
    zOutput.format(z),
    "R" + xyzFormat.format(r)];
}

function onCyclePoint(x, y, z) {
  if (isFirstCyclePoint()) {
    repositionToCycleClearance(cycle, x, y, z);
```

```
    // return to initial Z which is clearance plane and set absolute mode

    var F = cycle.feedrate;
    var P = (cycle.dwell == 0) ? 0 : clamp(0.001, cycle.dwell,
99999.999); // in seconds

    switch (cycleType) {
    case "drilling":
      writeBlock(
        gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(81),
        getCommonCycle(x, y, z, cycle.retract),
        feedOutput.format(F)
      );
      break;
    case "counter-boring":
      if (P > 0) {
        writeBlock(
          gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(82),
          getCommonCycle(x, y, z, cycle.retract),
          "P" + secFormat.format(P), // not optional
          feedOutput.format(F)
        );
      } else {
        writeBlock(
          gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(81),
          getCommonCycle(x, y, z, cycle.retract),
          feedOutput.format(F)
        );
      }
      break;
    case "chip-breaking":
      expandCyclePoint(x, y, z);
      break;
    case "deep-drilling":
      if (P > 0) {
        expandCyclePoint(x, y, z);
      } else {
        writeBlock(
          gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(83),
          getCommonCycle(x, y, z, cycle.retract),
          "Q" + xyzFormat.format(cycle.incrementalDepth),
          feedOutput.format(F)
        );
      }
      break;
    case "tapping":
      if (!F) {
        F = tool.getTappingFeedrate();
      }
      writeBlock(
```

```
        gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format((tool.type == TOOL_TAP_LEFT_HAND) ? 74 : 84),
        getCommonCycle(x, y, z, cycle.retract),
        feedOutput.format(F)
      );
      break;
    case "left-tapping":
      if (!F) {
        F = tool.getTappingFeedrate();
      }
      writeBlock(
        gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(74),
        getCommonCycle(x, y, z, cycle.retract),
        feedOutput.format(F)
      );
      break;
    case "right-tapping":
      if (!F) {
        F = tool.getTappingFeedrate();
      }
      writeBlock(
        gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(84),
        getCommonCycle(x, y, z, cycle.retract),
        feedOutput.format(F)
      );
      break;
    case "fine-boring": // not supported
      expandCyclePoint(x, y, z);
      break;
    case "back-boring":
      if (P > 0) {
        expandCyclePoint(x, y, z);
      } else {
        var I = cycle.shift * 1;
        var J = cycle.shift * 0;
        writeBlock(
          gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(87),
          getCommonCycle(x, y, z, cycle.retract),
          "I" + xyzFormat.format(I),
          "J" + xyzFormat.format(J),
          "K" + xyzFormat.format(cycle.bottom - cycle.backBoreDistance),
          feedOutput.format(F)
        );
      }
      break;
    case "reaming":
      writeBlock(
        gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(85),
        getCommonCycle(x, y, z, cycle.retract),
        feedOutput.format(F)
```

```
      );
      break;
    case "stop-boring":
      writeBlock(
        gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(86),
        getCommonCycle(x, y, z, cycle.retract),
        feedOutput.format(F),
        // conditional(P > 0, "P" + secFormat.format(P)),
        "P" + secFormat.format(P) // not optional
      );
      break;
    case "manual-boring":
      writeBlock(
        gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(88),
        getCommonCycle(x, y, z, cycle.retract),
        "P" + secFormat.format(P), // not optional
        feedOutput.format(F)
      );
      break;
    case "boring":
      if (P > 0) {
        writeBlock(
          gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(89),
          getCommonCycle(x, y, z, cycle.retract),
          "P" + secFormat.format(P), // not optional
          feedOutput.format(F)
        );
      } else {
        writeBlock(
          gRetractModal.format(98), gAbsIncModal.format(90),
gCycleModal.format(85),
          getCommonCycle(x, y, z, cycle.retract),
          feedOutput.format(F)
        );
      }
      break;
    default:
      expandCyclePoint(x, y, z);
    }
  } else {
    if (cycleExpanded) {
      expandCyclePoint(x, y, z);
    } else {
      var _x = xOutput.format(x);
      var _y = yOutput.format(y);
      if (!_x && !_y) {
        xOutput.reset(); // at least one axis is required
        _x = xOutput.format(x);
      }
      writeBlock(_x, _y);
    }
```

```
  }
}

function onCycleEnd() {
  if (!cycleExpanded) {
    writeBlock(gCycleModal.format(80));
    zOutput.reset();
  }
}

var pendingRadiusCompensation = -1;

function onRadiusCompensation() {
  pendingRadiusCompensation = radiusCompensation;
}

function onRapid(_x, _y, _z) {
  var x = xOutput.format(_x);
  var y = yOutput.format(_y);
  var z = zOutput.format(_z);
  if (x || y || z) {
    if (pendingRadiusCompensation >= 0) {
      error(localize("Radius compensation mode cannot be changed at rapid
traversal."));
      return;
    }
    writeBlock(gMotionModal.format(0), x, y, z);
    feedOutput.reset();
  }
}

function onLinear(_x, _y, _z, feed) {
  // at least one axis is required
  if (pendingRadiusCompensation >= 0) {
    // ensure that we end at desired position when compensation is turned
off
    xOutput.reset();
    yOutput.reset();
  }
  var x = xOutput.format(_x);
  var y = yOutput.format(_y);
  var z = zOutput.format(_z);
  var f = feedOutput.format(feed);
  if (x || y || z) {
    if (pendingRadiusCompensation >= 0) {
      pendingRadiusCompensation = -1;
      var d = tool.diameterOffset;
      if (d > numberOfToolSlots) {
        warning(localize("The diameter offset exceeds the maximum
value."));
      }
      writeBlock(gPlaneModal.format(17));
      switch (radiusCompensation) {
      case RADIUS_COMPENSATION_LEFT:
```

```
        dOutput.reset();
        writeBlock(gMotionModal.format(1), gFormat.format(41), x, y, z,
dOutput.format(d), f);
        break;
      case RADIUS_COMPENSATION_RIGHT:
        dOutput.reset();
        writeBlock(gMotionModal.format(1), gFormat.format(42), x, y, z,
dOutput.format(d), f);
        break;
      default:
        writeBlock(gMotionModal.format(1), gFormat.format(40), x, y, z,
f);
      }
    } else {
      writeBlock(gMotionModal.format(1), x, y, z, f);
    }
  } else if (f) {
    if (getNextRecord().isMotion()) { // try not to output feed without
motion
      feedOutput.reset(); // force feed on next line
    } else {
      writeBlock(gMotionModal.format(1), f);
    }
  }
}

function onRapid5D(_x, _y, _z, _a, _b, _c) {
  if (!currentSection.isOptimizedForMachine()) {
    error(localize("This post configuration has not been customized for
5-axis simultaneous toolpath."));
    return;
  }
  if (pendingRadiusCompensation >= 0) {
    error(localize("Radius compensation mode cannot be changed at rapid
traversal."));
    return;
  }
  var x = xOutput.format(_x);
  var y = yOutput.format(_y);
  var z = zOutput.format(_z);
  var a = aOutput.format(_a);
  var b = bOutput.format(_b);
  var c = cOutput.format(_c);
  writeBlock(gMotionModal.format(0), x, y, z, a, b, c);
  feedOutput.reset();
}

function onLinear5D(_x, _y, _z, _a, _b, _c, feed) {
  if (!currentSection.isOptimizedForMachine()) {
    error(localize("This post configuration has not been customized for
5-axis simultaneous toolpath."));
    return;
  }
  // at least one axis is required
```

```
  if (pendingRadiusCompensation >= 0) {
    error(localize("Radius compensation cannot be activated/deactivated
for 5-axis move."));
    return;
  }
  var x = xOutput.format(_x);
  var y = yOutput.format(_y);
  var z = zOutput.format(_z);
  var a = aOutput.format(_a);
  var b = bOutput.format(_b);
  var c = cOutput.format(_c);
  var f = feedOutput.format(feed);
  if (x || y || z || a || b || c) {
    writeBlock(gMotionModal.format(1), x, y, z, a, b, c, f);
  } else if (f) {
    if (getNextRecord().isMotion()) { // try not to output feed without
motion
      feedOutput.reset(); // force feed on next line
    } else {
      writeBlock(gMotionModal.format(1), f);
    }
  }
}

function onCircular(clockwise, cx, cy, cz, x, y, z, feed) {
  // one of X/Y and I/J are required and likewise

  if (pendingRadiusCompensation >= 0) {
    error(localize("Radius compensation cannot be activated/deactivated
for a circular move."));
    return;
  }

  var start = getCurrentPosition();

  if (isFullCircle()) {
    if (isHelical()) {
      linearize(tolerance);
      return;
    }
    // TAG: are 360deg arcs supported
    switch (getCircularPlane()) {
    case PLANE_XY:
      writeBlock(gPlaneModal.format(17), gMotionModal.format(clockwise ?
2 : 3), xOutput.format(x), iOutput.format(cx - start.x, 0),
jOutput.format(cy - start.y, 0), feedOutput.format(feed));
      break;
    case PLANE_ZX:
      writeBlock(gPlaneModal.format(18), gMotionModal.format(clockwise ?
2 : 3), zOutput.format(z), iOutput.format(cx - start.x, 0),
kOutput.format(cz - start.z, 0), feedOutput.format(feed));
      break;
    case PLANE_YZ:
```

```
      writeBlock(gPlaneModal.format(19), gMotionModal.format(clockwise ?
2 : 3), yOutput.format(y), jOutput.format(cy - start.y, 0),
kOutput.format(cz - start.z, 0), feedOutput.format(feed));
      break;
    default:
      linearize(tolerance);
    }
  } else {
    switch (getCircularPlane()) {
    case PLANE_XY:
      writeBlock(gPlaneModal.format(17), gMotionModal.format(clockwise ?
2 : 3), xOutput.format(x), yOutput.format(y), zOutput.format(z),
iOutput.format(cx - start.x, 0), jOutput.format(cy - start.y, 0),
feedOutput.format(feed));
      break;
    case PLANE_ZX:
      writeBlock(gPlaneModal.format(18), gMotionModal.format(clockwise ?
2 : 3), xOutput.format(x), yOutput.format(y), zOutput.format(z),
iOutput.format(cx - start.x, 0), kOutput.format(cz - start.z, 0),
feedOutput.format(feed));
      break;
    case PLANE_YZ:
      writeBlock(gPlaneModal.format(19), gMotionModal.format(clockwise ?
2 : 3), xOutput.format(x), yOutput.format(y), zOutput.format(z),
jOutput.format(cy - start.y, 0), kOutput.format(cz - start.z, 0),
feedOutput.format(feed));
      break;
    default:
      linearize(tolerance);
    }
  }
}

var mapCommand = {
  COMMAND_STOP:0,
  COMMAND_OPTIONAL_STOP:1,
  COMMAND_END:2,
  COMMAND_SPINDLE_CLOCKWISE:3,
  COMMAND_SPINDLE_COUNTERCLOCKWISE:4,
  COMMAND_STOP_SPINDLE:5,
  COMMAND_ORIENTATE_SPINDLE:19,
  COMMAND_LOAD_TOOL:6,
  COMMAND_COOLANT_ON:8,
  COMMAND_COOLANT_OFF:9
};

function onCommand(command) {
  switch (command) {
  case COMMAND_START_SPINDLE:
    onCommand(tool.clockwise ? COMMAND_SPINDLE_CLOCKWISE :
COMMAND_SPINDLE_COUNTERCLOCKWISE);
    return;
  case COMMAND_LOCK_MULTI_AXIS:
    return;
```

```
  case COMMAND_UNLOCK_MULTI_AXIS:
    return;
  case COMMAND_BREAK_CONTROL:
    return;
  case COMMAND_TOOL_MEASURE:
    return;
  }

  var stringId = getCommandStringId(command);
  var mcode = mapCommand[stringId];
  if (mcode != undefined) {
    writeBlock(mFormat.format(mcode));
  } else {
    onUnsupportedCommand(command);
  }
}

function onSectionEnd() {
  writeBlock(gPlaneModal.format(17));
  forceAny();
}

function onClose() {
  onCommand(COMMAND_COOLANT_OFF);

  if (!properties.singleToolMode) {
    writeBlock(gFormat.format(28), gAbsIncModal.format(91), "Z" +
xyzFormat.format(0)); // retract
    zOutput.reset();

  setWorkPlane(new Vector(0, 0, 0)); // reset working plane

  if (!machineConfiguration.hasHomePositionX() &&
!machineConfiguration.hasHomePositionY()) {
      writeBlock(gFormat.format(28), gAbsIncModal.format(91), "X" +
xyzFormat.format(0), "Y" + xyzFormat.format(0)); // return to home
  } else {
    var homeX;
    if (machineConfiguration.hasHomePositionX()) {
      homeX = "X" +
xyzFormat.format(machineConfiguration.getHomePositionX());
    }
    var homeY;
    if (machineConfiguration.hasHomePositionY()) {
      homeY = "Y" +
xyzFormat.format(machineConfiguration.getHomePositionY());
    }
    writeBlock(gAbsIncModal.format(90), gFormat.format(53),
gMotionModal.format(0), homeX, homeY);
  }
  }

  onImpliedCommand(COMMAND_END);
  onImpliedCommand(COMMAND_STOP_SPINDLE);
```

```
  writeBlock(mFormat.format(30)); // stop program, spindle stop, coolant
off
  writeln("%");
}
```