|  |  |
| --- | --- |
| OASIS Baseline Scanpath User Manual |  |
|  | |
|  | |

**Project 3024, Acceleration of Large-Scale Additive Manufacturing**



(Released: August 2020)

A picture containing drawing

Description automatically generated

**Submitted by:**

David Toledano  
Systems Engineer

GE Research

1 Research Circle

Niskayuna, New York 12309

Phone: 518-387-5189  
Fax: 518-387-7080

**ncdmm.org**

**americamakes.us**

"This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8650-16-2-5700. The U.S. Government is authorized to reproduce and distribute reprints for. Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government."

List of Figures 3

List of Tables 3

1. Executive Summary 4

2. Introduction and Problem Statement 4

2.1. Use case 4

2.2. Features of the ALSAM XML scan schema 4

2.3. Limitations 5

2.3.1. Baseline code limitations 5

2.3.2. Potential machine limitations 6

3. Scanpath generation overview 6

3.1. End-to-end process 6

3.2. Requirements to execute the baseline code 7

4. Quick start: Download and run some examples 8

4.1. Set things up on your machine 8

4.2. Generate sample builds 9

5. Setting up a new build 11

5.1. General considerations 11

5.2. Recommended navigation through config-file tabs 12

5.3. Typical configuration file errors 13

5.4. Specifications and bounds 14

6. Configuration file reference 16

6.1. Tab 1.Header 16

6.2. Tab 2.General 16

6.3. Tab 3.VelocityProfiles 17

6.4. Tab 4.SegmentStyles 18

6.5. Tab 5.Regions 19

6.6. Tab 6.Parts 20

6.7. Tab 7.PathProcessing 21

6.8. Tab 8.Stripes 22

7. XML Layer format 22

7.1. Overview 22

7.2. Slic3r 23

8. XML Scan file content 23

8.1.1. Header 24

8.1.2. VelocityProfileList 24

8.1.3. SegmentStyleList 24

8.1.4. TrajectoryList 25

8.2. Mapping configuration-file “regions” to trajectories and paths 26

8.2.1. Grouping parts and regions into trajectories 26

8.2.2. An example 26

List of Abbreviations 28

List of XML Scan Schema Terminology 29

List of Figures

Figure 1 End-to-end of generating a built object from STL files 7

Figure 2 Structure of the Executable folder after downloading Slic3r 8

Figure 3 Recommended “bottom-up” method for configuration file setup 13

List of Tables

Table 1 Sample Parts table, for trajectory generation example 27

Table 2 Sample RegionProfile table, for trajectory generation example 27

Table 3 Results of trajectory generation example 27

**Disclaimer**

Unless otherwise noted, modules developed by GE under ALSAM project 3024 are Copyright 2020 America Makes and are licensed under the BSD 2-clause license. The following text appears at the start of each file:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Executive Summary

This document describes the usage of the baseline scanpath generation code and associated configuration file to generate scanpaths for powder-bed fusion Additive Manufacturing (PBFAM) involving lasers and metallic powders. The code was developed as a method to create sets of hatches and contours in an XML format which may be fed to an open-source machine controller. The XML format is intended for use on a wide range of PBFAM machines, pending the development of a compatible machine controller for any particular class of machines

The baseline code is capable of generating all current features of the XML schema as described in “ALSAM3024 multiLaser XML schema 2020323.docx” although it has numerous limitations (and opportunities for expansion) as noted in following sections

This user manual replicate information in the “Quick-start guide” and adds information to help set up your own builds, including a field-by-field reference to the configuration file

# Introduction and Problem Statement

## Use case

This code is intended to generate build files which demonstrate features of the ALSAM XML scan schema. Parts are to be defined via STL files. Individual builds are be described via an Excel-based configuration file which indicates the filename and build position of each part, assigns scan strategies to hatches and/or contours by part, and indicates the relative build order of each part

Within an individual part, the only scan-strategy differentiation provided by this code is between hatches and contours (i.e. all hatches of part X receive the same scan strategy, irrespective of geometry)

As an intermediate step prior to generating scan files, the code generates XML layer files, using an ALSAM XML layer schema. This simplifies the overall build process by separating part-slicing from scanpath creation and allows alternate scanpaths to be regenerated from the same set of layer files. However, use and creation of XML layer files are not end-user requirements because layer files are not utilized by the machine controller code

The code was developed for use on a Windows machine and reads a Microsoft Excel-based configuration file. However, the XML scan schema itself is intended to be platform independent

## Features of the ALSAM XML scan schema

See “ALSAM3024 multiLaser XML schema 2020323.docx” for details of the schema itself. To recap the features of the scan schema itself,

* The XML scan schema provides a machine-independent method of describing build files for powder-bed fusion additive machines. The scan files themselves are machine-independent, analogous to PDF-formatted documents
* The schema provides fields to describe and configure various scan strategies (combinations of laser power, velocity and other factors) and apply these strategies to build trajectories (segments of marks and jumps). A separate XML scan file is to be generated for each build layer, named in numerical order, for example scan\_001.xml, scan\_002.xml … scan\_135.xml (with sufficient leading zeroes in the layer number to ensure proper alphanumerical sorting)
* Scan files are to be interpreted by a Machine Controller to be developed by end users for their particular model of printer. Under the current ALSAM project, a Machine Controller is provided for a GE Concept Laser M2 machine
* Within the schema, scan strategy and coordinates can be altered segment-by-segment to enable complete control over difficult geometries
* The schema does *not* support dynamic scan strategies, in which scan parameters within an individual layer are adjusted based on real-time sensor feedback from the same layer. However, such feedback could be used to alter the parameters used to generate build files for *successive* layers. This is not a feature of the baseline source code

## Limitations

### Baseline code limitations

The scan schema provides much more flexibility than the ALSAM 3024 scanpath generation code and configuration file. Some limitations of the baseline code, which could be addressed by third-party development, include:

* Layer thickness is constant across all layers of the build, based on the value indicated in the configuration file
* All parts in the build are sliced at this same layer height, and all parts are hatched/contoured on all layers. Skipping layers, to achieve higher layer heights for individual parts, is not implemented
* Parts cannot be subdivided into smaller areas to apply individual scanpath strategies
  + Each part is permitted a single strategy (a combination of laser power, velocity and other parameters) to be applied to all its hatches, and a separate strategy for all its contours – across all layers and regions
  + The following values are also held fixed across all layers of an individual part: hatch/contour offsets, hatch-to-hatch spacing and contour-to-contour spacing
  + If a part is replicated in the same build, however, each copy can be assigned individual pairs of hatch/contour strategies
* The fixed sets of scan strategies per by part are applied without respect to local geometry, x/y/z thicknesses, layer height, over/underhangs or other conditions
* The same laser is used for all hatches of a particular part on all layers, irrespective of part size
* All build files are generated prior to start of the build. During the build, XML files for upcoming layers are not regenerated or altered based on sensor feedback from current and prior layers

### Potential machine limitations

The scanpath code conducts basic (“sanity check”) analyses on the configuration file to highlight infeasible conditions. Such conditions include referencing a nonexistent (undefined) scan strategy, calling for an STL file which cannot be found, or specifying a zero or negative scan velocity

However, since the code is intended to be machine-independent, it places few restrictions on the range of machine parameters specified in a configuration file. Such “unrestricted” parameters include the number of available lasers, min/max layer thickness, maximum laser power and scan velocity, and maximum build dimensions (x/y range and z-height). End-users must ensure that their XML build files comply with their specific machine limitations

Laser ID’s (the tag used in XML scan files to indicate use of a specific laser on a particular segment) are intended to be semi machine-independent via the use of an ID-to-serial-number lookup file utilized by the machine controller. Thus, the build configuration file should typically refer to lasers by generic ID’s such as 1, 2, 3 or 4

# Scanpath generation overview

## End-to-end process

Creating a physical part proceeds in several steps as indicated in the list below and diagram that follows. The baseline scanpath code executes the first three steps:

Step 1: Generate layer files (in XML layer format) based on STL files. One XML layer file is created per layer

Step 2: Generate scanpath files (in XML scan format) from the layer files. One XML scanpath file is created per layer

Step 3: Zip up the scanpath files and change the extension of the resulting archive to “.scn” to form a scan package for LabVIEW

Step 4: Run the ALSAM LabVIEW code to control a machine based on the given scanpaths. This step is not part of the OASIS Challenge, and will be conducted by GE and OASIS staff

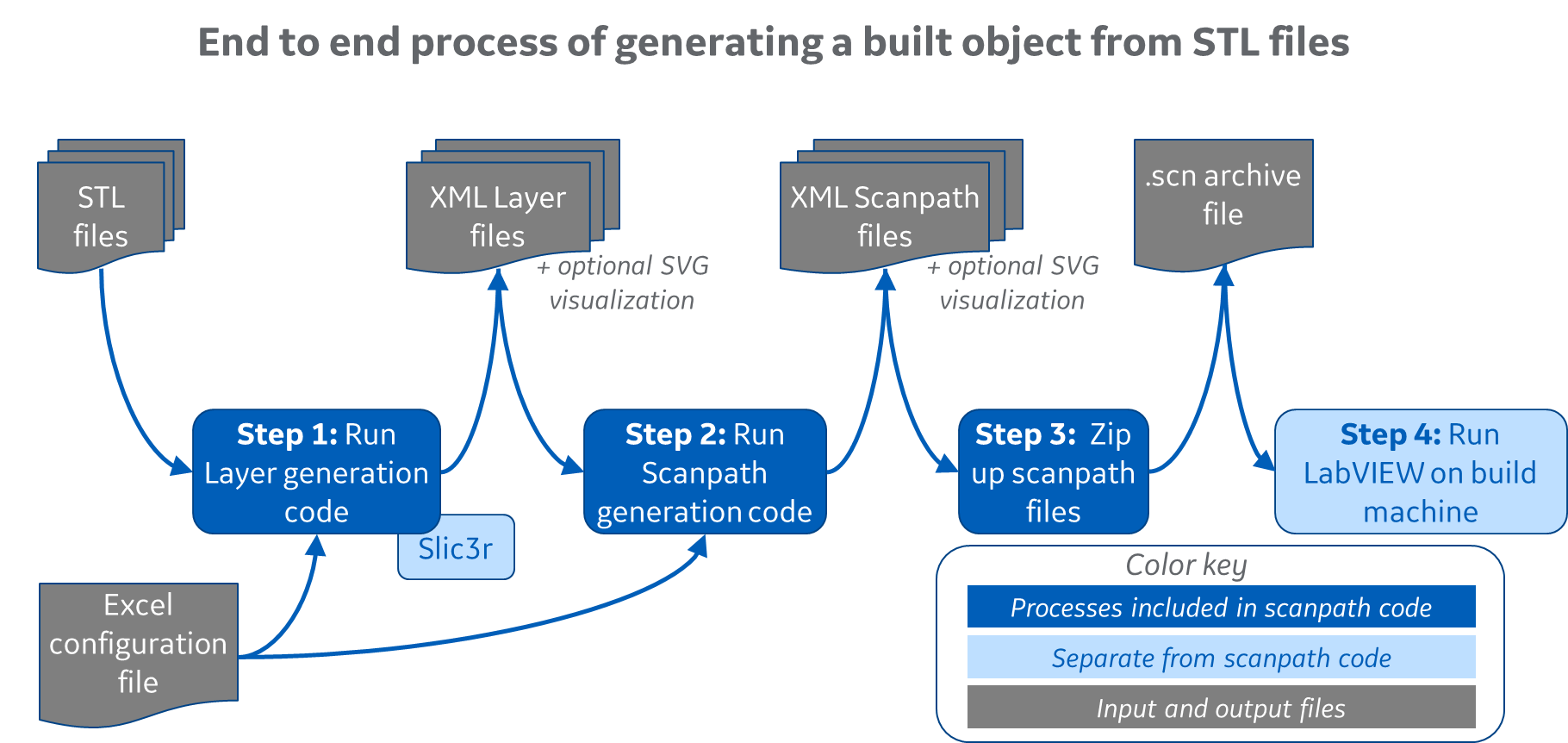


Figure 1 End-to-end of generating a built object from STL files

The XML layer files created in step 1 are not needed after step 2 is completed; the files can be deleted or retained to generate alternate sets of scanpath files. The baseline code allows you to execute any combination of steps 1-3 with the understanding that you must have (or be creating) a set of XML layer files before XML scan files can be generated

In addition to XML outputs, you may opt to create additional SVG (structured vector graphics) files which can be used to quickly visualize individual layer polygons or scanpath outputs in any web browser

## Requirements to execute the baseline code

This list does not include requirements to recompile (or modify) the baseline code, only to execute the pre-compiled binaries:

* A Windows environment
  + This is a requirement of the compiled executables, which perform numerous system calls for file navigation and other functions, rather than a restriction of the XML scan schema or machine controller
  + The executables have not been tested under non-Windows environments
* Copies of the three executables (createScanpaths.exe, genLayer.exe, genScan.exe), placed in a common folder. These may be downloaded from the PBFAM or OASIS GitHub sites via the Multilaser\_Pre-compiledBinaries repository or zip file
* Download of the slic3r 1.3.0 package from www.slic3r.org and unpacking its contents into the slic3r folder included in the precompiled-binaries executable folder
  + This package is not included in source code due to its license details and size (it’s considerably larger than the ALSAM scanpath package)
* An Excel configuration file describing the build
  + This file has a specific format and version number. Several examples and a blank file are included in the precompiled-binaries repository
  + Excel 97-03 (.xls) format is used, because this format is compatible with open-source editors such as Apache OpenOffice

# Quick start: Download and run some examples

## Set things up on your machine

a) Download (at least) the precompiled binaries folder from the OASIS GitHub at <https://github.com/AmericaMakes/OASIS-Challenge-Baseline-Code-and-Models>. You can download the OASIS baseline source code folder as well, but it is not required to run the baseline code. The new folder can be placed whatever you’d like on your machine

b) Download Slic3r version 1.3.0 from <https://dl.slic3r.org/win/Slic3r-1.3.0.64bit.zip>

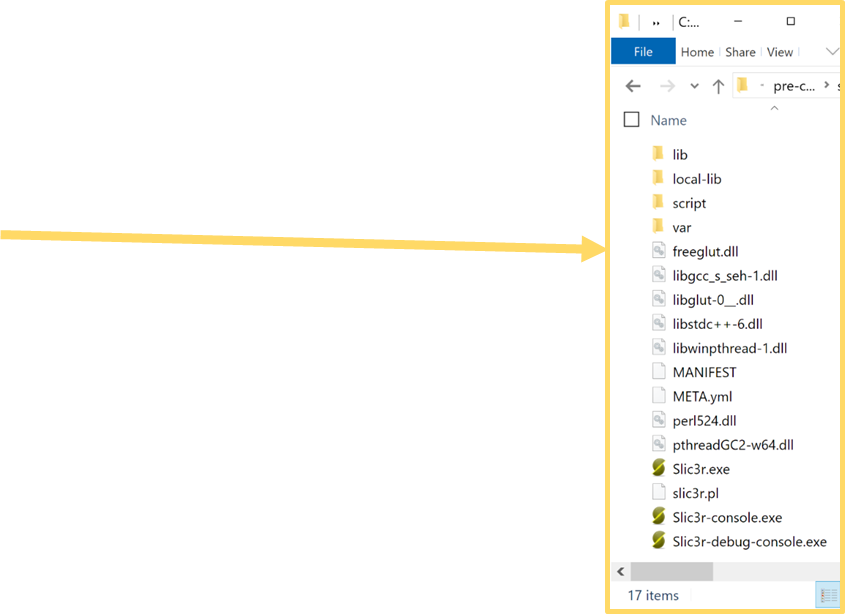
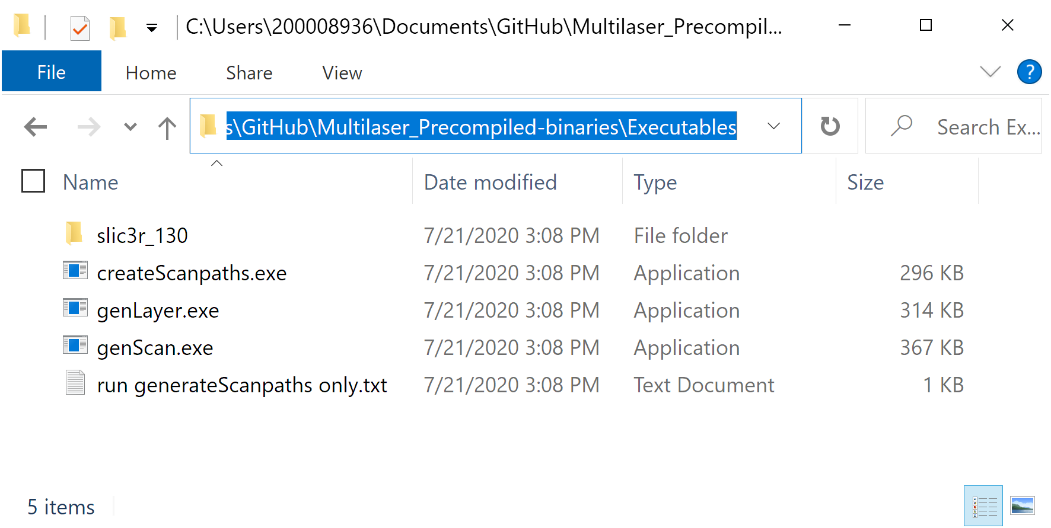
c) Unzip the Slic3r package into the slic3r\_130 folder that appears in the Oasis baseline precompiled binaries\Executables folder created in the previous step. The slic3r executables and four subfolders should now appear in OASIS baseline precompiled binaries\Executables\slic3r\_130

Figure 2 Structure of the Executable folder after downloading Slic3r

## Generate sample builds

Several examples are provided in OASIS baseline precompiled binaries/Project Examples folder, along with a blank template for your use. The outputs of both layer and scanpath generation are included so that you can view XML and SVG results and verify that your execution returns the same results

Folder “A original examples – updated format” contains the original two examples in updated format:

**OASIS Ex1 - many parts.xls** generates a build files populated with multiple 1mm x 1mm cubes, and demonstrates most features of the Excel configuration file and XML schema

**OASIS Ex2 - coin.xls** generates build files describing a single 40mm-wide coin on top of circular supports. This build contains more detailed (interesting) scan lines but is still fairly compact in size and generation runtime

Folder “B additional examples” contains two new examples:

**OASIS Ex3 - larger parts.xls** generates a few parts of somewhat larger dimension (30-50mm wide, up to 60mm high) to evaluate several hundred layers at a time

**OASIS Ex4 - singlestripes plus parts.xls** demonstrates the single-stripe tab, which may be used to conduct bead-on-plate experiments for parameter development (not used in the OASIS Challenge; this is simply a shared example to the original America Makes codebase)

To generate any of the examples, execute *createScanpaths.exe* in the Executables folder. You can also create a shortcut to this file and place it on your desktop or other convenient location. Execution and user choices will proceed as follows:

1. Double-click on createScanpaths.exe to open a file-select window
   1. Navigate up and over to the Project Examples folder and select any of the Excel configuration files. Click cancel to quit without generating anything
   2. The select window will only permit you to select Excel files ending in .xls
2. The code will quickly scan the config file for errors, such as missing or conflicting identifiers and values out of range
   1. If an error is found, the code will identify the error, write the information to an error-report file and then quit (after giving you a chance to read the text and press a key). The error report will be created in the same folder as the config file you selected
   2. If no errors are found, the code next determines whether any layer and/or scan outputs exist in the project output folder listed in the config file. This is used to determine which output-generation options (layer-only, layer-plus-scan, scan-only) are available
3. You are then presented with a list of output-generation options. Input your selection (in upper or lower case) and press Enter. The options may include
   1. L = generate layers files only
   2. S = generate scan files only (only available if the project output folder already contains layer outputs)
   3. B = generate both layer and scan files (layers first, then scans, of course)
   4. C = cancel without overwriting or deleting anything
4. If your choice involves re-generating or overwriting something which currently exists in the project output folder, you’ll be asked whether to overwrite (delete) the existing output or merge with it. Merge will still overwrite any files of the same name, but will retain other prior output
5. The process will then create (or erase and recreate) the project output folder and/or layer and scan subfolders, and then proceed with file generation
6. If your choice involves layer generation, the process proceeds in two sub-steps
   1. Slice all the indicated STL files into layers, contained in one or more new SVG files
   2. Generate XML and SVG layer files (layer by layer) and place them into subfolders of the layer folder
7. If your choice involves scan generation, the process iterates layer-by-layer until done and then (optionally) creates a zip file with the XML scan output

Before or after generating the examples, you may view the (pre-)generated results by navigating to any of the example folders, then into a project output folder, then into a LayerFiles or ScanFiles folder

* The XMLdir subfolder scan files, which can be viewed in an XML editor such as Notepad++
* The SVGdir subfolder contains visualization files, which are not required for the actual build. Double-clicking on an SVG file will likely give you the option to view the file in a web browser (Chrome does this well). The SVG files display a fixed 2000 pixel-wide view of the populated build surface, which may omit small details and hatching that is present in the XML build files

The sample Excel files can be used as a guide to develop and generate your own build using the existing or alternate STL files. Both text and binary STL files are suitable and may be mixed in the same build

# Setting up a new build

## General considerations

Part inputs must in in STL format (ASCII or binary). It is fine to include files with differing formats in the same build. Part files must appear in the same folder as the configuration file, unless you supply a full path in the filename field

Each segment in the output scanpath file will be given a *segment style* which defines a combination of laser speed, power and spot size. You must pre-define all segment styles to be used in the build on the appropriate tab of the configuration file

* There is no particular limit to the number of such rows, and therefore the number of permutations, which may be defined
* Segment styles are referenced by region profiles on tab 5, which define inter-segment offsets and other items. A segment style may be referenced by any number of regions or be unused. Contours and hatches of a particular region may use the same or differing segment styles
* You don’t need to create segment styles specifically for the jump segments which will be interspersed between hatches and contour polygons. The baseline code will auto-generate one “jump” segment style for each region profile on tab 5, and apply that style to that region’s jump segments

Each segment style references a *velocity profile*, which defines laser travel speed and various delays used for that type of segment

* As with segment styles, all velocity profiles used in the build must be defined on the velocity profile tab
* Each profile may be referenced by any number of multiple segment styles, or be unused

The concept of “regions” was introduced as a simple means to assign contour and hatch strategies to each part. In this context, “scan strategy” refers to a complete combination of segment style, velocity profile and inter-segment parameters such as hatch spacings

* Regions combine segment styles with parameters which apply only to sets of segments, not to a single segment in isolation. These include items such as the contour offset from the part, hatch-to-hatch spacing and hatch angle
* The baseline code provides minimal granularity in assigning scan strategies because each part is considered a single region. Only one contour strategy and one hatch strategy is applied to all its geometric areas
* Regions are not part of the XML scan format, which could permit the location and segment style of each of thousands of hatches to be defined separately. More advanced scanpath generation methods could take advantage of this extreme granularity to utilize multiple regions per part as selected by geometry or other means

Trajectories allow build items to be grouped and ordered, but are optional

* The LabVIEW machine code proceeds through the build file in order of appearance. With this in mind, the baseline code generates scanpaths in order of trajectory number (lowest first). Each part is permitted one contour trajectory and one hatch trajectory, which may be the same or different. Trajectories are integers greater than zero
* Trajectories values may be left blank, in which case the contours and/or hatches of that part will be generated after all other parts on the layer
* Within a particular trajectory number, the baseline code will group segments of the same contour/hatch type and region profile into paths. For example, if the hatches and contours of all parts are given the same trajectory number, and all parts are given the same region profile, all scanpaths will appear in a single trajectory containing two separate paths (a hatch path and a contour path). However, if one part has a different region profile, its hatches will appear in the same trajectory but as a third path (a separate hatch path)
* Refer to the later section titled ‘Mapping configuration-file “regions” to trajectories and paths’ for explanation of the baseline code handling of trajectories
* During the build, all scanpaths of the current trajectory must be completed before any paths or segments of the next trajectory will begin

## Recommended navigation through config-file tabs

The baseline code provides error checking which will catch general issues such as referencing an undefined segment style ID, as described in the next segment. Many errors can be avoided with a careful process, and there are two general ways to proceed

* The “bottom up” method shown in Figure 3 is relatively foolproof, since you can ensure that all references and ID’s exist. In this method, proceed across the configuration file from left to right to first create your velocity profiles, then the segment styles that use those profiles, then the region profiles that use those styles, then finally listing the parts that use those region profiles. At each step, the drop-down selection for referenced ID’s will only let you select what’s been created
* However, the “top down” method may be somewhat more intuitive. In this method, you would start by listing the parts involved, and progress to create the required region profiles, segment styles and velocity profiles. In a complex build it may be easy to miss a reference, but the error-check will let you know

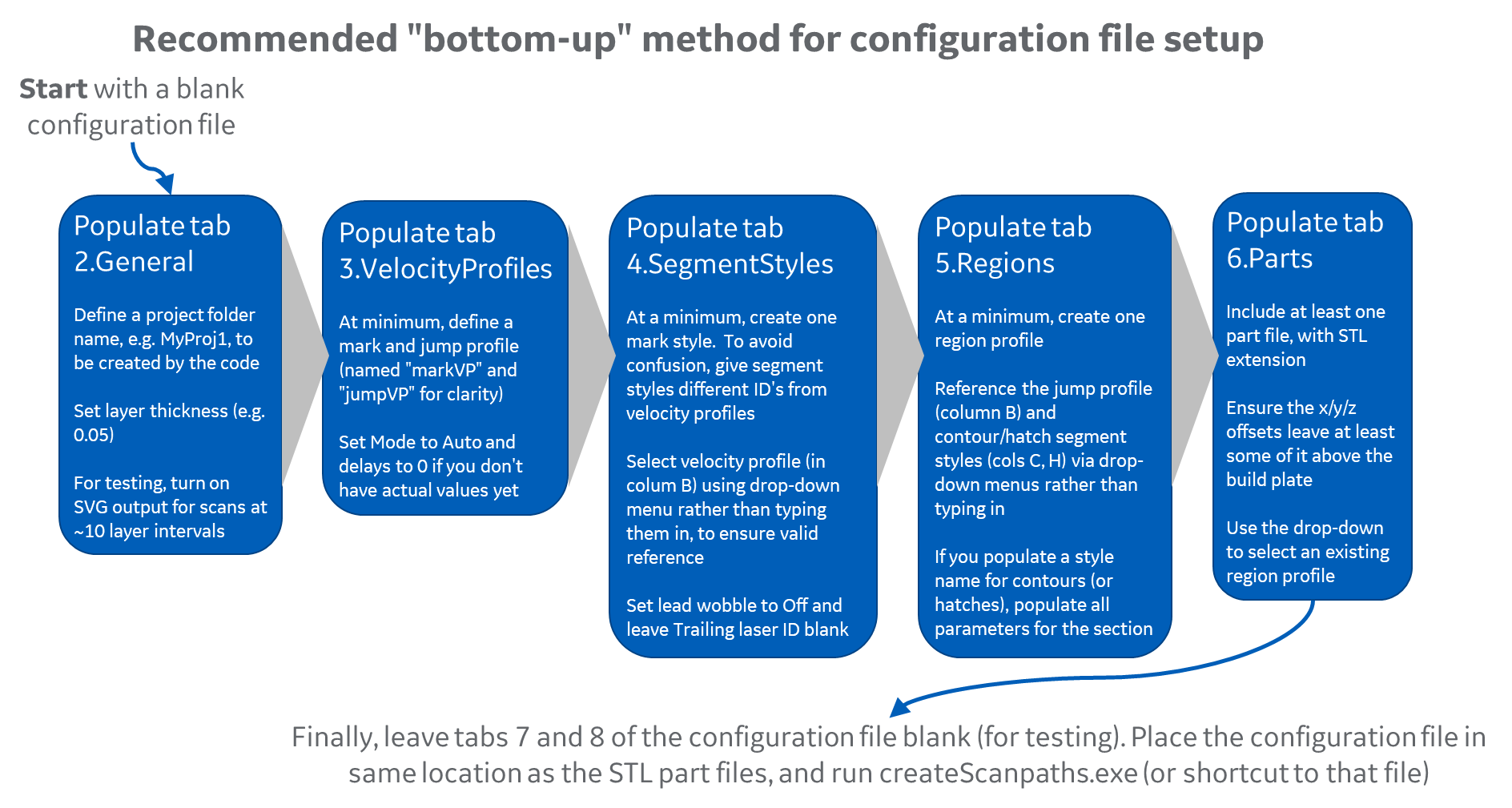


Figure 3 Recommended “bottom-up” method for configuration file setup

## Typical configuration file errors

This list is organized by tab in reverse order since errors typically appear on the “higher level” tabs which reference lower level tabs

Potential issues on Tab 6.Parts

* + File extension not included (e.g. myPart instead of myPart.stl) or is not .stl
  + File is not present in same folder as configuration file
  + Offsets are too large, placing the part off the build plate. These are in units of millimeters
  + The region profile listed does not exist on tab 5. Profile names are not case-sensitive
  + Blank line between two parts. This is interpreted as end of list, so the part(s) below are ignored

Potential issues on Tab 5.Regions

* + Two regions share the same name except for capitalization. Profile ID’s are case insensitive, so the duplication is an error
  + The velocity profile for jumps (column B) does not exist on tab 3. This should not be a segment style reference to tab 4. It’s merely a reference a velocity profile. The baseline code will create a “jump” segment style containing this velocity profile
  + The contour and/or hatch segment styles do not exist. It’s ok to leave either the contour or hatch styles blank (which will omit that type of segment from the region), but the ID’s must be found on tab 4 if they’re populated
  + Offsets or spacings are too large. These fields are in units of millimeters, as are the part X/Y/Z offsets
  + The cumulative contour offsets (from multiple contours) plus the hatch offset may be so large that small segments of the part disappear. This isn’t an error; it’s actually an expected outcome

Potential issues on Tab 4.SegmentStyles

* + Two rows share the same ID, differentiated by capitalization. The baseline code is case-insensitive, so these are treated as duplicates
  + A velocity profile (column B) does not exist on tab 3
  + Spot size is too large or small. Values are in microns (unlike most other fields, which are in millimeters)

Potential issues on Tab 3.VelocityProfiles

* + Two rows share the same ID, differentiated by capitalization. The baseline code is case-insensitive, so these are treated as duplicates
  + Delay values are too large or small. This field is in units of microseconds, and typical values are +/- 500uS or less
  + Mode is something other than Auto or Delay. The current machine code only accepts these two values

Potential issues on Tab 2.General

* + Project folder is blank or points to a drive number that cannot be accessed
  + Layer thickness is too large. This field is in millimeters
  + Starting or ending scan layer values exclude some layers of the build. To generate all layers, set starting layer to 1 and ending layer to -1

## Specifications and bounds

Dimensions

* + Millimeter-denominated values include the desired layer thickness via tab 2 of the configuration file, X/Y/Z positioning of each part via tab 6, scan velocities listed on tab 3 and contour/hatch spacings and offsets on tab 5
  + Micron-denominated items include spot size on tab 4

SVG coordinates

* + SVG visualization files created by the code display the build area, centered on areas containing parts
  + Positive X proceeds to the right and positive Y proceeds downward

Build plate dimensions and coordinates

* + The scanpath code is not aware of the dimensions of the target build plate, so there is no check that all parts fall on the plate
  + The code is also not aware of the plate location of coordinate 0,0 for the target scanning system (whether at upper left or in middle of plate, for instance), so it is up to the user to determine this and offset each part appropriately via tab 6 of the configuration file
  + Some trial and error may be required to determine the generated X,Y coordinates of a particular STL file when placed at scanpath coordinate 0,0. This will be resolved in further refinement of the scanpath code and documentation

Layer thickness

* + Layer thicknesses from 0.001 mm (1 microns) to at least 5 mm can be generated by the current release of the Slic3r package
  + The minimum layer thickness that has been extensively tested is 0.01 mm (10 microns). Even smaller values are likely feasible, but 0.01 mm should be sufficient for all current build systems

Maximum number of layers

* + This is not known, but builds containing several thousand layers have been generated with no issues
  + To reduce generation speed for very large builds, disable or reduce the frequency of SVG-file generation via tab 2 of the configuration file

# Configuration file reference

## Tab 1.Header

The open fields on this tab are for your own reference and documentation. They are not read or processed by the baseline code. The config-file format version in cell B2, which is locked, is referenced by the code. As of August 2020, the baseline code supports file version 3 (only)

## Tab 2.General

This tab sets up global parameters for the build, such as defining a project folder and layer thickness. Of the items on this tab, only layer thickness and dosing factor actually affect build outcome

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Cell(s)** | **Format** | **Range** | **Usage** |
| Project folder | C4 | Text |  | Baseline code will create an output folder with this name, in the same location as the configuration file. May instead be a full pathname starting from root directory |
| Layer thickness | C6 | Real, mm | >0 | Thickness at which each part will be sliced. All layers will be of this constant thickness. Values of 0.020 to 0.150 mm are common |
| Dosing factor | C7 | Real | >0 | Amount of extra powder to be applied to each layer, as a multiplier on the layer thickness. Values of 1.5 and higher are typical |
| Replace string ID's? | C8 | Yes/No |  | If Yes, all ID's for velocity profiles and segment styles will be replaced with auto-generated integers, which will reduce the size (but also readability) of the output scan XML files |
| Create zip file? | C9 | Yes/No |  | If Yes, scan generation will create a zip archive in the output folder containing all scan XML files. Its extension will be .scn, compatible with the open-source machine controller |
| Create SVG for layers? | C14 | Yes/No |  | If Yes, layer generation will create one or more SVG files which visualize the inner and outer part contours for a particular layer. This increases the time for layer generation |
| Interval for layer SVG generation | C15 | Integer | >=1 | If layer SVG's are enabled by the prior parameter, this value indicates the layer interval between SVG files. 1 = create SVG for all layers, 2 = every other layer, 5 = every 5th layer, etc. Larger values lead to faster generation times at the loss of visualization granularity |
| Create SVG for scans? | D14 | Yes/No |  | If Yes, scan generation will create one or more SVG files which visualize the output scanpaths for a particular layer. This increases the time for scan generation |
| Interval for scan SVG generation | D15 | Integer | >=1 | If scan SVG's are enabled by the prior parameter, this value indicates the layer interval between scan SVG files. Operates the same as Interval for layer SVG, above |
| Starting scan layer | C18 | Integer | >=1 | If set to 1 (always the bottom-most layer), scan-file creation begins at build plate height. If set to a value higher than (layer#) 1, layers below this layer number will be omitted. This may be useful if only certain layers need to be regenerated |
| Ending scan layer | C19 | Integer | -1 or >=C18 | If set to -1 (default), scan files will be generated for all layers between the starting layer (the preceding parameter) and the top of all parts. If set to a specific value >0, scan file creation will stop after completing that layer (or when reaching the end of all parts, whichever is reached first) |

## Tab 3.VelocityProfiles

This tab populates the VelocityProfileList section of the XML scan files and is referenced by tabs 4 and 5. All combinations of laser speeds and delays used in the build must be pre-defined on this tab

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Cell(s)** | **Format** | **Range** | **Usage** |
| Velocity profile ID | Column A | Text | Must be unique | Names a profile which can be referenced by one or more segment styles on tab 4 |
| Velocity | Column B | Real, mm/s | >0 | Laser travel speed |
| Mode | Column C | Text | Delay, Auto | Auto should be sufficient for most purposes |
| Laser-on delay | Column D | Real, uS | Positive, zero or negative | Per RTC5 docs, delay from scanner arrival at segment to start of laser marking |
| Laser-off delay | Column E | Real, uS | Positive, zero or negative | Per RTC5 docs, delay from scanner departure from segment to end of laser marking |
| Jump delay | Column F | Real, uS | Positive, zero or negative | Per RTC5 docs, additional delay to be added prior to beginning an inter-segment jump |
| Mark delay | Column G | Real, uS | Positive, zero or negative | Per RTC5 docs, additional delay to be added prior to beginning mark segment |
| Polygon delay | Column H | Real, uS | Positive, zero or negative | Per RTC5 docs, additional delay prior to be added prior to marking successive segments of a polygon |

## Tab 4.SegmentStyles

This tab populates the SegmentStyleList section of the XML scan files and is referenced by tab 5 (Regions). SegmentStyle parameters apply to segments in isolation and exclude inter-segment spacings and offsets which appear on tab 5. All combinations of segment-level laser parameters used in the build must be pre-defined on this tab. The baseline code adds an auto-generated jump style for each region on tab 5, which is applied to jump segments between individual hatches and contour polygons

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Cell(s)** | **Format** | **Range** | **Usage** |
| Velocity profile ID | Column A | Text | Must be unique | Names a profile which can be referenced by one or more segment styles on tab 4 |
| SegmentStyle ID | Column A | Text | Must be unique | Names a style which can be referenced by one or more regions on tab 5 |
| Velocity Profile | Column B | Text | Must match an item on tab 3 | Must reference a velocity profile listed on tab 3, to be used in this segment style |
| Lead laser ID | Column C | Text | Typically integer, e.g. 1 | Defines lead laser for marks in this style (can leave blank for jump styles, which will reference most recently-used laser). Set to 1 for OASIS Challenge. Interpreted by machine files |
| Lead laser power | Column D | Real, Watts | >=0 | Mark power. If omitted, zero power is assumed |
| Lead laser spot size | Column E | Real, um | >0 | Laser spot size in microns, dependent on machine capabilities |
| Lead laser wobble | Column F | Yes/No |  | Yes enables wobble for this strategy |
| Lead wobble frequency | Column G | Real, Hz |  | Only used if wobble enabled. See RTC5 docs |
| Lead wobble shape | Column H | Integer | 0,1,2 | Only used if wobble enabled. See RTC5 docs  The remaining parameters (in gray) are not utilized within the OASIS challenge.  Set Lead laser wobble to "No" and leave the Trailing Laser section blank |
| Lead wobble transverse amplitude | Column I | Real, mm |  | Only used if wobble enabled. See RTC5 docs |
| Lead wobble longitudinal amplitude | Column J | Real, mm |  | Only used if wobble enabled. See RTC5 docs |
| Trailing laser ID | Column K | Text |  | If populated, defines a lead-follow scan strategy |
| Trailing laser sync offset | Column L | Integer, uS | >0, in 10uS intervals | Trailing laser delay. Must be >0 in intervals of 10 uS |
| Trailing laser power | Column M | Real, Watts | >=0 | Power for trailing laser |
| Trailing laser spot size | Column N | Real, um | >0 | Spot size for trailing laser |
| Trailing laser wobble | Column O | Yes/No |  | Yes enables wobble for this strategy, for trailing laser (independently of lead laser) |
| Trailing wobble frequency | Column P | Real |  | Only used if trailing wobble enabled |
| Trailing wobble shape | Column Q | Real, Hz | 0,1,2 | Only used if trailing wobble enabled |
| Trailing wobble transverse amplitude | Column R | Integer |  | Only used if trailing wobble enabled |
| Trailing wobble longitudinal amplitude | Column S | Real, mm |  | Only used if trailing wobble enabled |

## Tab 5.Regions

This tab defines sets of contour and hatch scan strategies which combine segment styles with inter-segment spacings and offsets. Different sets could be applied to various regions of a part based on geometry or other factors. The baseline code, however, treats each part as a single "region" and applies one contour/hatch set to all areas of the part

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Cell(s)** | **Format** | **Range** | **Usage** |
| Region Profile ID | Column A | Text | Must be unique | Names a regional scan strategy. Baseline code references just one style for each part on tab 6 |
| Velocity profile for jumps | Column B | Text | Must match an item on tab 3 | Must reference a velocity profile listed on tab 3, which will be used for contour and hatch jumps in this region. Baseline code auto-generates a segment style to be referenced by these jump segments |
| Contour segment style | Column C | Text | Blank or item on tab 4 | If blank, contours will be omitted in this region. Otherwise, must reference a segment style listed on tab 4 which indicates laser parameters to be referenced by all contour segments in this region |
| Number of contours | Column D | Integer | >=0 | Number of concentric contours, starting from outer/inner edges of part and progressing toward the interior |
| Contour offset from part | Column E | Real, mm | >=0 | Positive offset indents the first contour toward interior of the part |
| Contour-to-contour spacing | Column F | Real, mm | >0 | Center-to-center spacing between successive contours (toward interior) |
| Contour skywriting mode | Column G | Integer | 0,1,2,3 | See RTC5 documentation |
| Hatch segment style | Column H | Text | Blank or item on tab 4 | If blank, hatches will be omitted in this region. Otherwise, must reference a segment style listed on tab 4 which indicates laser parameters to be referenced by all hatch segments in this region |
| Hatch offset from contours | Column I | Real, mm | >=0 | Positive offset indents hatches toward interior of the part from the final (inner-most) contour |
| Hatch-to-hatch spacing | Column J | Real, mm | >0 | Center-to-center spacing between successive hatches |
| Hatch skywriting mode | Column K | Integer | 0,1,2,3 | See RTC5 documentation |
| Hatch scheme | Column L | Integer | 0,1 | 0=build hatches using global grid overlay without respect to part geometry; may add excessive jumps between widely-spaced areas under the same profile. 1=attempt to minimize build time by grouping marks to minimize jumps |
| Initial hatch angle | Column M | Real, degrees | -360 to 360 | Hatch angle for layer 1; positive=counter-clockwise |
| Inter-layer hatch rotation | Column N | Real, degrees | -360 to 360 | For all layers, this and prior parameter are combined to determine hatch angle as:  ( (Initial angle) + (Layer number -1)\*(Inter-layer rotation) ) mod 360 |

## Tab 6.Parts

Each row on this tab represents a separate part to be included in the build, as represented by an ASCII or binary-formatted STL file. Parts may be duplicated by including them on multiple rows. X/Y/Z offsets permit parts to be spaced or joined as desired. Contour and hatch trajectory numbers indicate the grouping and build ordering of parts and their regions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Cell(s)** | **Format** | **Range** | **Usage** |
| Filename | Column A | Text | \*.stl | Names an STL part file in ASCII or binary format. Files must be in same folder as the configuration file, unless full path is provided. Parts may be duplicated (with different offsets) by listing them on multiple rows |
| X translation | Column B | Real, mm | +/- (build plate size/2) | X translation applied to this instance of this part. Orientation is system dependent, but +X is often to the right |
| Y translation | Column C | Real, mm | +/- (build plate size/2) | Y translation applied to this instance of this part. Orientation is system dependent, but +Y is often to the "top" |
| Z translation | Column D | Real, mm | +/- (build plate size/2) | Z translation applied to this instance of this part. -Z offset moves the part down in relation to the build plate; +Z moves it up. If the part is offset with +Z such that it does not contact the build plate, one or more layers will exclude the part (until it intersects the build area) |
| Region profile | Column E | Text | Must match an item on tab 5 | Must reference a region profile listed on tab 5. The baseline code applies that set of contour and hatch strategies to all areas of this part |
| Contour trajectory number | Column F | Integer | >0 | If populated, indicates the build priority of the contours of this part. The baseline code generates scanpaths in trajectory order (lowest first). Multiple parts (or contours/hatches of a part) may share the same trajectory number; their scanpaths will be built as a group. If left blank, a contour trajectory number of 9998 is applied |
| Hatch trajectory number | Column G | Integer | >0 | If populated, indicates the build priority of the hatches of this part, which may be the same or different from its contour trajectory number. To build hatches after contours, set hatch trajectory numbers higher than contour numbers. If left blank, a hatch trajectory number of 9999 is applied |

## Tab 7.PathProcessing

*This tab is not used in the OASIS Challenge and may be left unpopulated*

In multi-laser systems, this tab controls whether scanpaths with the same trajectory (utilizing different lasers) should be processed concurrently or in series. This tab has no function in a single-laser system, which must process all scanpaths in series. If a trajectory is omitted from this tab, its paths will be built in series. Note: concurrent operation requires that the paths within the trajectory utilize different lasers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Cell(s)** | **Format** | **Range** | **Usage** |
| Trajectory number | Column A | Integer | >0 | Should reference a contour or hatch trajectory number defined on tab 6. Any "new" trajectory number listed here will be ignored |
| Path processing mode | Column B | Text | Sequential, Concurrent | Indicates how the regions within this trajectory should be built, in a multi-laser system. Sequential = complete each region before starting the next. Concurrent = begin the next region as soon as the appropriate laser is free. To achieve concurrency, the trajectory must include multiple regions which use different lasers, such as contours from several parts which have different region profiles |

## Tab 8.Stripes

*This tab is not used in the OASIS Challenge and may be left unpopulated*

This tab may be used to define single-segment laser "stripes" for parameter exploration purposes. Each stripe may be given a separate segment style, although they will share a common inter-stripe jump profile and skywriting mode. Stripes are built prior to "actual" parts listed on tab 6, and must therefore be given negative trajectory numbers (since real parts use trajectories >0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Cell(s)** | **Format** | **Range** | **Usage** |
| Inter-stripe jump velocity profile | C5 | Text | Must match an item on tab 3 | Must reference a velocity profile listed on tab 3, to be used in this segment style |
| Stripe skywriting mode | D5 | Integer | 0,1,2,3 | See RTC5 documentation |
| Trajectory number | Column A | Integer | <0 | The first blank entry in this column indicates the end of the stripe list. Otherwise, indicates the build priority of this stripe (lowest numbers are built first) |
| Stripe number | Column B | Text |  | For user reference; not read or utilized by the baseline code |
| Segment style | Column C | Text | Must match an item on tab 4 | Must reference a segment style listed on tab 4 which indicates laser parameters to be used for this stripe |
| Starting X coordinate | Column D | Real, mm | +/- (build plate size/2) | The stripe extends from starting to ending coordinate |
| Starting Y coordinate | Column E | Real, mm | +/- (build plate size/2) |  |
| Ending X coordinate | Column F | Real, mm | +/- (build plate size/2) |  |
| Ending Y coordinate | Column G | Real, mm | +/- (build plate size/2) |  |
| Stripe height | Column H | Real, mm | >=0 | If >0, indicates a desired absolute height over the build plate, which enables multi-thickness experiments. The layer number closest in height to this value (based on layer thickness) will be used. Values of 0 and [layer thickness] will both result in build on layer 1 |

# XML Layer format

## Overview

The XML layer format is an intermediate step between STL file slicing and scanpath generation. The baseline code generates XML layer files for all layers before beginning scanpath creation. Use of layer files may allow more advanced scanpath algorithms to consider the geometry of nearby layers (up and down) when selecting one or more region profiles for a specific layer

Sample layer files may be found in the precompiled binaries folder under any of the examples, in <ProjectFolder>\LayerFiles\XMLdir

Each XML file covers a single layer and consists of three top-level fields: Thickness (a single field), VertexList and Slice

*Thickness* indicates the z-thickness of the layer, in mm

*VertexList* lists all vertices (intersection points) on all contours

*Slice* contains one or more Regions, which are closed contours identified with a specific part file. The layer file contains a single slice which lists regions from all parts present on the layer. An individual part, if it appears on a particular layer, may have any number of regions based on the inner/outer contour content of that height in its STL file

Each region is defined by the following fields:

* Type: Inner or Outer, indicating whether the region encloses actual part (Outer) or an interstitial space (Inner)
* Tag: Alphanumeric value that identifies the region profile assigned to this part in the configuration file. Region profiles define hatch and contour scan strategies, offsets, spacings and other parameters. Note that the tag does not necessarily link the region to a specific part, since multiple parts may share the same region tag (i.e. you may not be able to tell which part generated a particular region)
* contourTraj (contour trajectory): Integer value that indicates the build order and grouping of any contours that will be built from this region. Traj = Trajectory, which is a grouping of scanpaths. Trajectories will be built in numerical order, lowest first. If multiple regions (hatches and/or contours from one or multiple parts) have the same contourTraj number, they will be built as a group. If no contour trajectory value is assigned to a part in the config file, its regions will receive contourTraj = 9998
* hatchTraj (hatch trajectory): Integer value indicating the build order and grouping of any hatches that will be built from this region. Interpretation is analogous to hatchTraj, above. The configuration file permits separate trajectory numbers to the contours and hatches of each part. If no hatch trajectory value is assigned to a part in the config file, its regions will receive hatchTraj = 9999

Following the four fields above, each Region contains a list of Edges (line segments) which are defined in terms of Start/End vertex numbers. The vertex numbers don’t appear in the layer file; they are determined by counting

## Slic3r

The Slic3r package ([www.slic3r.org](http://www.slic3r.org)) is used to slice each STL file into one or more layers. Source code is not included in this solution; end-users are instructed to download the executable and unpack it in a specific folder

# XML Scan file content

The XML scan format is fully described in “ALSAM3024 multiLaser XML schema 2020323.docx.” The following sections briefly note the major top-level areas of the schema, and how these items are defined in the baseline scanpath code. Care has been taken to differentiate schema requirements from baseline code implementation, but if in doubt about actual schema capabilities, please refer to the schema document above

### Header

This first section of the schema contains a few elements that help identify the file and its position in the build

AmericaMakesSchemaVersion is currently 2020-03-23 (last update point)

LayerNum should match the number in the filename, but it typically not evaluated

LayerThickness (in mm) may be varied from layer-to-layer, although the baseline code uses a constant thickness as indicated in the configuration file

AbsoluteHeight may be useful in cases where LayerThickness varies. The baseline code, however, computes this simply as LayerNum \* LayerThickness

DosingFactor is taken directly from the configuration file

BuildDescription is intended as free text, although it may be useful to list min/max ranges of various parameters used in the build file, such as laser power

### VelocityProfileList

A VelocityProfile (VP) in the XML scan schema defines a specific combination of laser velocity and delays. All such combinations used in the layer must be listed and defined in the VelocityProfileList section which precedes the list of segment. A VP combined with other laser parameters defines a SegmentStyle, as described in the next section

The list of VP’s which genScan outputs to the build file will be identical to the list defined by the user on the VelocityProfiles tab of the configuration file. No VP’s will be omitted, even if they are not used in the entire build or a particular layer

Any number of VelocityProfiles may be created, to permit very granular control of build parameters. Each VP may be reused in multiple SegmentStyles. If a SegmentStyle calls for a VP which does not exist, this will cause an error in either scanpath generation or machine execution. However, it is fine to define VP’s which are not used

### SegmentStyleList

Each line segment in the scan file, both mark and jump, must be assigned a SegmentStyle (SS). A SegmentStyle defines a combination of laser ID, laser power, velocity (via reference to a VelocityProfile) and other parameters

All SegmentStyles used in a particular layer must be defined in the SegmentStyleList section of the XML scan file. In creating this list, genScan will start by including all user-defined styles on the SegmentStyles tab of the configuration file. To this list, genScan will add auto-generated “jump” SegmentStyles corresponding to the jump VelocityProfile indicated for each RegionProfile listed in the configuration file. This is simply a convenience so that the user does not need to create both a VelocityProfile and SegmentStyle for jumps; only a VelocityProfile

Regarding jump segments: SegmentStyles intended for jumps (power=0) can be simplified by omitting the laser power parameter; this will be interpreted as zero power. Further, if the laser ID parameter is omitted, it will be assumed that the laser used in the immediately-preceding segment is to perform the jump (per build file order, in the same path). In this way, as few as one generic “jump” SegmentStyle may be used in combination with any number of multi-laser mark styles

Finally, it should be clear that SegmentStyles (by themselves) do not indicate geometric parameters such as hatch spacing or offset. Geometric parameters are implemented via the relative positioning of different segments, whereas SegmentStyles simply indicate what to do when encountering a particular type of line segment

### TrajectoryList

The final major section of the XML scan file contains the actual line segments, grouped into trajectories and paths

A trajectory defines one or more paths (groups of related segments) which must be fully completed before the next trajectory can be started. Trajectories will be built in the order they appear in the XML scan file, irrespective of TrajectoryID (but note that genScan uses trajectory numbers to determine how to order things when generating the XML)

Each trajectory contains a number of key fields

* TrajectoryID is for reference and does not affect the build
* PathProcessingMode defines how the various paths within the trajectory are to be built, sequentially (in appearance order) or in parallel, based on the trajectory’s PathProcessingMode parameter. Parallel building will only be realized if there is more than one path, and if the segments of these paths utilize different lasers (via their SegmentStyles); otherwise, the paths will be built sequentially

Each path within the trajectory contains the following items:

* Type is either contour, hatch or single\_stripes (note that the baseline code separates contours and hatches from each part into separate paths, although they may still be in the same trajectory)
* Tag corresponds to the RegionProfile (from the config file) which was used to select parameters for all segments in the path. A RegionProfile defines both hatch and contour parameters, so the baseline code will assign the same tag for contours and hatches of a particular part, even though the actual SegmentStyles and geometric values may differ between contour and hatch paths
* SkyWriting mode is taken from the configuration file as 0, 1, 2 or 3
* NumSegments indicates the total segments in the path. Miscounting will result in an error upon build
* Start indicates the starting coordinate of the first vertex
* Segment defines the SegmentStyle (abbreviated SegStyle) and endpoint of each segment (which is also the starting point of the following segment). Note that segments are defined in a “point to point” format which mimics a continuous contour; if the path is actually to be a combination of alternating marks and jumps, the SegStyle’s must simply be alternated

## Mapping configuration-file “regions” to trajectories and paths

The Excel configuration file utilized by the baseline scanpath code includes a level of abstraction compared to the XML scan schema, because this code does not implement the full segment-by-segment parameter flexibility provided by the schema. In particular, the scanpath code (via the configuration file) assigns a single RegionProfile to each part, which defines a fixed set of contour and hatch spacings and parameters to be used across the part

The schema is agnostic to parts and regions, and is organized as:

TrajectoryList🡪Trajectory(s)🡪Path(s)🡪combinations of (Segment, SegmentStyle)

In contrast, the baseline code implements a flow-down from a part (which has no analog in the XML schema), to a region profile (which also has no analog in the schema, due to its complete flexibility), and ending in paths and segments, as:

Part P🡪RegionProfile R, hatch trajectory number h, contour trajectory number c

RegionProfile R🡪hatch and contour parameters

Hatch path🡪region tag R, segments all using SegmentStyle R[hatches]

Contour path🡪region tag R, segments all using SegmentStyle R[contours]

### Grouping parts and regions into trajectories

To generate trajectories, the baseline code groups together all paths with the same trajectory value. In the example above, contours and hatches for part P might appear in the same or different trajectory depending on the user-assigned contour and hatch trajectory values. Following identification of all trajectory values, genScan cross-indexes all regions (i.e. contours or hatches of specific part) by trajectory number

### An example

As an example, suppose the configuration file defines parts as shown in Table 1, which is a simplified view of the Parts tab including region profile and trajectory assignments

|  |  |  |  |
| --- | --- | --- | --- |
| **Part (filename)** | **RegionProfile tag** | **Contour trajectory** | **Hatch trajectory** |
| Part 1.stl | RP1 | 3 | 3 |
| Part 2.stl | RP2 | 3 | 2 |
| Part 3.stl | RP1 | 3 | 1 |

Table 1 Sample Parts table, for trajectory generation example

And suppose the configuration file defines RegionProfiles as shown in Table 2. Note that hatch parameters are blank for profile RP2, meaning that any region using this tag is should not be hatched

|  |  |  |  |
| --- | --- | --- | --- |
| **RegionProfile** | **Jump velocity profile** | **Contour params** | **Hatch params** |
| RP1 | VP1 | <parameters> | <parameters> |
| RP2 | VP2 | <parameters> |  |

Table 2 Sample RegionProfile table, for trajectory generation example

Combining information from Tables 1 and 2, the baseline scanpath code will identify a total of two “active” trajectories and create trajectories and paths as shown in Table 3. Trajectory 2 does not appear because its only region would be hatches for profile RP2, which are omitted. In addition, regions within a trajectory are grouped into paths based on equivalent types (contour or hatch) and RegionProfile tag values

|  |  |
| --- | --- |
| **TrajectoryID** | **Paths and tags within the trajectory, in order they will appear in this trajectory** |
| 1 | Path 1 = {Part 3 hatches}, tag RP1 |
| 3 | Path 1 = {Part 1 contours + Part 3 contours}, tag RP1  Path 3 = {Part 1 hatches}, tag RP1  Path 3 = {Part 2 contours}, tag RP2 |

Table 3 Results of trajectory generation example

List of Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Definition** |
| ALSAM | Acceleration of Large Scale Additive Manufacturing, an America Makes initiative |
| AM | Additive Manufacturing (or AmericaMakes.us) |
| DMLM | Direct Metal Laser Melting, typically an alternate terminology for PBFAM |
| DOM | Microsoft Domain Object Model, a package for XML tree creation and output via msxml6.h |
| PBFAM | Powder-Bed Fusion Additive Manufacturing |
| STL | Stereolithography file, which portrays a three-dimensional shape in terms of triangular sections |
| SVG | Structured Vector Graphics, a visualization format |
| VS | Microsoft Visual Studio |

List of XML Scan Schema Terminology

This is duplicated from ALSAM3024 multiLaser XML schema 2020323.docx

|  |  |
| --- | --- |
| **Schema term** | **Description and usage** |
| ***AbsoluteHeight*** | Absolute build height of a particular layer. Not evaluated by the code |
| ***AmericaMakes***  ***SchemaVersion*** | XML schema version, which LabVIEW uses to determine the appropriate parser |
| ***BuildDescription*** | Free text which may be used to indicate expected conditions or required capabilities, such as maximum power and velocity occurring in the build |
| ***DosingFactor*** | Indicates the depth of powder to be applied during recoat, as a multiplicative factor of LayerThickness  1.5 = 50% additional dose (1.5 x layer height) |
| ***End*** | Ending coordinate of the current Segment |
| ***Header*** | Collection of parameters which do not fall under the lists of velocity profiles, segment styles or trajectories |
| ***LaserMode*** | Defines the mode of operation when multiple lasers (Traveler sections) are included in a particular SegmentStyle. May be omitted when <2 Traveler sections are included within the style   * Independent = multiple lasers operating on separate build segments. In this mode of operation, PathProcessingMode (below) is referenced for sequencing information * FollowMe = multiple lasers traveling in synchronized fashion along the same set of segments:   + The TravelerID having SyncDelay=0 (required) is the Master; all other (Slave) TravelerID sections must include a SyncDelay > 0 in reference to the Master laser   + Each TravelerID can have separate power, SpotSize and wobble settings   + The indicated VelocityProfileID is used for all lasers, with the Master laser leading |
| ***LayerThickness*** | Indicator of the thickness of this build layer in the system’s defined units |
| ***LayerNum*** | Optional numbering from bottom to top layer. Not evaluated by the code, so may be non-sequential |
| ***Path*** | Set of related scan paths (Segments) which make up a specific aspect of the build. Typically, a part should be divided into multiple Paths such that one path might include all the contours, and another includes all the hatches. All Segments within a particular Path should utilize the same laser(s) to avoid synchronization and timing issues within the Path. Parts can be subdivided into as many Paths as desired |
| ***PathProcessing***  ***Mode*** | Determines how multiple Paths within a particular TrajectoryID will be sequenced   * Sequential (default if omitted): Paths will be processed in the order in which they are listed. Each successive Path will begin building only after its predecessor is completed. This mode assures that the laser(s) used within a particular TrajectoryID are not subject to timing uncertainties (if the same laser(s) are used in multiple Paths) but may result in a longer build * Concurrent: Paths will be built concurrently as permitted by laser availability. This mode will typically be used when each Path is assigned to a different laser. In general, any Paths which utilize unique lasers will be initiated immediately, but timing after any Path is completed will depend on the availability of each laser and cannot be guaranteed |
| ***Power*** | Laser marking power in watts. If the TravelerID section is omitted from a SegmentStyle, Power will be set to 0 and the SegmentStyle is assumed to be a jump |
| ***Segment*** | An individual unit of laser mark or jump between two points. The first Segment in a Path begins at the Path’s Start coordinate and continues to that Segment’s End coordinate (which is also the starting point for the next Segment). The laser(s) which actually carry out the mark or jump are determined by referencing the SegmentStyleID:   * If the SegmentStyle indicates one or more TravelerID’s, these lasers carry out the Segment * If there is no TravelerID listed in the SegmentStyle, the Segment is assumed to be a jump. Power is set to zero and the laser(s) used in the immediately preceding Segment carry out the jump. Therefore, it is improper to begin a Path with a jump segment unless the SegmentStyle specifies a TravelerID to make the jump |
| ***SegmentStyle*** | A set of parameters which defines a single mode of mark or jump operation. Each SegmentStyle must include a VelocityProfileID and may (optionally) include LaserMode and one or more TravelerID (laser parameter) sections:   * Jumps may omit the TravelerID section. In this case the “jumping” laser(s) will assumed to be the laser(s) used in the immediately prior Segment within a particular Path. The laser’s Power will be set to zero during the jump. This may reduce the number of styles needed, since a single jump style can be utilized for all lasers * If LaserMode is omitted or set to Independent, only one TravelerID section should be included in the SegmentStyle * If LaserMode is set to FollowMe (synchronized), at least two TravelerID sections must be included. See LaserMode and TravelerID for further details |
| ***SegmentStyleList*** | List of one or more segment styles. Only those styles included in the build’s SegmentStyleList may be referenced by the build |
| ***SegStyle*** | SegmentStyle ID to be applied to a particular segment. Name truncated to reduce XML file size |
| ***SkyWritingMode*** | Optional laser motion mode. If omitted or set to 0, will be disabled. See Scanlab RTC5 documentation for details |
| ***SpotSize*** | Laser spot size value in microns |
| ***Start*** | Starting coordinate of the first Segment of the Path   * All Segments are assumed to be contiguous from the End of the previous Segment, in contour fashion. Each Segment specifies only an End coordinate, rather than a separate Start/End for each Segment * To create non-contiguous hatches, set up alternating “mark” and “jump” segments by choosing a different SegmentStyles for each type of Segment |
| ***SyncDelay*** | Indicates the delay in microseconds between a particular Slave laser and the Master laser in “FollowMe” LaserMode. Should be omitted if LaserMode is set to Independent or if the SegmentStyle is a jump. Each Slave laser’s SyncDelay is absolute with respect to the Master, independent of any other Slave lasers which may be synchronized to the same Master. The Scanlab RTC5 supports delays in increments of 10 microseconds only |
| ***Trajectory*** | Grouping of related scan paths for one or more lasers   * Each TrajectoryID may contain multiple Paths, which may each be processed by the same or different lasers as controlled by SegmentStyles and PathProcessingMode * If there are multiple Trajectories within the layer, they will be processed sequentially. The first Trajectory must complete before the second can begin, irrespective of the lasers used by each Trajectory. Any elements which are to be built concurrently should be included in the same Trajectory |
| ***TrajectoryList*** | Contains all the scan paths for the layer. The individual trajectories will be built strictly sequentially in the order that they appear. Within a trajectory, however, individual paths (scan path groupings) may be built either sequentially or concurrently as defined by the trajectory’s PathProcessingMode value |
| ***Traveler*** | Section of a SegmentStyle which identifies and defines parameters for one specific laser. If a SegmentStyle utilizes multiple lasers, it should include multiple Traveler sections. Traveler:ID should be the system’s reference to a specific laser, such as “1” (serial numbers should not be used, to avoid machine-dependent files; a lookup from scanfile ID’s to serial numbers will be provided as part of the end-user machine controller) |
| ***Type*** | Indicates whether the Path consists of hatches or contours. Informational only; does not affect parameters |
| ***VelocityProfile*** | Metrics which defines a single mode of laser travel, including linear speed and various delays |
| ***VelocityProfile***  ***List*** | List of one or more velocity profiles. Only those profiles included in the build’s VelocityProfileList may be referenced by the build |
| ***Wobble*** | Optional mode of laser marking which adds an oscillating motion independent of laser travel speed. See Scanlab RTC5 documentation for details |