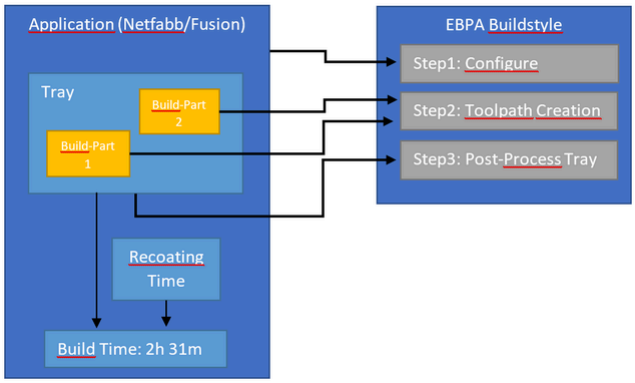
Build-Time Estimation

Making a reasonable build-time estimation for print jobs is a very important part of every buildstyle project. The buildstyle project's code is as near to the machine and its operation, as you can get from a software point of view. Therefore, most parts of the estimation are done within the buildstyle's JavaScript functions.

The build-time estimation for a specific machine may depend on one or more of these parameters:

- Constant, generic machine preferences (i.e. setup and recoating time).
- Layer depended machine preferences (i.e. layer exposure time).
- Toolpath lengths and speed (i.e. how much time does the machine need to follow the hatch pattern on each layer).
- Travel time in between toolpaths, to move from a toolpath end position to the next toolpath start position.

The following diagram shows the workflow and interaction between the Application and an EBPA buildstyle. The information is reduced to focus on the things which are relevant for build time estimation.



The EBPA buildstyle is responsible for creating the toolpaths and therefore also for estimating the time which is spent to process the toolpaths on each layer.

There are two types of time estimation generated by the EBPA. The first one is related to each individual build part and does not consider traveling time in between different parts on the tray. The second time estimation is considering all build parts on the tray including traveling time in between build parts as well as considering the path order which may be changed/optimized at step 3.

For the tray build time the application is using the second more precise time-estimation and may add time which is spent in addition to the toolpath processing time, e.g. the recoating time for each layer.

The first per-part time estimation is used to build up a statistic of build parts with certain properties (volume, size, etc.) and the corresponding build time reported by the EBPA. With this information the application can guess a build time for a certain EBPA based on the properties of a 3d build part without calculating toolpaths.

Based on this the application can instantly present a guessed build time estimation for a tray based on 3D build parts and the selected EBPA. After toolpaths are created this guess will be replaced by the more precise toolpath-based time estimation.

Step 1: Configure

There are a couple of script functions which are called by the application in order to configure the buildstyle. In order to achieve a good build time estimation, it is important to get Step 3 right. And this needs some configuration in the first place.

`configurePostProcessingSteps` is an application entry function which is called to let the buildstyle define its requirements for post-processing a tray later in step 3.

```
/**
* @param  a_config    bsPostProcessingConfig
*/
exports.configurePostProcessingSteps = function(a_config)
{
  // 1. Sorting the toolpaths and calculate build time for each layer
  a_config.addPostProcessingStep(
    postprocessLayerStack_MT,{bMultithread: true, nProgressWeight: 10});

  // 2. Collecting the resulting build times from all layers
  // and store the total time on the tray.
  a_config.addPostProcessingStep(
    postprocessLayerStack_ST,{bMultithread: false, nProgressWeight: 1});
}
```

In this example we are defining two script functions which will be called by the application for post-processing.

The first function is intended to calculate the build time individually for each layer of the tray. If a sorting of toolpaths is wanted, e.g. to reduce the jump length from one path to the next, then this should be done before analyzing all toolpaths for build time calculation. For a better performance, this function will be called in parallel for all available layers.

The second function is used only to sum up the build time values from each layer and then store the result as a property of the tray. This is necessary just because the first calculation step is done in parallel and therefore it cannot be used to calculate and store the final sum.

The implementation of these two functions is discussed in Step 3.

Step 2: Toolpath creation

`makeExposureLayer` is an application entry function which is called to calculate the toolpaths for the layers on the tray. The function is called many times, once for each layer of each part on the tray. It is also called in parallel which means that multiple threads are executing this function at a time, but never for the same layer.

The following example is reduced to a minimum just to show the concept of build time estimation. In real life the toolpaths will be more complex.

```
/**
* Calculate the exposure data / hatch vectors for one layer of a part
* @param  a_modelData    bsModelData
* @param  a_hatchResult  bsHatch
* @param  a_nLayerNr     Current layer to be processed
*/
exports.makeExposureLayer = function(a_modelData, a_hatchResult, a_nLayerNr)
{
  var layer_toolpaths = new HATCH.bsHatch();

  // process all islands on the layer
  var island_it = a_modelData.getFirstIsland(a_nLayerNr);
  while(island_it.isValid())
```

```
  {
    let hatch_paths = new HATCH.bsHatch();
    island_it.getIsland().hatch(hatch_paths, 0.3, 45, 0);
    layer_toolpaths.moveDataFrom(hatch_paths);
    island_it.next();
  }

  // process all open polylines on the layer
  var polyline_it = a_modelData.getFirstLayerPolyline(
    a_nLayerNr, POLY_IT.nLayerOpenPolylines);
  while(polyline_it.isValid())
  {
    let hatch_paths = new HATCH.bsHatch();
    polyline_it.polylineToHatch(hatch_paths);
    layer_toolpaths.moveDataFrom(hatch_paths);
    polyline_it.next();
  }

  // melting at 1.5 m/s
  let melt_time = layer_toolpaths.getExposureLength() / 1500.0;

  // traveling from path end to next path start at 6 m/s
  let skip_time = layer_toolpaths.getSkipLength() / 6000.0;

  // Additional delay of 0.002 seconds for each skip
  let skip_count = layer_toolpaths.getSkipCount() + layer_toolpaths.getHatchBlockCount();
  let skip_delays = skip_count * 0.002;

  // reporting estimation for a single layer
  HATCH_INFO.addExposureTimeSeconds(
    melt_time + skip_time + skip_delays);

  a_hatchResult.moveDataFrom(layer_toolpaths);
};
```

In this example the build time calculation is based on some basic toolpath properties. Much more complexity could be added here to achieve a better estimation.

`addExposureTimeSeconds` is called to report the build time for that one layer of the given build part. This value can be used by the application to get a time estimation for a single part or for single layers of a part.

In step 3 we will see how the build time is calculated for the whole tray while considering all the build parts on the tray and traveling time in between parts.

Step 3: Post-process tray

In step 1 two functions have been specified which are called to postprocess the toolpaths on the tray. The first one will be called multi-threaded, so it underlies the same restrictions as *makeExposureLayer* except that this time all build parts on the tray are available and all the toolpaths are already calculated.

```
/**
 * Multithreaded post-processing. This function may be called
 * several times with a different layer range.
 * @param  a_modelData        bsModelData
 * @param  a_progress         bsProgress
 * @param  a_layer_start_nr   Integer. First layer to process
 * @param  a_layer_end_nr     Integer. Last layer to process
 */
var postprocessLayerStack_MT = function(
  a_modelData,
  a_progress,
  a_layer_start_nr,
  a_layer_end_nr)
{
  a_progress.initSteps(a_layer_end_nr - a_layer_start_nr + 1);

  for(let layer_nr = a_layer_start_nr; layer_nr <= a_layer_end_nr; ++layer_nr)
  {
    var exposure_array = a_modelData.getLayerPolylineArray(
      layer_nr, POLY_IT.nLayerExposure, "rw");

    // Sort array by bounding box in Y direction
    // and assign processing order
    exposure_array.sort(function(a,b){
      return a.getBounds().minY - b.getBounds().minY
    });
    for(var i=0; i < exposure_array.length; ++i){
      exposure_array[i].setAttributeInt('_processing_order', i);
    }

    let melt_length = 0.0;
    let skip_length = 0.0;
    let skip_count = exposure_array.length;
    let laser_pos = {x:0.0,y:0.0};

    for(var i=0; i < exposure_array.length; ++i)
    {
      skip_count += exposure_array[i].getSkipCount();
      skip_length += exposure_array[i].getSkipLength();
      melt_length += exposure_array[i].getExposureLength();
      skip_length += exposure_array[i].getXYCoord(0).distance(laser_pos);
      laser_pos = exposure_array[i].getXYCoord(exposure_array[i].getPointCount()-1);
    }

    // melting at 1.5 m/s
    let melt_time = melt_length / 1500.0;

    // traveling from path end to next path start at 6 m/s
    let skip_time = skip_length / 6000.0;

    // Additional delay of 0.002 seconds for each skip
    let skip_delays = skip_count * 0.002;

    // Write tray attribute using a unique name for each layer
    a_modelData.setTrayAttribEx("BTd0elf2-" + layer_nr,
      melt_time + skip_time + skip_delays);
```

```
    a_progress.step(1);
  }
};
```

The function is calculating the time for each layer and storing it as a tray attribute where the attribute-name contains the layer number.

The layer time is calculated after the toolpaths have been sorted (a simple sort in this example) because the resulting time depends on the order of toolpaths.

The second post-process function is iterating all layers, reading the previously stored tray attribute and stores the total tray build time as another tray attribute.

The per-layer values are no longer needed so they are removed by this function.

```
/**
 * Single threaded post-processing. This function will be called only once
 * with the full layer range
 * @param  a_modelData        bsModelData
 * @param  a_progress         bsProgress
 * @param  a_layer_start_nr   Integer. First layer to process
 * @param  a_layer_end_nr     Integer. Last layer to process
 */
var postprocessLayerStack_ST = function(
  a_modelData,
  a_progress,
  a_layer_start_nr,
  a_layer_end_nr)
{
  var total_layer_time = 0.0;

  a_progress.initSteps(a_layer_end_nr - a_layer_start_nr + 1);

  var layer_iter = a_modelData.getPreferredLayerProcessingOrderIterator(
      a_layer_start_nr, a_layer_end_nr, POLY_IT.nLayerExposure);

  while(layer_iter.isValid() && !a_progress.cancelled())
  {
    var layer_nr = layer_iter.getLayerNr();

    var layer_time = a_modelData.getTrayAttribEx("BTd0elf2-" + layer_nr);

    if(layer_time !== undefined)
    {
      total_layer_time += layer_time;

      a_modelData.delTrayAttrib("BTd0elf2-" + layer_nr);
    }
    else{
      throw new Error("Layer time undefined in layer " + layer_nr);
    }

    layer_iter.next();
    a_progress.step(1);
  }

  // reporting estimation for the whole tray
  a_modelData.setTrayAttrib(
    'built_time_estimation_ms', (total_layer_time*1000.0).toFixed()
  );
}
```

Parent topic: Concepts