MCP Integration Testing

Short Technical Test

Milan Platisa

# Introduction

The Materialise Control Platform is an eco-system of products that seamlessly work together:

* Operator User Interface
* Controller
* Build Processor

The controller is an embedded electronic platform that must be built into the additive manufacturing machine to control and steer the machine. The controller runs embedded software that interconnects with the user interface to visualize the machine’s status and progress during a build. The controller software receives the machine toolpaths from the build processor.

## Operator User Interface

The operator user interface serves as a visual human-machine interface and runs on a Windows platform. Typically this is a touch panel computer, built into the machine. The user interface shows the available jobs on the embedded controller and allows an operator to start a job. During the job build, the progression is tracked on a 3D visual. The operator can pause, cancel/stop the build or he can cancel a part from the build. He cannot alter the build in any way, however.

The operator interface is linked via FTP to retrieve the jobfiles from the controller, via OPC-UA to read variables and via webservices to trigger methods on the controller.

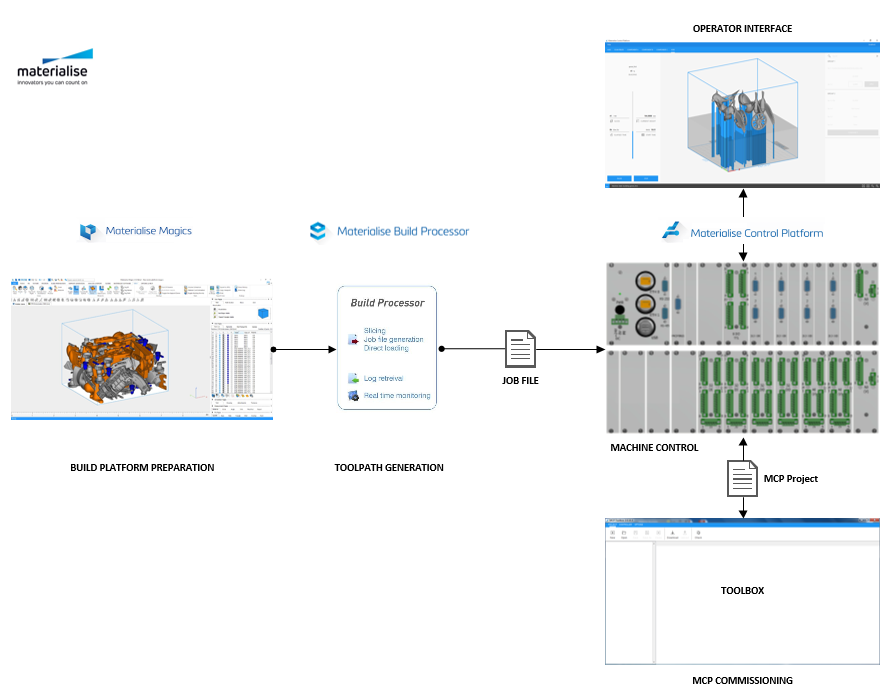
## Controller

The controller runs a real-time Linux distribution. This OS hosts our embedded application that actually controls the motion, heating, atmosphere and scan/laser processes in additive manufacturing machines. There are three technologies supported: stereolithography, selective laser sintering and selective laser melting.

To cope with the technological differences, the control application exposes a programmable layer within the software. This layer can be programmed in “Lua” (<www.lua.org>). An extensive API is available for customers to program the machine behavior as they need.

## Build Processor

The build processor converts the 3D CAD data to sliced and hatched data or “toolpaths”. These toolpaths compensate beam diameter, material crimp, … and contain the ideal process parameters such as laser power, laser speed and laser beam diameter. The build processor stores the toolpaths in a so-called “job file” and uploads this jobfile via FTP to the embedded controller.



# Test

## Webservice interface

The MCP web service exposes the MCP public object methods and properties through a RESTful web service server running on the embedded target. The RESTful web services use HTTP as the communication layer to communicate with the web service host. The RESTful web services used in the MCP application use [JSON](https://www.w3schools.com/js/js_json_intro.asp) to encode the input and output arguments of a function.

The web service method to get a complete list of all object names is a HTTP GET request with URL:

http://Address:Port/MCP

The MCP web server is configured to use port 8050.

An example of the get request output can be found in the attached “webservices.example” file.

### Querying objects for details

Object details can be requested by appending the object name to MCP separated by '/' (HTTP GET request):

http://Address:Port/MCP/ObjectName

Where ObjectName is an element from the ‘all object names’ list (Example see “webservices.example”).

### Task

Create an application or a script in a language or your choice that reads all the channel details from an MCP server. Come up with your own ideas to do version control & management.

## Lua Unit Framework

To test Lua MCP API we are using Lua Unit framework (<http://luaunit.readthedocs.io/en/latest/>)

A trimmed version of a MCP test repository can be found in the “scipts.zip” archive:

* "UnitTests/mcp-entry-scripts/runnerSystem.lua" 🡪 a script to run all MCP System unit test (in this case only "UnitTests\MCP-System\testCom.lua")
* "UnitTests/helpers" contains common and utility scripts
* "ThirdParty” folder contains different third party tools (like luaunit)

### Task

UnitTests\MCP-System\testCom.lua implements a test for ***COM.Init*** API function (described in 30.17.1 "COM.Init" in the "mcp\_manual.txt").

Based on this example implement tests for the following API functions (described in the mcp\_manual.txt)

* 30.17.8 COM.SetDataBits
* 30.17.9 COM.SetBaudRate
* 30.17.24 COM.SetStopBits

Additional question: the current implementation of stty funcion (unitCom.lua) only works on Linux targets. How we extend this test to support Windows targets?