



작성자: hcchoi@spirallab.co.kr

샘플 코드 및 GitHub 저장소: <https://github.com/labspiral/sirius3>

문서 이력

- 0.9.3 (2025.12.5) 초안 작성

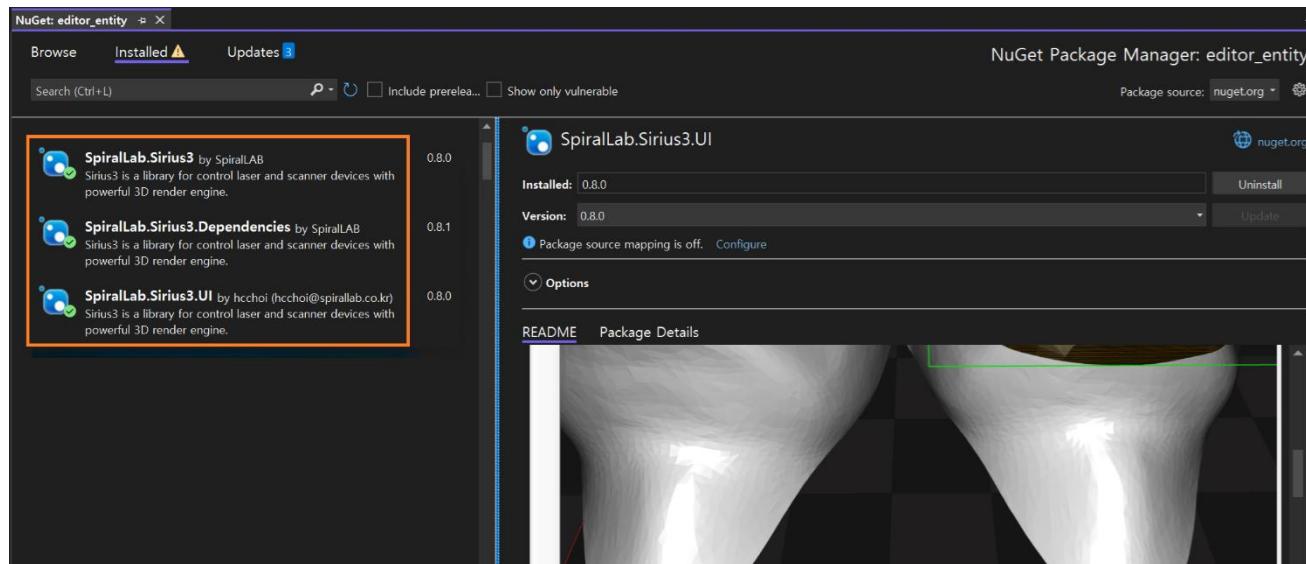
목차

1. 라이브러리 설치	5
2. 라이브러리 초기화 (Pseudo code)	7
3. 문서(Document) 와 기능들	8
4. 편집기(Editor)	10
5. 기본 개체들(Primitive Entities)	13
6. 제어 개체들(Control Entities)	14
7. 펜 속성 (Pen Properties)	15
7.1 레이어 펜(Layer Pen) 속성 항목	16
7.2 개체 펜(Entity Pen) 속성 항목	16
8. 계측 (Measurement)	17
8.1 레이어 펜(Layer Pen) 속성의 Skywriting 기능을 사용한 결과	18
8.2 개체 펜(Entity Pen) 속성의 Wobble 기능을 사용한 결과	19
9. 마커(Marker)	20
9.1 MarkerRtc 객체	20
10. 수동(Manual)	23
11. 스캐너(Scanner)	24
12. 레이저(Laser) 소스	26
12.1 레이저 소스 제어부 구현	26
12.2 레이저 소스 UI 구현	26
13. 디지털 입출력 (DIO)	28
14. 파워메터(PowerMeter)와 파워맵(PowerMap)	29
15. 해치(Hatch)	32
15.1 선분 해치	32
15.2 폴리곤 해치	36
15.3 코드를 통한 해치 처리	39
16. 슬라이서 (Slicer)	41

17. 텍스트 데이터 변환 (Text Converter)	44
17.1 이벤트 (Event)	44
17.2 단순 스크립트 (Simple Script)	45
17.3 파일 (File)	45
17.4 오프셋 (Offset)	46
18. 그룹(Group) 및 블록(Block)	47
19. 웨이퍼(Wafer), 기판(Substrate) 맵(Map)	49
20. syncAXIS (XL-SCAN) 사용법	50
21. 환경 설정(Config)	54

1. 라이브러리 설치

시리우스 3 라이브러리는 마이크로 소프트의 NuGet 패키지 설치 관리자를 이용해 최신 버전의 라이브러리 설치 및 업데이트가 지원됩니다.



라이브러리는 총 3 개의 DLL 파일로 구성되어 있으며 각각 다음과 같은 기능을 제공합니다.

- **SpiralLab.Sirius3.Dependencies**
 - SCANLAB RTC4/5/6 관련 프로그램 및 DLL 파일들
 - syncAXIS 관련 프로그램 및 DLL 파일들
 - 사용자 폰트 및 샘플 파일들
 - 런타임에 필요한 필수 파일들
- **SpiralLab.Sirius3**
 - 스캐너, 레이저, 파워메터와 같은 제어 장치들에 대한 구현부
- **SpiralLab.Sirius3.UI**
 - 문서 및 기본 개체, 가공용 개체, 제어용 개체 제공
 - 쉐이더 및 3D 렌더링 엔진 제공
 - 윈폼(WinForms) 기반 컨트롤 제공

지원 플랫폼

- .NET Framework 4.8.1 이상
- .NET8.0-windows 이상
- (기타) 윈도우 10.11 이상, 64 비트 환경, OpenGL 3.3 이상 드라이버 설치

사용자 프로젝트 설정

- NuGet 관리자를 통해 3 개의 DLL 이 참조되도록 설치(혹은 업데이트) 합니다.
- 기타 의존성 있는 DLL 파일들(Microsoft.Extensions.Logging, OpenTK, Newtonsoft.Json 등)을 참조합니다.
- 스캐너(IScanner), 레이저 소스(ILaser), DIO, 파워메터(IPowerMeter), 마커(IMaker) 등의 제어 객체를 생성 및 초기화 합니다.
- SiriusEditorControl 사용자 컨트롤을 생성시키고, 제어 객체를 서로 연결(Assign) 합니다.

(참고) 인증 정보가 포함된 USB 동글기가 없을 경우 최대 30 분 동안 평가판 모드(Evaluation Copy Mode)로 실행되며, 일부 기능의 사용이 제한될 수 있습니다.

2. 라이브러리 초기화 (Pseudo code)

```

// Initialize sirius3 library
SpiralLab.Sirius3.Core.Initialize();

// Create scanner device
var fov = 100.0; // 100mm
var kfactor = Math.Pow(2, 20) / fov; // 20 bits resolution
Var rtc = ScannerFactory.CreateRtc5(0, kfactor, Yag1, ActiveHigh, ActiveHigh,
@“C:\...\cor_1to1.ct5”);
rtc.Initialize();
rtc.CtlFrequency(50_000, 2); // default frequency, pulse width: 50KHz, 2usec
rtc.CtlSpeed(500, 500); // default jump, mark speed: 500mm/s

// Create D.IO devices
var dInExt1 = IOFactory.CreateInputExtension1(rtc);
dInExt1.Initialize();
var dOutExt1 = IOFactory.CreateOutputExtension1(rtc);
dInExt1.Initialize();
// and more ... 

// Create laser source device
var laser = LaserFactory.CreateVirtualDutyCycle(0, 20.0, 0, 99);
laser.Scanner = rtc;
laser.Initialize();
laser.CtlPower(20 * 0.1);

// Create powermeter and power map
var powerMeter = PowerMeterFactory.Create...
powerMeter.Initialize();
var powerMap = PowerMapFactory.Create...
laser.PowerMap = powerMap;

// Create marker controller
var marker = MarkerFactory.CreateRtc(0);
marker.Initialize();

// Assign devices into editor
siriusEditorControl.Scanner = rtc;
siriusEditorControl.Laser = laser;
siriusEditorControl.PowerMeter = powerMeter;
siriusEditorControl.Marker = marker;

// Do marker as ready status
marker.Ready(siriusEditorControl.Document, siriusEditorControl.View, rtc, laser,
powerMeter);

// Your application is running ...

// Clean-up sirius3 library
SpiralLab.Sirius3.Core.Cleanup();

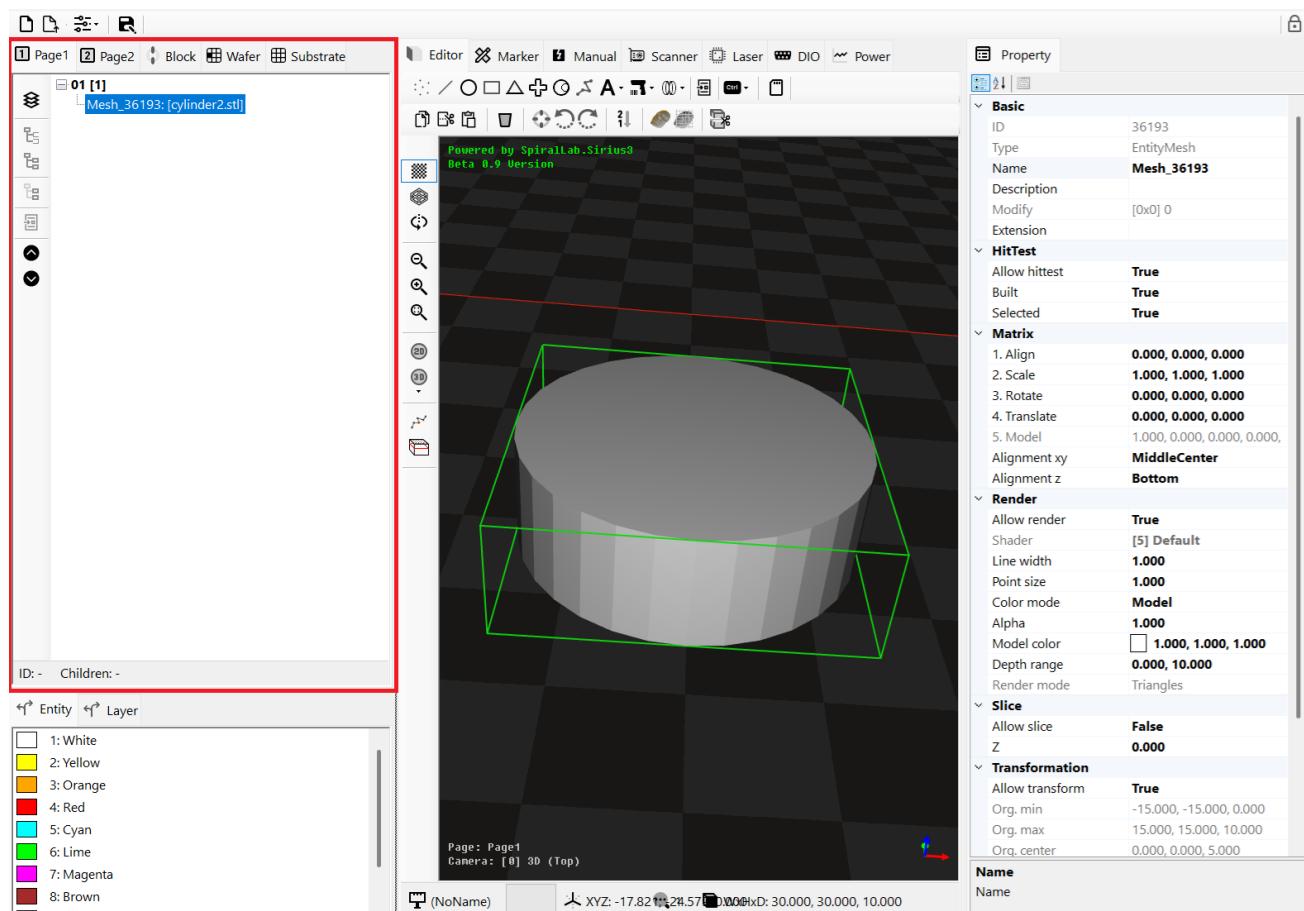
```

3. 문서(Document) 와 기능들

저장 및 불러오기를 이용해 처리되는 문서(IDocument)는 내부에 실제 내부 데이터(IDocumentData) 객체로 관리됩니다. 이 내부 데이터 객체는 아래와 같이 다양한 아이템으로 구성되어 있습니다.

IDocument 내부에 실제 **IDocumentData** 데이터를 구성하는 목록

- **Blocks**: 개체들을 묶어 Block 개체를 만들어 생성되는 목록으로, 추후 **BlockInsert**로 사용됩니다.
- **Pages**: 최대 4 개의 Page 가 지원되며, Page 별로 서로 다른 데이터 편집, 가공 등이 가능합니다.
- **EntityPens**: 마커(IMarker) 가공시 개별 개체(Entity)에 적용되는 개체용 펜 목록
- **LayerPens**: 마커(IMarker) 가공시 개별 레이어(Layer)에 적용되는 레이어용 펜 목록
- **Wafers**: 웨이퍼 맵 항목
- **Substrates**: 기판 맵 항목

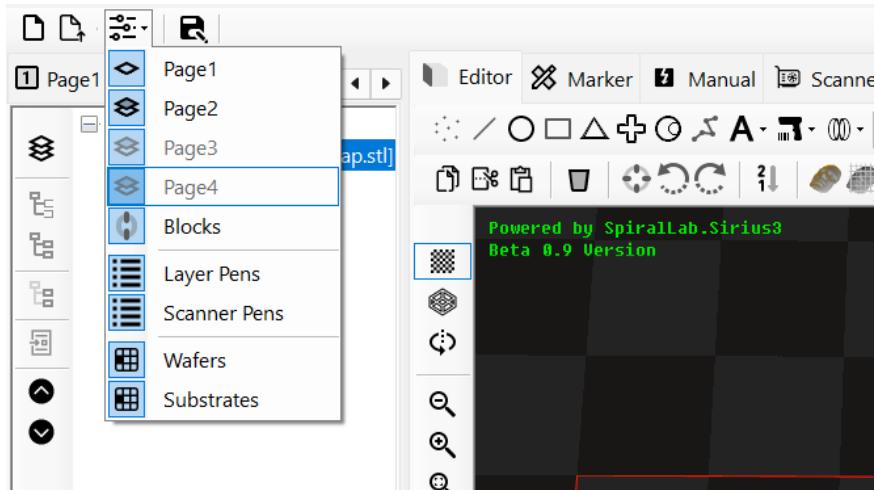


편집기 화면에서 좌측 탭에 **IDocumentData**의 하위 항목들이 트리 컨트롤과 리스트 컨트롤 형태로 출력되며, 탭 변경(혹은 이동)시 활성화 페이지(Page) 변경을 통해 편집기(Editor)의 출력 내용도 같이 업데이트 됩니다.

- **Save** (문서저장): **IDocumentData**의 하위 항목들이 모두 저장됩니다. 이때 파일 크기를 줄이기 위해 기본적으로 압축되어 저장됩니다. 만약 사용자가 저장된 데이터 포맷을 읽어 처리하고자 한다면 **Config**.

`IsCompressedFileFormat`를 `False`로 설정해 압축 기능을 사용하지 않도록 할 수 있습니다.
(참고) `IDocument.ActSave` 함수를 통해 파일 저장이 지원됩니다.

- **New (새 문서) 및 Open (문서 열기)**

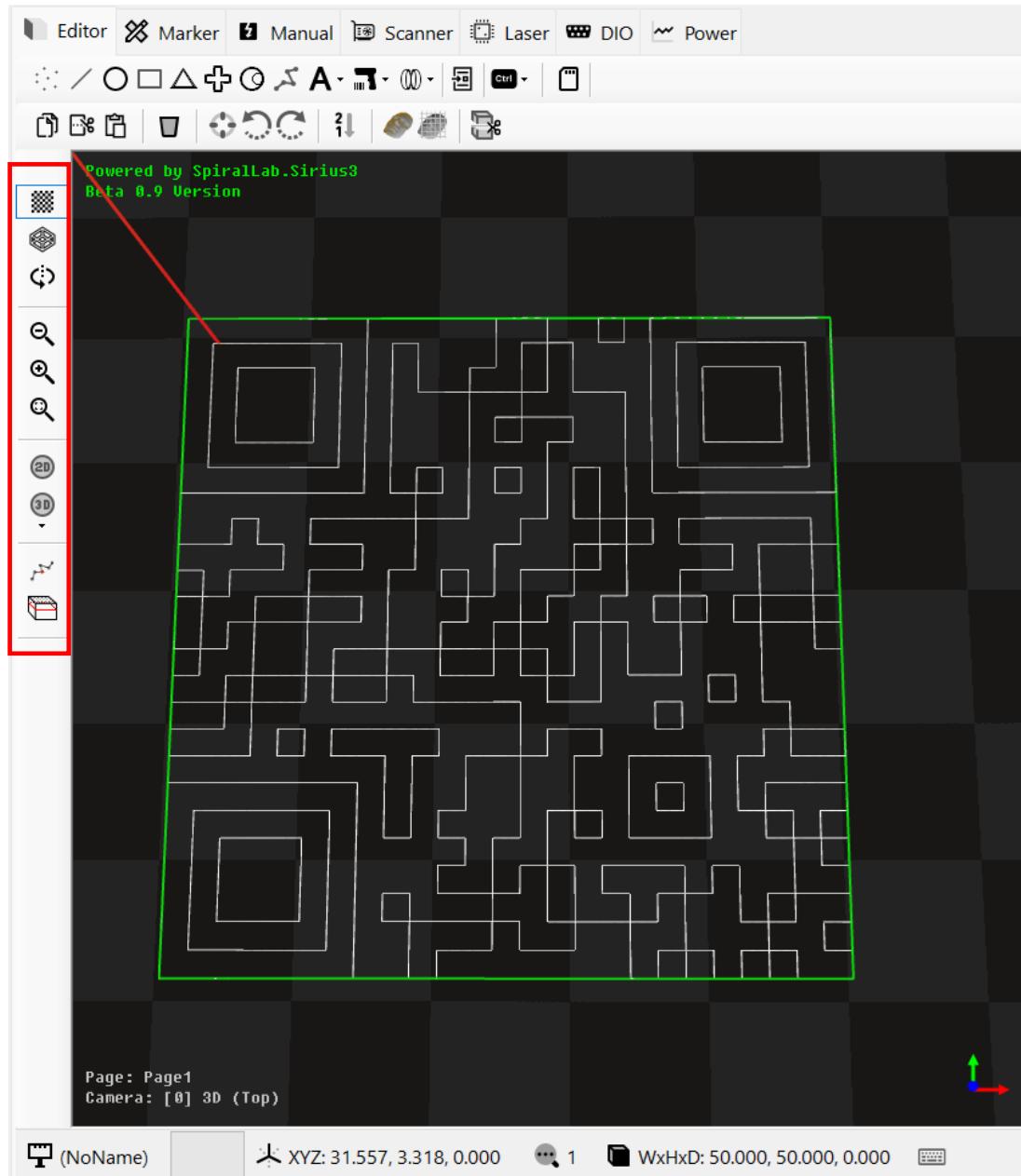


새 문서(New) 혹은 문서 열기(Open)를 하게 되면 기존 데이터는 모두(삭제 혹은 변경) 됩니다. 만약 사용자가 특정 아이템이 변경되는 것을 막고자 하는 특별한 경우에는 위와 같이 아이템 옵션 항목들에 대해 선택해 처리가 가능합니다.

(참고) `IDocument.ActNew` 및 `IDocument.ActOpen` 함수를 통해 새 파일 및 파일 열기가 지원됩니다.

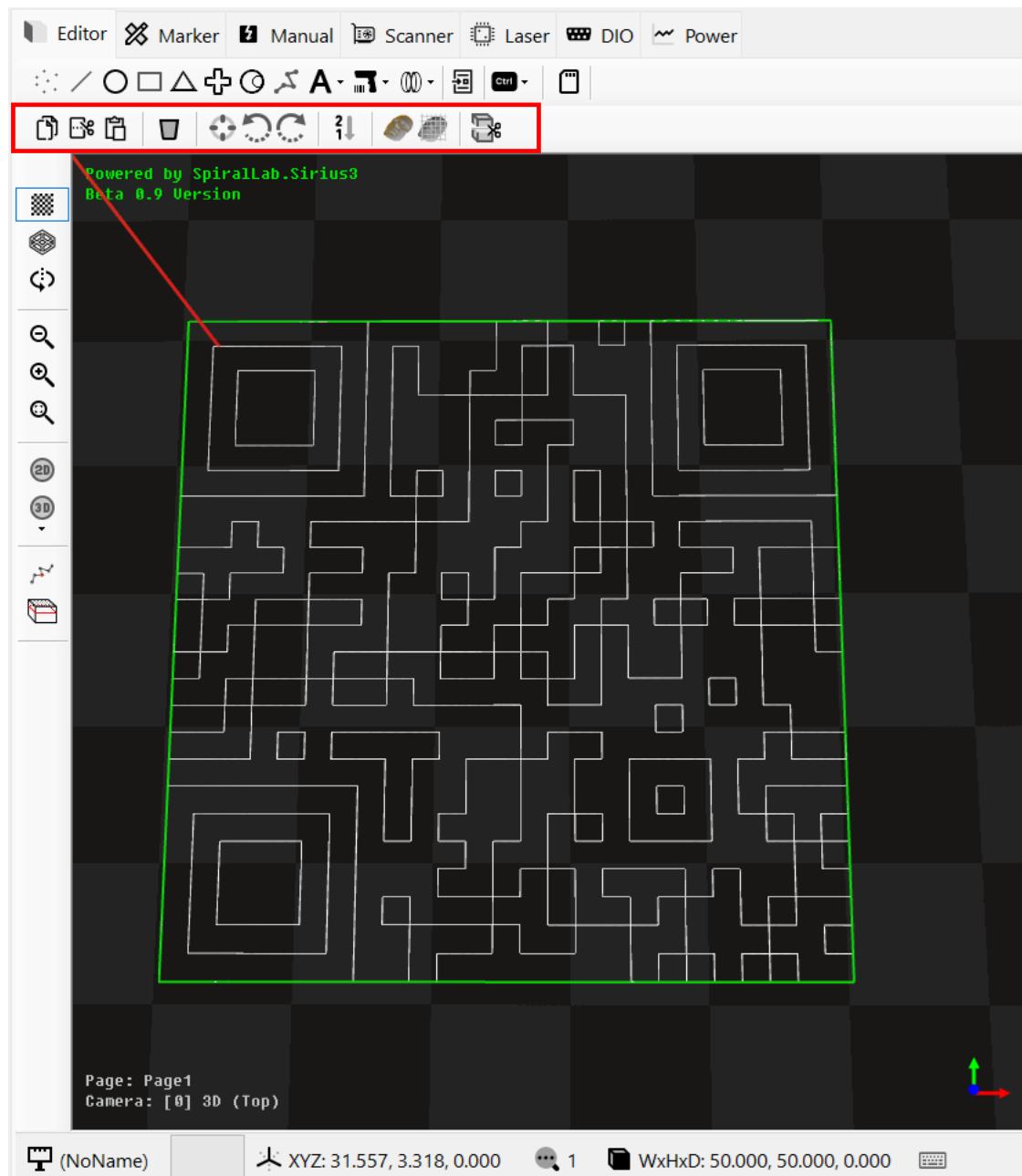
4. 편집기(Editor)

편집기에서는 개체의 생성, 선택, 삭제, 편집 등 모든 데이터 조작이 가능합니다. 또한 확대, 축소, 뷰 카메라 시점 변경, 레이저 가공 시뮬레이션 등의 부가기능도 지원됩니다.



- **CheckerBoard/Grid:** $Z = 0$ 평면에 대해 격자 혹은 체커보드로 배경을 렌더링 합니다.
- **Polygon Mode:** 폴리곤을 점, 선, 면으로 렌더링 처리합니다.
- **Rotating:** 카메라를 자동으로 회전시키며 렌더링 합니다.
- **Zoom Out, Zoom In, Zoom Fit:** 카메라 축소, 확대, 맞추기를 처리합니다.
- **Camera 2D(Ortho):** 편집 화면을 2D 카메라, 즉 직교 좌표계로 설정합니다.
- **Camera 3D(Perspective):** 편집 화면을 3D 카메라로 설정합니다. 또한 상단(Top), 앞면(Front), Right(우측면), Rear(후면), Left(좌측면) 순서로 변경합니다.

- **Simulation:** 선택된 개체에 대해 레이저 가공 시뮬레이션을 시작합니다.
단축키: F1(고속으로 시작), CTRL+ F1(중속으로 시작), CTRL+ALT+F1(저속으로 시작), ESC(중지)
- **Preview:** 선택된 개체에 대해 지시빔(Guide Beam)을 이용해 외곽 영역에 대해 미리 보기 시작합니다.
단축키: F4 (지시빔 시작), F6(마커 중지)
(참고) 구현된 레이저 소스(ILaser) 객체가 ILaserGuideControl 인터페이스를 상속 구현되어 있어야 합니다.
(참고) 내부적으로 IMarker.Preview 함수가 호출되어 실행됩니다.
- 추가적인 단축키(Shortcut)는 편집기 하단에 키보드 아이콘을 눌러 확인해 주시기 바랍니다.



- **Copy:** 선택된 개체(들)을 내부 클립보드로 복사합니다.
(참고) IDocument.ActCopy 함수가 사용됩니다.

- **Cut:** 선택된 개체(들)을 내부 클립보드로 잘라내기 합니다. 원본 개체(들)은 삭제됩니다.
(참고) `IDocument.ActCut` 함수가 사용됩니다.
- **Paste:** 클립보드에 있는 개체(들)을 현재 마우스 위치로 붙혀넣기 합니다.
(참고) `IDocument.ActPaste` 함수가 사용됩니다.

5. 기본 개체들(Primitive Entities)

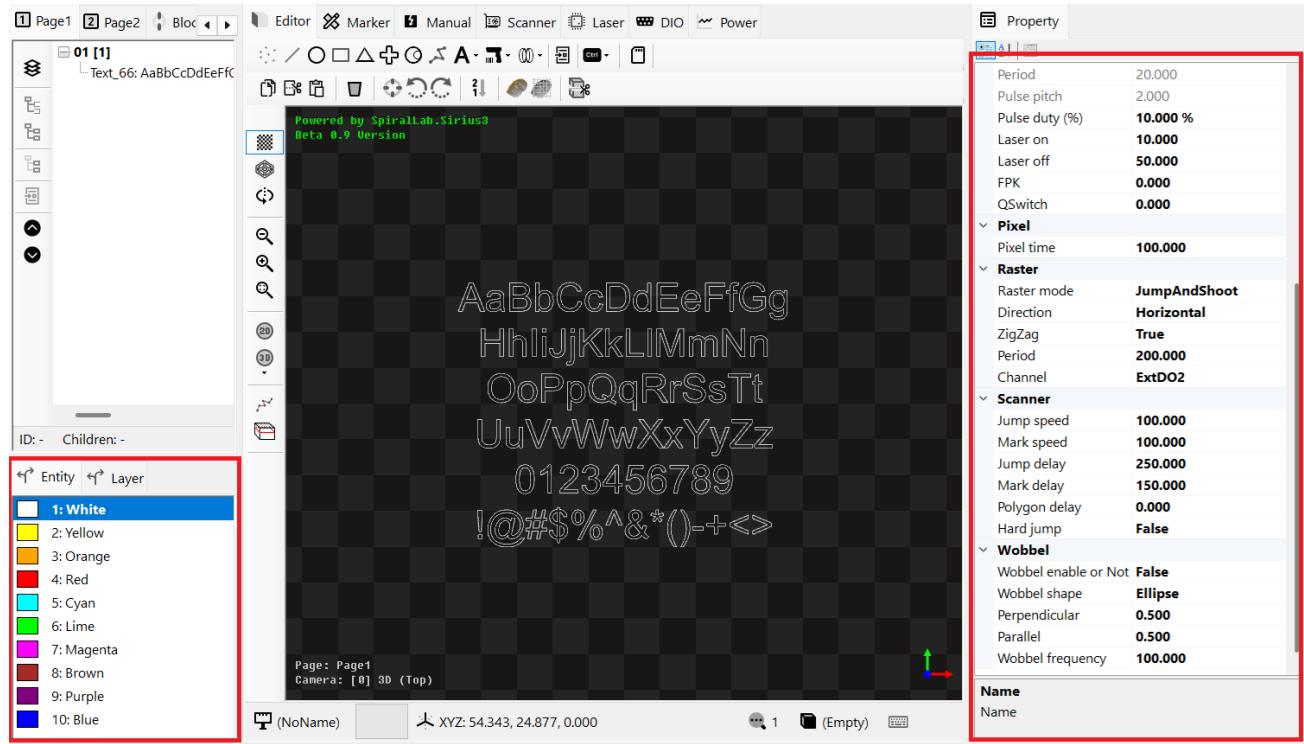
EntityFactory.Create... 함수들을 참고해 주시기 바랍니다.

6. 제어 개체들(Control Entities)

EntityFactory.Create... 함수들을 참고해 주시기 바랍니다.

7. 펜 속성 (Pen Properties)

시리우스 3에서 펜(Pen)은 레이어(Layer)용과 객체(Entity)용 이렇게 2 가지로 제공됩니다. 즉 레이어(Layer) 펜은 레이어 내부에 있는 모든 개체들에 대한 전역적인 설정 값들로 가공중에 적용되며, 개체(Entity) 펜은 개별 개체별로 지정된 펜 색상에 따라 그 설정 값이 가공중에 변경되어 적용됩니다.



이 펜 설정은 좌측 하단에 Entity (개체용 펜), Layer (레이어용 펜) 탭에서 사용하고 있는 펜 색상을 선택하여 속성값을 수정할 수 있습니다.

때문에 페이지(Page) 목록에서 특정 레이어(Layer)를 선택한 후 레이어(Layer)의 펜 색상(Pen color)을 변경하면 해당 레이어에 포함된 모든 개체들에 대해 전역적인 레이어 펜 설정 값들이 마커의 가공시에 반영됩니다.

(참고) 마커 가공시 레이어(Layer) 펜 속성이 적용되는 루틴을 사용자가 직접 구현(Override)하는 방법

```
marker.OnMarkLayerPen += OnMarkLayerPen;

bool Marker_OnMarkLayerPen(IMarker marker, EntityLayerPen pen)
{
    // 자세한 구현 사항은 editor_pen 예제 프로젝트 참고
}
```

(참고) 마커 가공시 개체(Entity) 펜 속성이 적용되는 루틴을 사용자가 직접 구현(Override)하는 방법

```
marker.OnMarkEntityPen += OnMarkEntityPen;

bool OnMarkEntityPen(IMarker marker, EntityPen pen)
{
    // 자세한 구현 사항은 editor_pen 예제 프로젝트 참고
}
```

(참고) 펜 개체에는 사용자 데이터가 저장가능한 **ExtensionData** 속성이 제공되며, 이를 이용해 사용자 데이터를 전달해 다양한 확장 처리가 가능합니다.

7.1 레이어 펜(Layer Pen) 속성 항목

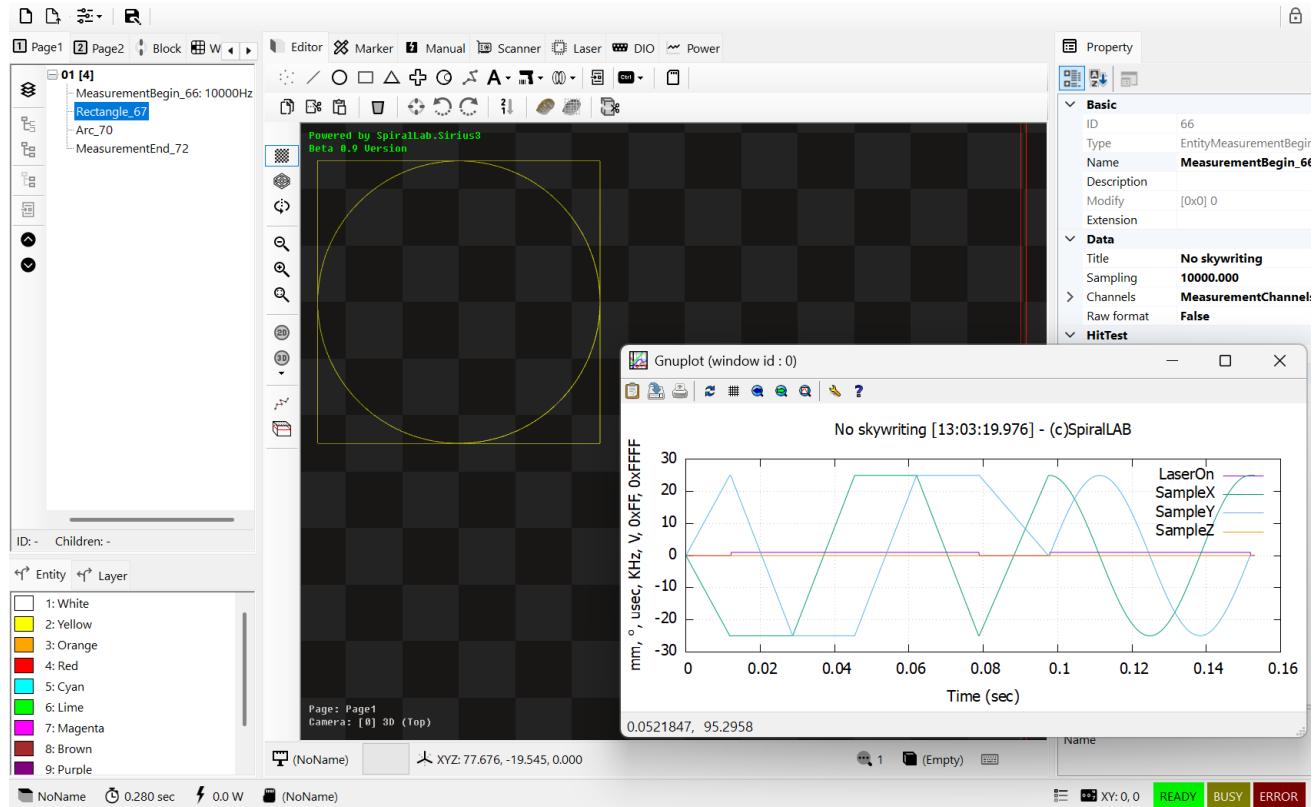
EntityLaserPen 속성값들을 참고해 주시기 바랍니다.

7.2 개체 펜(Entity Pen) 속성 항목

EntityPen 속성값들을 참고해 주시기 바랍니다.

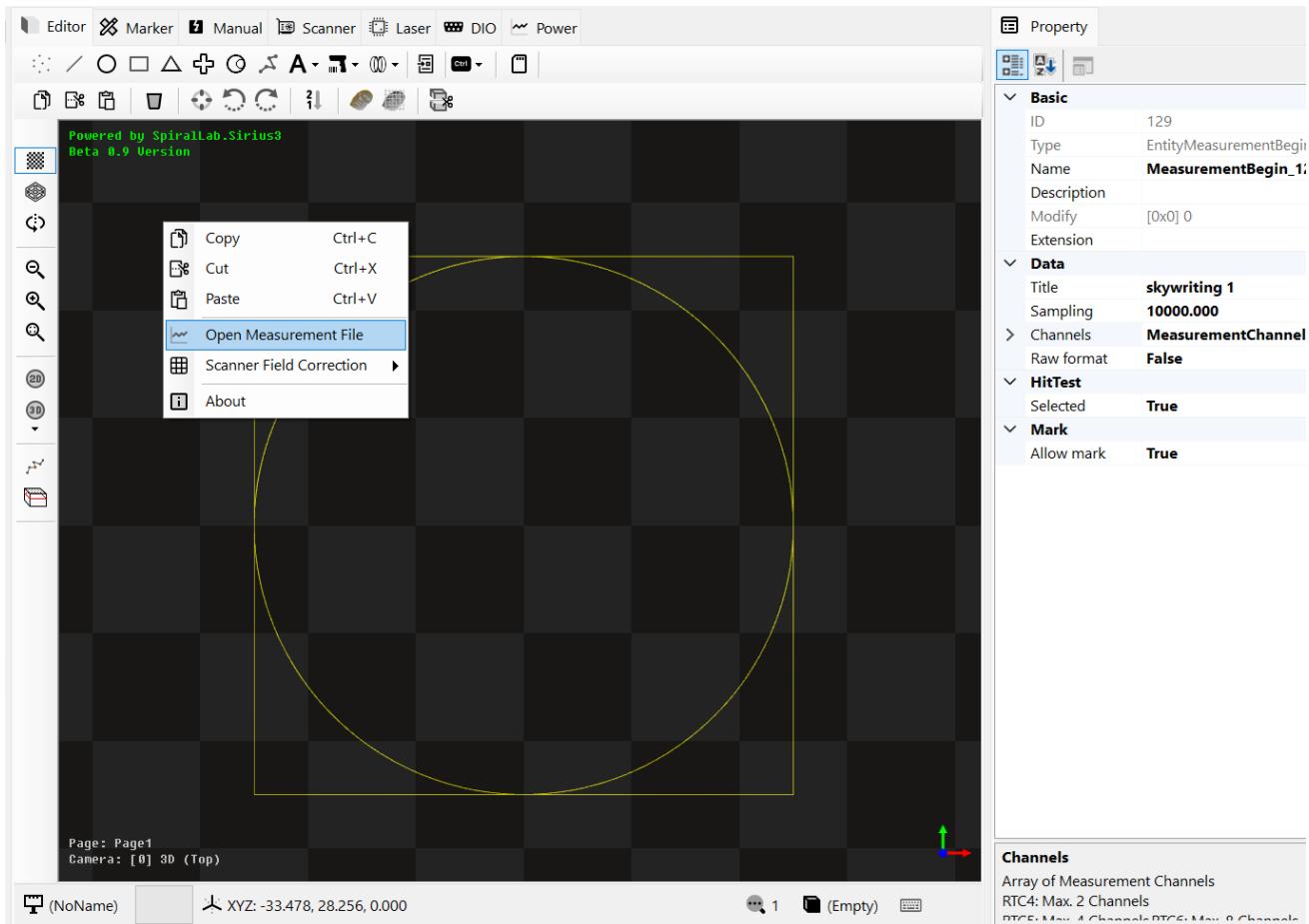
8. 계측 (Measurement)

계측 기능은 계측 시작 및 끝(Measurement Begin ~ End) 사이에 포함된 개체들에 대해서, 지정된 샘플링 주기(Sampling Frequency: 최대 100kHz)와 채널 정보를 사용하여 마커 가공이 완료 되면, 채널 데이터를 모아(Gathering) 이를 그래프로 출력(Plot)하는 기능을 제공합니다.



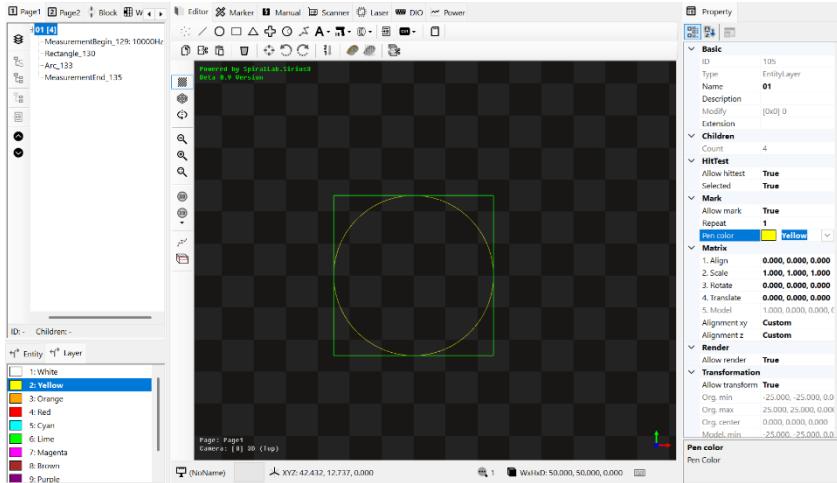
위 예제에서는 사각형과 원을 가공하는데 이때 4개의 채널, 즉 Sample X,Y,Z 및 Laser On 정보를 대상으로 데이터를 시각화 한 모습입니다. 가로축은 시간, 세로축은 데이터값으로 Sample X,Y 채널의 의미는 RTC 제어기에서 내려진 지령된(Commanded) 위치값(mm)을 의미하며, 기울기는 속도(Velocity)에 해당합니다. 또한 Laser On은 0 혹은 1 값을 가지는데, 1의 경우 LASER ON이 진행된 구간을 의미합니다.

Marker 에서는 Plot 값이 True 가 기본값으로 설정되어 있으며, 이 의미는 Measurement 세션이 있는 경우 취득 데이터를 파일에 저장하고 자동으로 그래프 출력(gnuplot 프로그램 사용됨)이 진행됩니다.

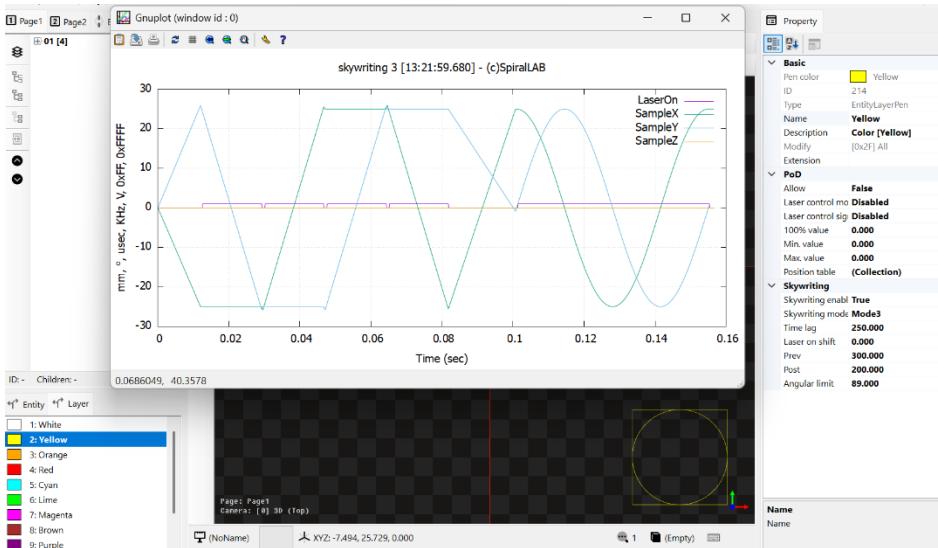


이전에 취득된 계측 데이터를 수동으로 그래프 출력하고자 하면, 편집기에서 마우스 오른쪽 버튼으로 보여지는 메뉴에서 **Open Measurement File**을 이용해 확인이 가능합니다.

8.1 레이어 펜(Layer Pen) 속성의 Skywriting 기능을 사용한 결과



레이어(Layer)를 선택하여 펜 색상(Pen color)을 노란색(Yellow)으로 변경합니다.

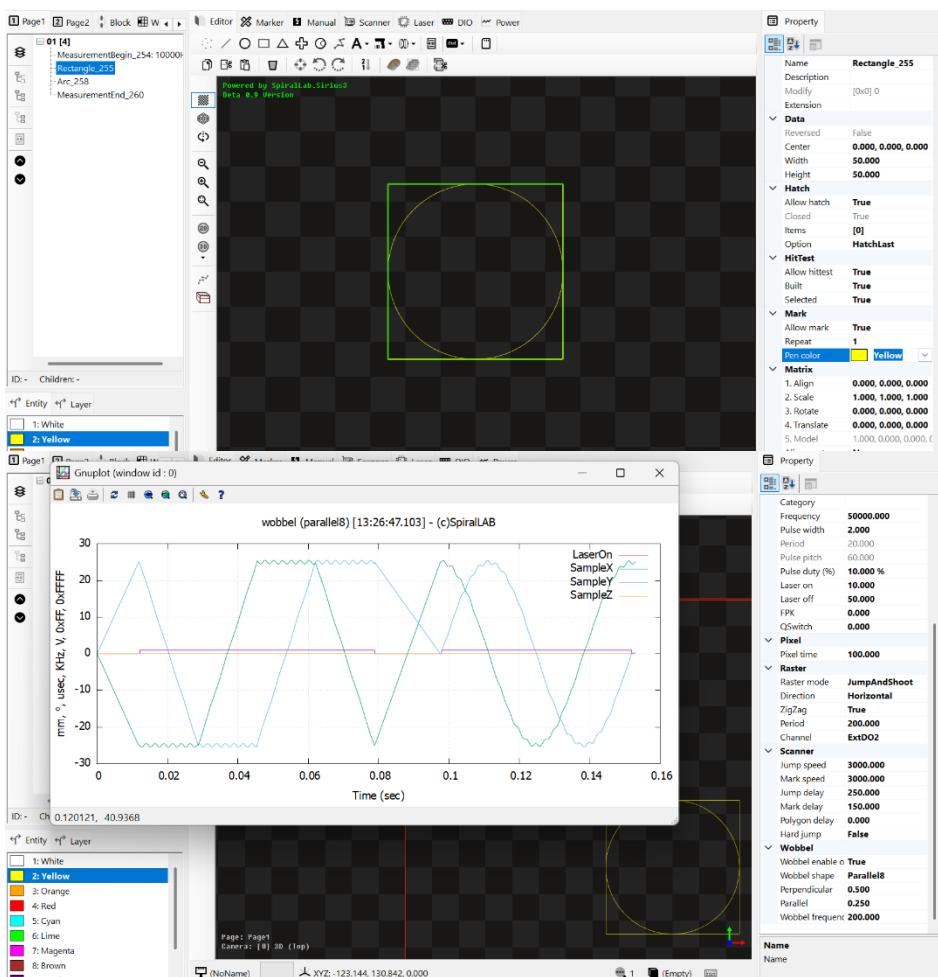


발생한 것을 확인할 수 있습니다.

레이어 펜 (Layer Pen)
탭에서 노란색 레이어용 펜을 선택하고 그 속성에서 Skywriting 모드 3를 선택 (활성화) 한 상태로 마커의 가공을 시작합니다.

Mode 3을 사용한 결과 활성화 각도 (Angular Limit) 가 적용되었고 결국 사각형의 모서리 (90도) 위치 에서는 전, 후진에 해당하는 스캐너의 이동이 추가되었고 또한 LASER ON/OFF 도

8.2 개체 펜(Entity Pen) 속성의 Wobble 기능을 사용한 결과



개체(Entity)를 선택하여 펜 색상 (Pen color)를 노란색 (Yellow)으로 변경합니다.

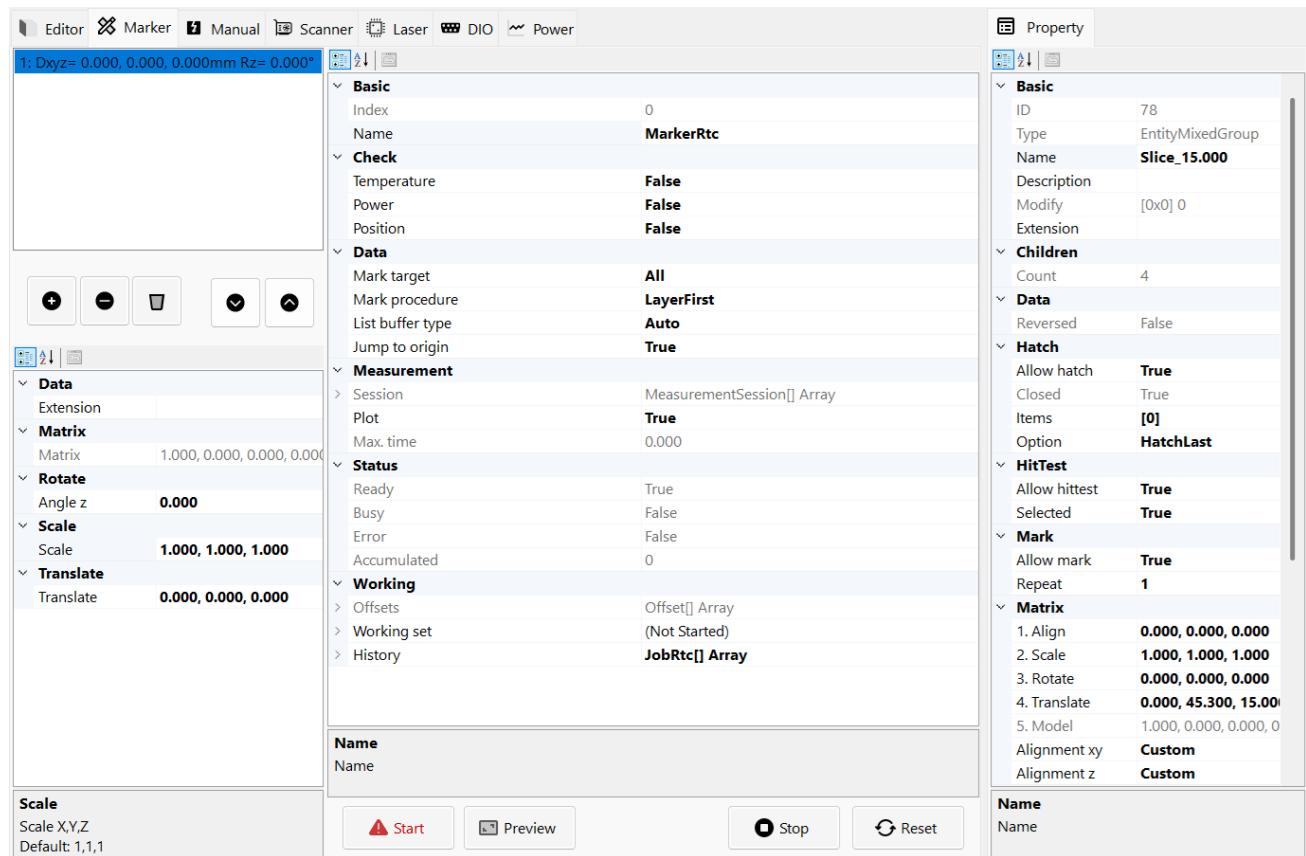
개체 (Entity Pen) 탭에서 노란색 개체용 펜을 선택하고 그 속성에서 Wobble 모드를 활성화 하고 (여기에서는 Parallel8 모양 선택) 진폭 및 주파수를 설정합니다. 그 결과 진행방향에 따라 추가적인 8자 스캐너 모션이 추가된 것을 확인할 수 있습니다.

(참고) RTC 제어기 장치별로

계측 가능한 최대 데이터 개수의 제한이 있으며, 최대 RTC5 4 채널, RTC6는 8 채널까지 사용이 가능합니다.

9. 마커(Marker)

특정 스캐너 제어기(IRtc)에 따라 마커(IMaker) 제어 객체를 생성해 사용하게 되면 아래와 같이 마커 탭에서 해당 객체 상태 및 정보를 확인할 수 있습니다.



마커의 역할은 가공 데이터(IDocument)를 이용해 지정된 페이지(IPage) 데이터에 대해서 레이저 가공을 진행할 때 가공 순서 및 옵션을 처리하는 제어용 객체입니다.

또한 가공 데이터들을 오프셋(Offset) 위치에 대해서 반복적으로 가공이 진행되며, 사용자는 오프셋 위치들을 여러 개 등록해서 동일한 데이터를 다양한 위치에 반복 가공이 가능합니다. (참고) 오프셋은 크기(Scale) → 회전(Angle Z) → 이동(Translate) 순서로 적용됩니다.

9.1 MarkerRtc 객체

- **Check**
 - **Temperature**: 마커 가공 시작(Marker.Start)시에 스캐너의 온도가 적절한지 검사하는 옵션입니다.
 - **Power**: 마커 가공 시작(Marker.Start)시에 스캐너의 전원이 정상적으로 공급 중인지 여부를 검사하는 옵션입니다.
 - **Position**: 마커 가공 시작(Marker.Start)시에 스캐너의 위치 응답(Position ACK)이 정상적인지를 검사하는 옵션입니다.
- **Data**
 - **Mark Target**: 선택된 개체(Selected)만 가공할지 아니면 모두(All) 가공할지 여부를 선택합니다.

- **Mark procedure:** 오프셋(Offset)과 레이어(Layer) 간에 가공 순서를 결정합니다.
 - **Layer First:** 오프셋 1 → 레이어 1, 레이어 2, ... → 오프셋 2 → 레이어 1, 레이어 2, ...
 - **Offset First:** 레이어 1 → 오프셋 1, 오프셋 2, ... → 레이어 2, → 오프셋 1, 오프셋 2, ...
- **List buffer type:** 스캐너 제어장치(IRtc)의 리스트 버퍼 모드를 결정합니다.
 - **Auto:** 더블 버퍼 처리가 자동으로 사용됨. (리스트 버퍼에 데이터가 충분히 저장되면 자동으로 가공 시 시작됨)
 - **Single:** 단일 버퍼 처리가 사용됨. (리스트 버퍼가 허용하는 데이터 개수만 입력되면, 가공 시작 명령(ListExecute)이 있어야 가공이 시작됨)
- **Jump to origin:** 가공 완료 후 스캐너 장치의 원점($0, 0$)으로 자동으로 점프할지 여부를 결정합니다.
- **Measurement**
 - **Plot:** 가공 완료 후 플롯 출력을 할지 여부
 - **RTC 제어기 사용시:** 계측 개체(Measurement Begin/End)를 이용하고 있을 때, **True** 인 경우에는 가공이 완료된 후 계측 데이터를 저장 후 gnuplot 를 이용해 지정된 채널 데이터에 대한 프로파일 출력을 자동으로 실행합니다.
 - **syncAXIS 제어기 사용시:** 시뮬레이션 모드를 사용하고 **True** 인 경우 가공이 완료되면 syncAXISViewer 프로그램을 이용해 스캐너, 스테이지에 대한 모션 프로파일을 출력 및 실행을 자동으로 실행합니다.
- **Status**
 - **Ready:** 마커가 가공이 가능한 준비 상태인지 여부
 - **Busy:** 마커가 가공중인 상태인지 여부
 - **Error:** 마커가 에러 상태인지 여부
- **Working**
 - **Offsets:** 가공중에 있는 현재 오프셋 정보입니다.
 - **Working set:** 가공중에 있는 현재 작업 집합 정보입니다. 주요 정보는 다음과 같습니다.
 - 현재 페이지(IPage) 및 페이지 번호
 - 현재 레이어(EntityLayer) 및 레이어 번호
 - 현재 개체(IEntity) 및 개체 번호
 - **History:** 누적된 가공 이력 정보들입니다.

Start 버튼: 현재 작업중인(화면에서 활성화된) 페이지에 대해 마커 가공 시작을 시도합니다. (주의) 마커의 상태가 **Ready** 가 **True**이고 **Error** 가 **False**인지 여부를 확인해야 합니다.

Preview 버튼: 현재 선택된 개체들이 있는 상태에서 이 버튼을 누르면 개체들을 감싸는 외곽 사각영역에 대해 레이저 소스의 지시빔(Guide Beam)을 활성화 한 후 반복적으로 해당 영역에 대해 지시빔 출력을 진행합니다.

(주의) 레이저 소스 객체가 지시빔 기능(ILaserGuideControl 인터페이스)을 상속 구현하고 있어야 합니다.

Stop 버튼: 가공을 강제로 중단(Abort) 합니다.

Reset 버튼: 마커의 에러 상태를 초기화 하기 위해 리셋을 시도합니다.

(참고) 마커에 대한 키보드 단축키(Marker Keyboard Shortcut) 정보

F4: 마커 가공 프리뷰(Preview) 시작

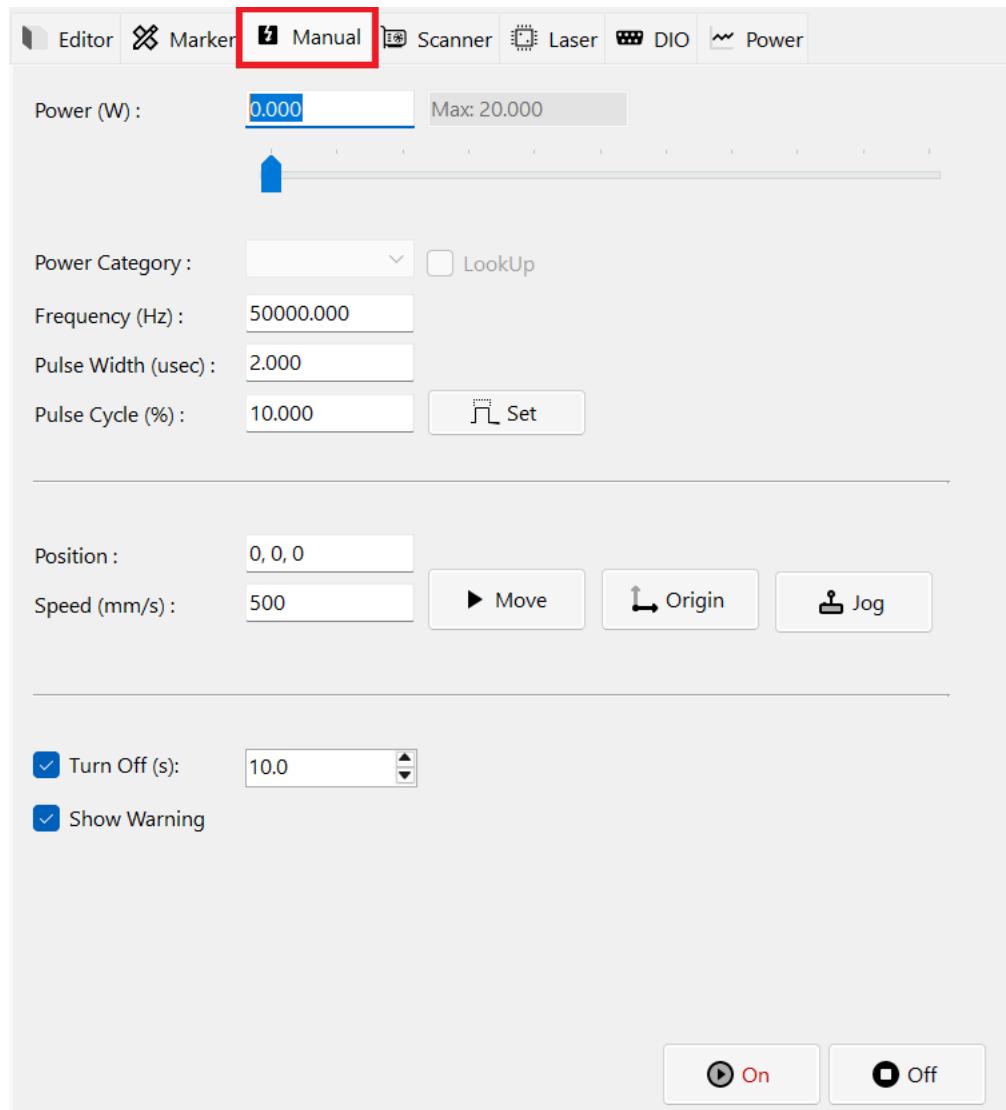
F5: 마커 가공 시작(Start)

F6: 마커 가공 중지(Stop)

F8: 마커 에러 리셋

10. 수동(Manual)

사용자는 레이저 출력 파워(W)를 변경하고 일정시간동안 레이저를 출사하고 스캐너의 위치를 이동시키는 수동 작업을 수동 화면 탭에서 수행할 수 있습니다.



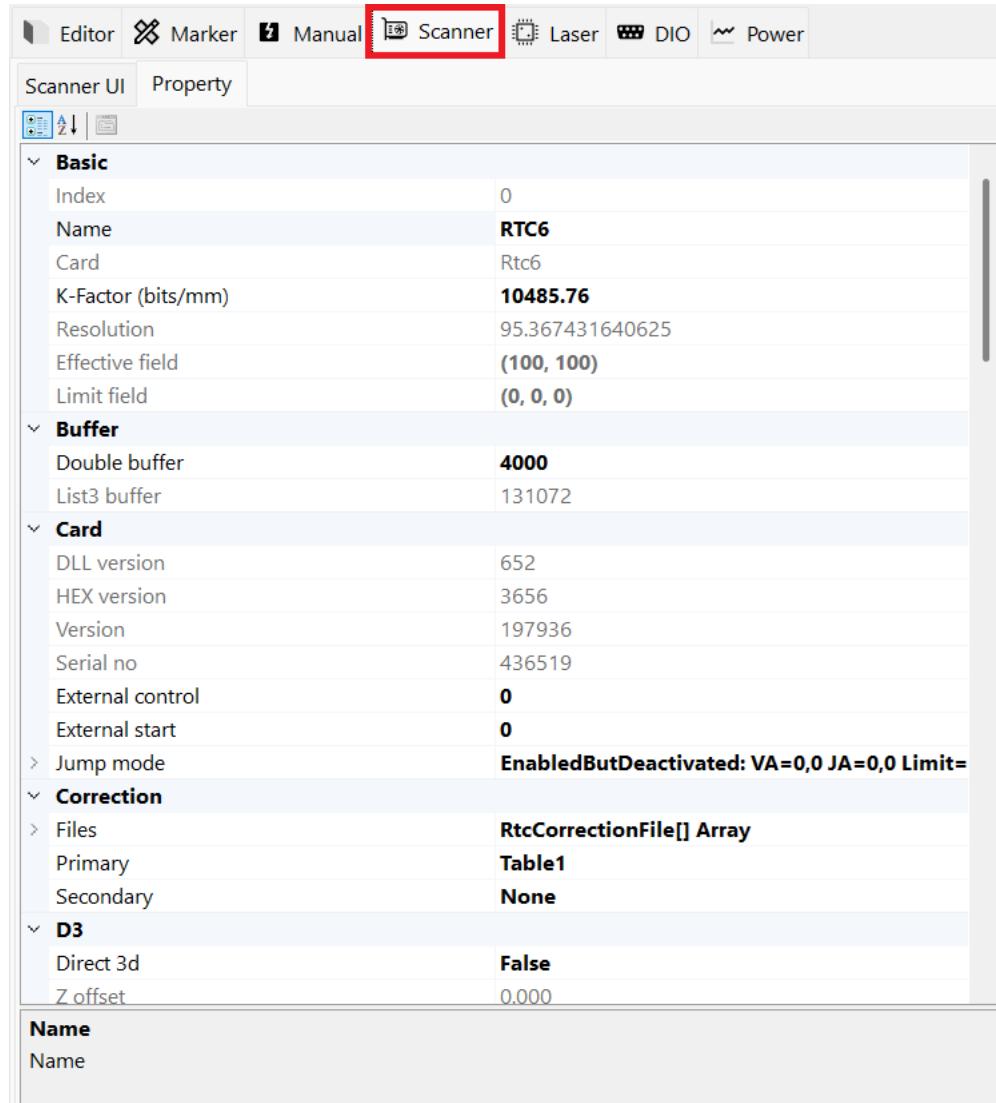
Manual 탭에서는 레이저 소스의 출력값을 수동으로 변경이 가능하고, 스캐너의 반사 X,Y 위치 이동 등 다양한 수동 작업이 가능합니다. 여기에서는 레이저 빔의 정렬(Alignment) 혹은 수동 파워 측정등의 용도로 사용 가능합니다.

(참고 1) RTC 제어기의 LASER 1,2 및 LASERON 신호선에서 발생하는 펄스 신호가 레이저 소스 입력 처리되어 발진됩니다. 이때 신호 레벨(Active Low, High)을 확인해야 하고, Yag 1,2와 같이 어떤 레이저 모드(Laser Mode)를 사용하고 있는지 확인해야 합니다. 또한 레이저 소스 제품의 사용자 매뉴얼을 참고해 RTC 제어기와 레이저 소스간에 정상적인 결선이 되어 있어야 합니다.

(참고 2) 레이저 발진(On)을 시도할 경우에는 반드시 레이저 빔에 의한 위험 요소를 모두 확인하고 진행해야 합니다.

11. 스캐너(Scanner)

스캐너(IScanner) 제어 장치에 대한 상태 및 설정을 변경할 수 있습니다. 예를 들어 RTC6 제어를 위해 Scanner 객체를 생성해 사용한다면 아래와 같이 RTC 내부의 다양한 상태를 확인할 수 있고, 나아가 세부적인 설정을 변경할 수 있습니다.



ScannerFactory에 의해 생성된 IRtc 객체 정보가 출력됩니다. 만약 레이저 소스 UI를 삽입하는 것처럼 Scanner UI에 외부에서 만든 사용자 컨트롤을 Config. OnCreateScannerUI 이벤트를 사용해 추가할 수 있습니다.

```

SpiralLab.Sirius3.UI.Config.OnCreateScannerUI+= OnCreateScannerUI;

private Control OnCreateScannerUI (IScanner scanner)
{
    return new YourScannerUIControl()
    {
        Scanner = scanner
    };
}

```


12. 레이저(Laser) 소스

현실적으로 매우 다양한 레이저 소스 디바이스가 존재하며, 그 제어 방식 또한 사용자의 요구사항에 맞도록 사용하기 때문에, 모든 환경에 맞는 레이저 소스 구현부를 라이브러리 내부에서 모두 지원하기는 불가능합니다. 때문에 사용자는 손쉽게 레이저 소스 객체를 직접 구현하고 이를 라이브러리 내에 포함시켜 확장 사용이 가능합니다.

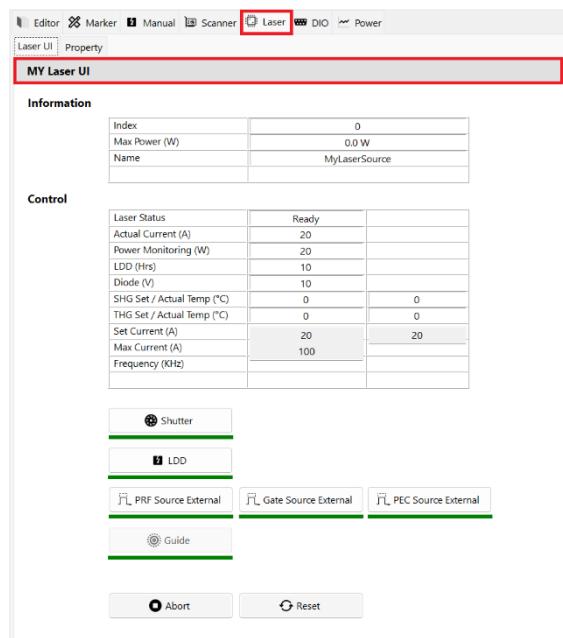
12.1 레이저 소스 제어부 구현

레이저 소스를 구현하기 위해서는 라이브러리에서 제공하는 **ILaser** 인터페이스를 상속 구현해야 합니다. (레이저 파워 제어를 위한 **ILaserPowerControl** 와 지시빔 제어를 위한 **ILaserGuideControl** 인터페이스도 제공됩니다.)

가장 손쉬운 방법은 다음과 같이 추상 객체인 **LaserBase** 를 상속 구현하는 것을 추천드립니다.

```
public class MyLaserSource
    : LaserBase
    , ILaserPowerControl
    , ILaserGuideControl
{
    ...
    // 레이저 소스 제어 로직을 구현
}
```

12.2 레이저 소스 UI 구현



앞선 레이저 소스 제어부 맞는 UI는 사용자 컨트롤 (**UserControl**)로 만들어 **SiriusEditorControl**의 **Laser** 탭 내부 컨트롤을 손쉽게 교체할 수 있습니다. 이를 위해서는 **YourLaserUIControl**에 해당하는 사용자 컨트롤 UI 객체를 직접 디자인 및 구현한 후 다음과 같은 이벤트 핸들러를 이용해 레이저 소스 UI를 변경(오버라이드) 할 수 있습니다.

```
SpiralLab.Sirius3.UI.Config.OnCreateLaserUI += OnCreateLaserUI;

private Control OnCreateLaserUI(ILaser laser)
{
    return new YourLaserUIControl()
    {
```

```
    LaserSource = laser // MyLaserSource 의 인스턴스 전달
};
```

(참고) `editor_laser_ui` 예제 프로젝트를 통해 실제 구현 및 사용된 예제가 제공됩니다.

13. 디지털 입출력 (DIO)

RTC 제어기에는 다양한 디지털 입출력 포트가 제공됩니다. 각각의 포트를 다음과 같이 생성 및 초기화 하고 이를 편집기에 설정합니다.

```
editor.DIExt1 = IOFactory.CreateInputExtension1(rtc);
editor.DIExt1.Initialize();
editor.DOExt1 = IOFactory.CreateOutputExtension1(rtc);
editor.DOExt1.Initialize();
editor.DOExt2 = IOFactory.CreateOutputExtension2(rtc);
editor.DOExt2.Initialize();
editor.DILaserPort = IOFactory.CreateInputLaserPort(rtc);
editor.DILaserPort.Initialize();
editor.DOLaserPort = IOFactory.CreateOutputLaserPort(rtc);
editor.DOLaserPort.Initialize();
```

(참고) syncAXIS 제어기 사용시 확장 1 포트만 사용 가능합니다.

DIN EXTENSION1 PORT			DOUT EXTENSION1 PORT		
No		Status	No		Status
0	D0	OFF	0	D0	OFF
1	D1	OFF	1	D1	OFF
2	D2	OFF	2	D2	OFF
3	D3	OFF	3	D3	OFF
4	D4	OFF	4	D4	OFF
5	D5	OFF	5	D5	OFF
6	D6	OFF	6	D6	OFF
7	D7	OFF	7	D7	OFF
8	D8	OFF	8	D8	OFF
9	D9	OFF	9	D9	OFF
10	D10	OFF	10	D10	OFF
11	D11	OFF	11	D11	OFF
12	D12	OFF	12	D12	OFF

DIN LASER PORT			DOUT EXTENSION2 PORT		
No		Status	No		Status
0	D0	ON	0	D0	OFF
1	D1	ON	1	D1	OFF
			2	D2	OFF
			3	D3	OFF
			4	D4	OFF
			5	D5	OFF

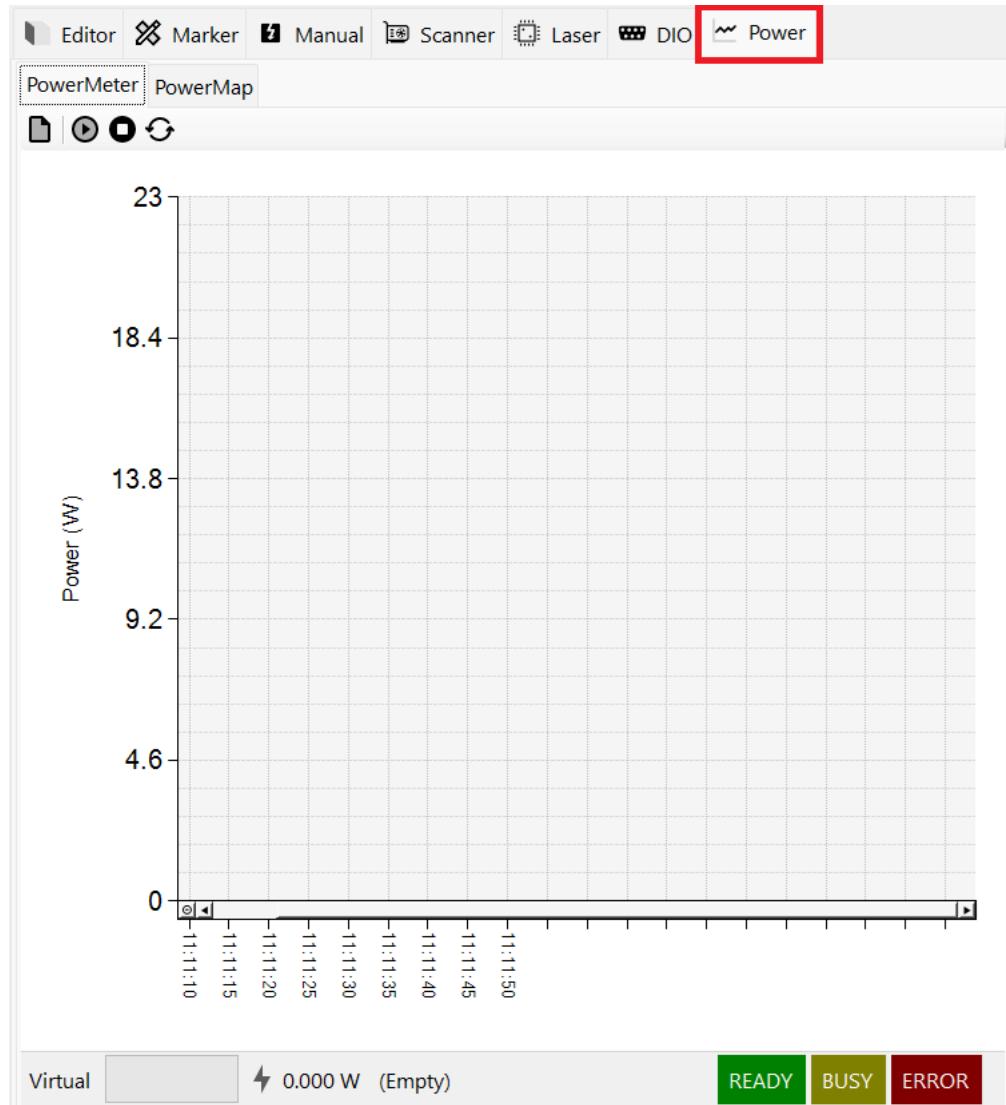
DOUT LASER PORT		
No		Status
0	D0	OFF
1	D1	OFF

D.IO 탭으로 이동하면 위와 같이 시각적으로 상태를 확인할 수 있으며, 우측 출력 포트의 경우 마우스 더블 클릭을 사용해 수동으로 출력을 On/Off 할 수 있습니다. 입력 포트의 특정 비트가 상태가 변경되어 이를 처리하는 등의 루틴이 필요하다면 `editor_dio` 예제 프로젝트를 참고해 주시기 바랍니다.

14. 파워메터(PowerMeter)와 파워맵(PowerMap)

파워메터는 발진하는 레이저 소스의 에너지를 측정하는 장치로 다양한 벤더의 제품들이 제공됩니다. 이를 위해 **IPowerMeter** 인터페이스가 제공되며 이를 상속 구현해 파워메터 장치를 구현합니다. 다음과 같이 생성 및 초기화하고 편집기에 **IPowerMeter** 속성에 지정해 사용합니다.

```
var powerMeter = PowerMeterFactory.Create... ;
powerMeter.Initialize();
```



파워메터 장치 측정을 시작하고, 종료 및 측정된 데이터를 주기적으로 그래프로 출력하는 것은 **Power** 탭에서 사용자가 버튼을 눌러 처리할 수 있습니다.

통상 파워메터를 이용하는 이유는 지령된 값(설정 파워)과 실제 취득된 값(측정 파워)의 편차가 발생하기 때문에 이를 보상(Compensate)하기 위해 파워맵(IPowerMap)을 사용합니다. 라이브러리에서는 다음과 같이 파워맵 객체를 생성하고 카테고리를 생성해 사용이 가능합니다. 또한 레이저 소스는 **ILaserPowerControl** 인터페이스를 상속구현하게 되면 **PowerMap** 속성이 존재하며, 이를 사용해 레이저 소스와 파워맵을 서로 연결시켜 사용할 수 있습니다.

```

var powerMap = PowerMapFactory.CreateDefault(0, "default");
powerMap.Reset1to1("10000", laserMaxPower);

// or 기존 파일이 있다면 PowerMapSerializer.Open 을 이용해 맵핑 데이터를 로드 합니다.

laser.PowerMap = powerMap;

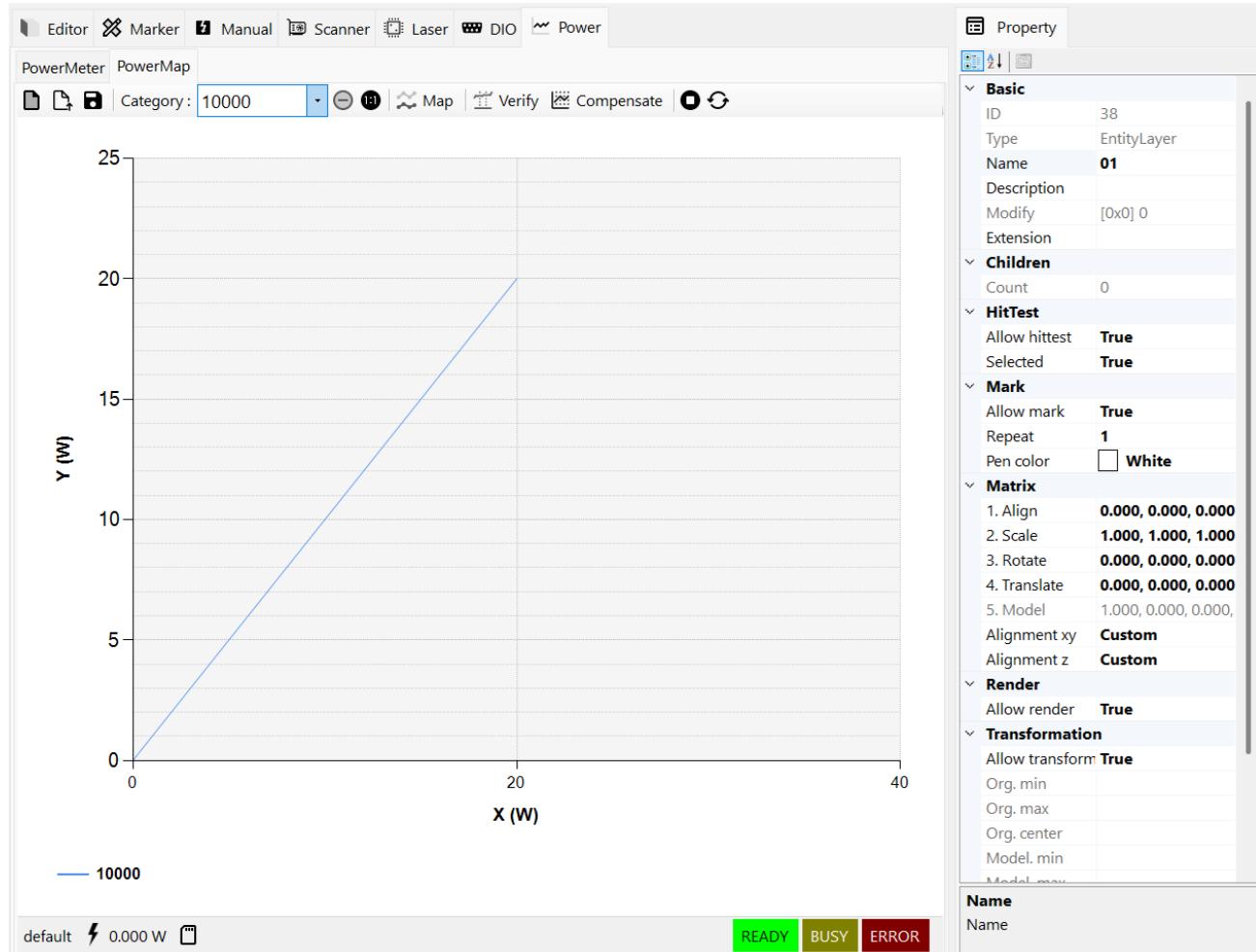
```

파워맵의 주요 기능은 다음과 같습니다.

Mapping: 여러 카테고리와 X 파워 집합에 대해 파워 맵핑을 시작합니다.

Verify : 지정된 카테고리별 목표 Y 파워를 사용해 검증을 수행합니다

Compensate: 카테고리/목표 Y(W)에 맞추기 위해 보상(재보정)을 수행합니다. 허용 범위를 벗어나면 재시도/테이블 갱신을 진행합니다

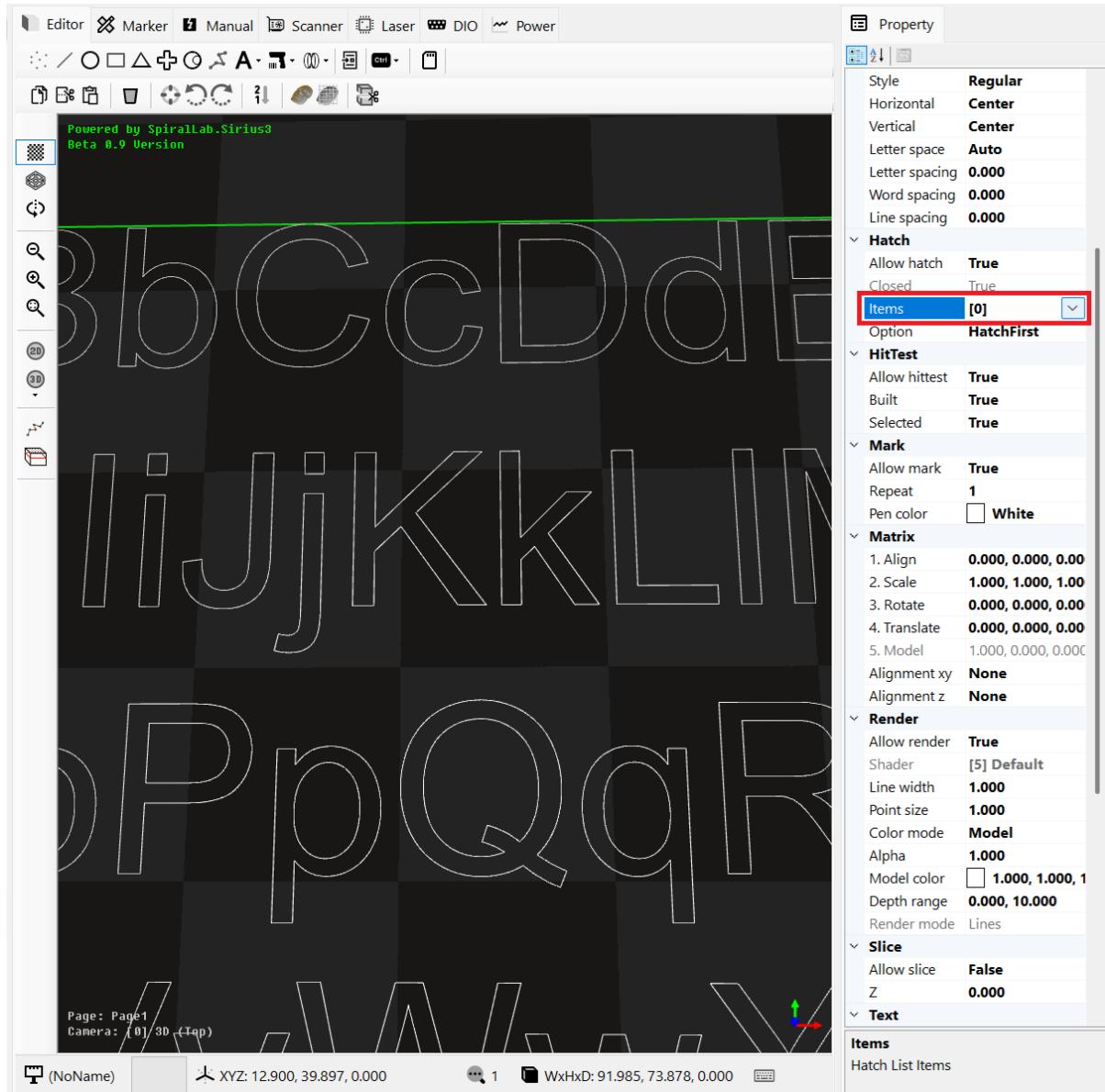


이 같은 기능은 PowerMap 탭에서 사용자가 버튼을 눌러 처리할 수 있습니다. 또한 PowerMapSerializer.Open, PowerMapSerializer.Save 함수로 파일 저장 및 파일 열기가 가능합니다. 파워맵의 자세한 구현 사항은 editor_powermap 예제 프로젝트에 공개되어 있습니다. 이 코드를 수정해 사용자가 원하는 다양한 방식으로 구현

및 확장이 가능합니다.

15. 해치(Hatch)

폐곡선¹ 영역을 가지는 개체들에 대해서는 내부 영역을 채우는 해치를 추가할 수 있습니다. 예를 들어 텍스트(Text) 개체에 대해 해치를 추가하기 위해서는 아래와 같이 Hatch 항목의 Items 버튼을 눌러 해치를 추가/삭제/수정이 가능합니다.

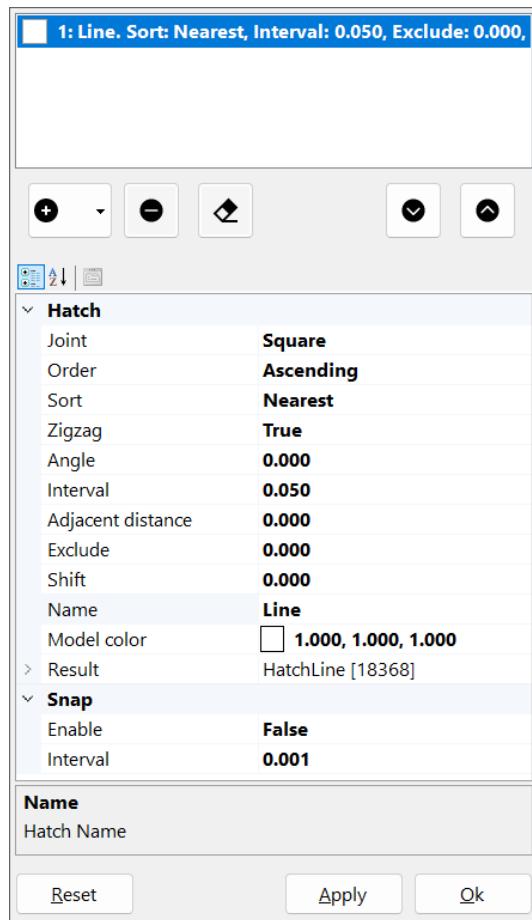


15.1 선분 해치

만약 선분(Line) 해치를 추가할 경우 '+' 버튼을 눌러 Line 해치를 추가하고 적용(Apply)을 누르면 미리보기가 가능해집니다. 또한 폴리곤(Polygon) 해치 모양도 지원되며, 여러 개의 선분, 폴리곤 해치를 무제한으로 추가, 삭제

¹ 원, 폴리 라인(IsClosed 값이 True), Text 과 같이 닫힌 경로를 가진 객체들

제가 가능합니다. 또한 위, 아래 이동 버튼을 이용해 해칭들 간의 가공 순서를 이동시킬 수 있습니다.



선분(Line) 해치 옵션 설명

- **Hatch Joint:** 볼록한 각도의 접합부에서 오프셋을 관리하는 방식을 지정합니다. 오목한 접합부는 항상 **Miter** 접합으로 오프셋 됩니다.
 - **Square:** 사각형 조인트 유형. 볼록한 접합부는 '스퀘어링' 모서리로 잘립니다. 그리고 이 스퀘어링 모서리의 중간점은 원래(또는 시작) 정점에서 정확히 오프셋 거리만큼 떨어져 있습니다.
 - **Round:** 둥근 조인트 유형. 모든 볼록한 조인에 반경이 오프셋 거리인 원호가 적용되며, 원래 조인 정점이 원호의 중심이 됩니다.
 - **Miter:** 마이터 조인트 유형. 가장자리는 먼저 원래(즉 시작) 가장자리 위치에서 지정된 거리만큼 평행하게 오프셋 됩니다. 이 오프셋 된 가장자리는 인접한 가장자리 오프셋과 교차하는 지점까지 연장됩니다.
- Hatch Order:** 선분 생성 순서를 지정합니다.
- **Ascending:** 오름차순 정렬됩니다.
 - **Descending:** 내림차순 정렬됩니다.

Hatch Sort: 선분들 간의 정렬 상태를 지정합니다.

- **None:** 생성된 순서를 그대로 이용합니다. (정렬 없음)
- **Nearest:** 가장 가까운(인접한) 선분을 탐색하여 정렬합니다.
- **Global:** 선분들을 전역적으로 경로 최적화를 통해 정렬합니다.

Hatch ZigZag: 선분 생성시 지그재그 형태로 생성할지 여부를 지정합니다.

Hatch Angle: 선분 생성시 회전 각도를 지정합니다.

Hatch Interval: 선분 생성시 선분간 간격을 지정합니다. (θ 보다 커야 합니다)

Hatch Adjacent Distance: 선분간 최소 인접 거리를 지정합니다. (θ 인 경우 사용되지 않습니다. **Sort** 가 설정되었을 때에만 사용됩니다)

Hatch Exclude: 제외할 영역에 대한 외곽 거리 값입니다. 폐곡선이 해당 값만큼 축소된 후 해치 생성이 이루어집니다.

Hatch Shift: 선분 생성시 해당 거리 값만큼 이동(오프셋) 시켜 생성됩니다.

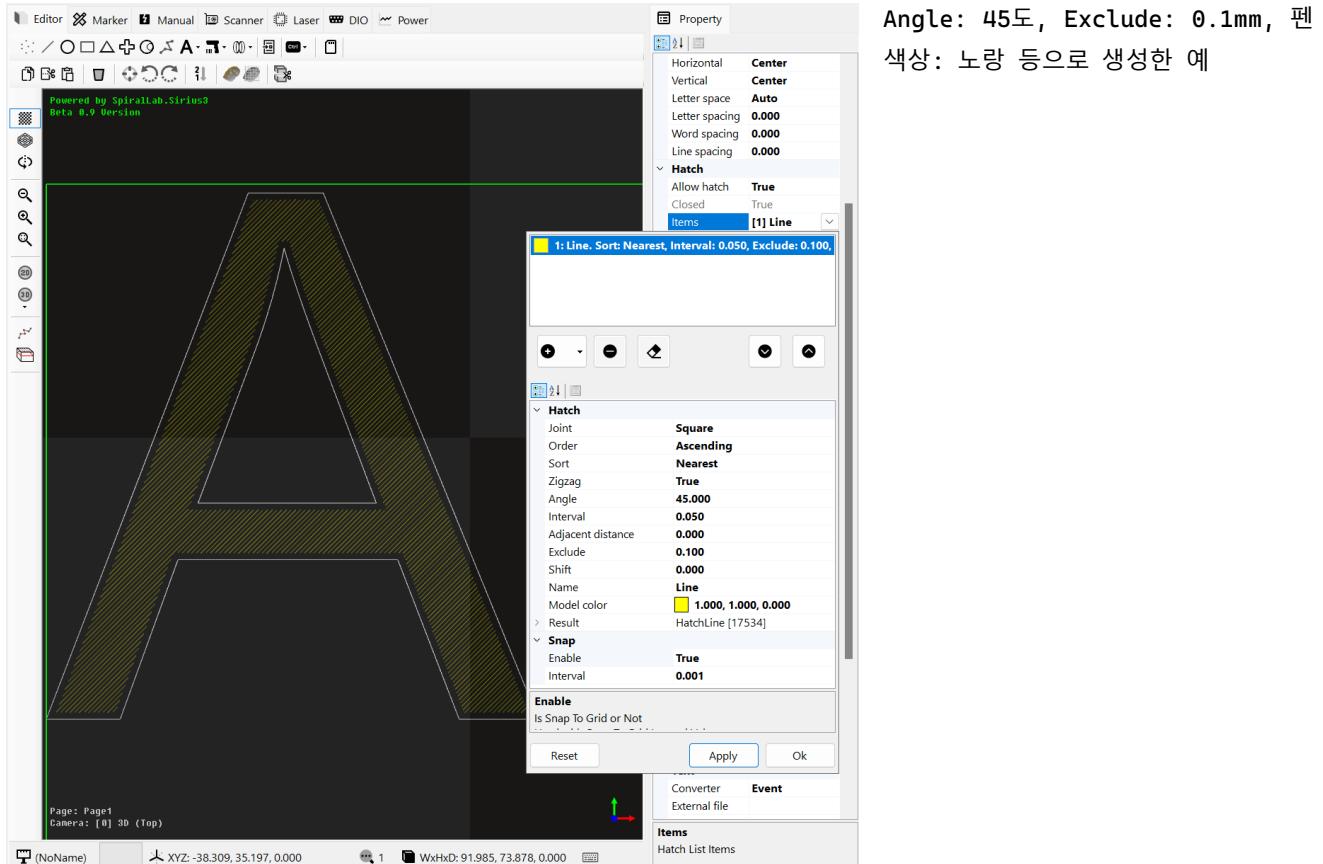
Hatch Model color: 해치에 사용할 펜 색상(Pen color)을 지정합니다. 색상 변경을 통해 레이저 가공 시 서로 다른 스캐너 펜(Scanner pen) 속성들을 적용해 가공할 수 있습니다.

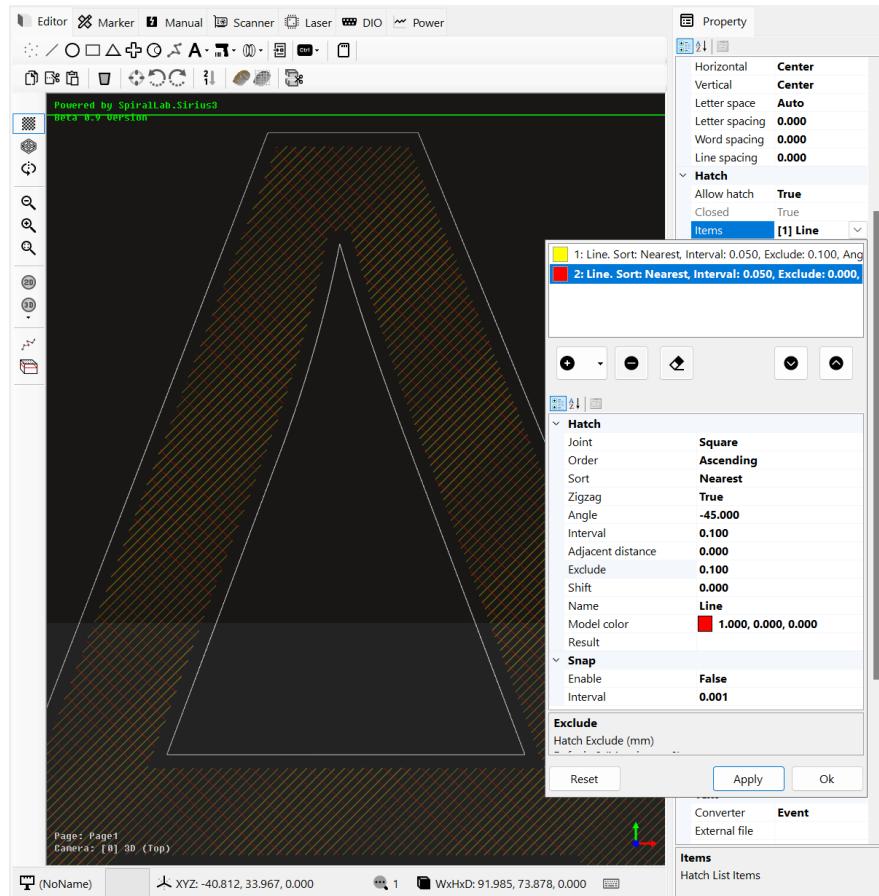
Hatch Result: 내부적으로 생성된 결과 저장용 객체입니다.

Snap Enable: 선분 생성시 Snap Interval 값의 배수 간격 위치에 생성시킬지 여부입니다. (Snap to Grid를 의미합니다)

Snap Interval: 스냅 간격에 대한 거리 값입니다. (스냅 Snap Enable 이 True 일 때만 사용됩니다)

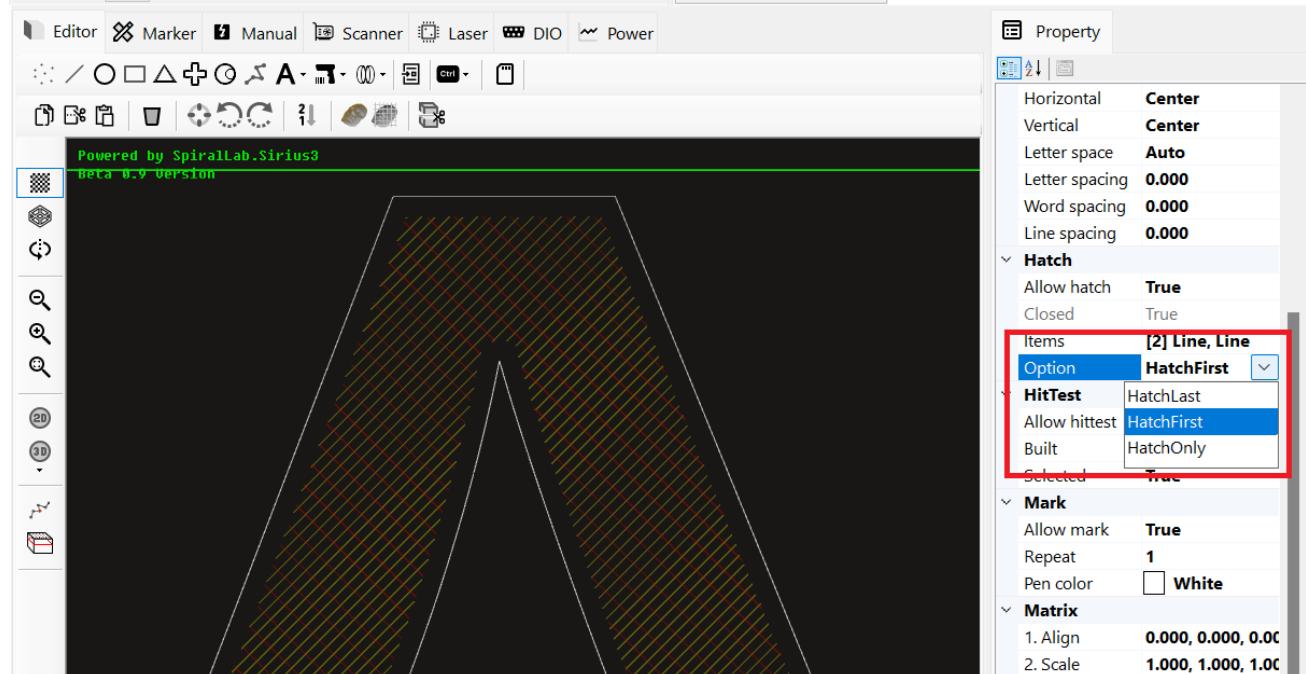
생성된 선분 해치를 예제를 보면,





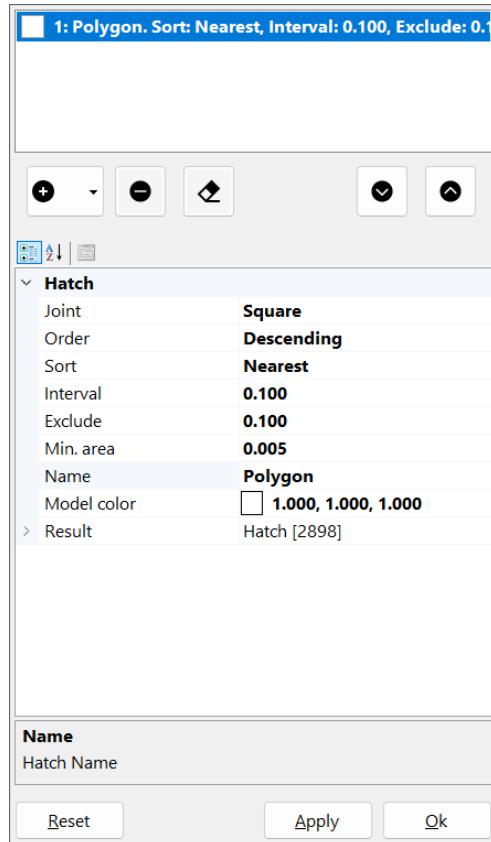
추가적으로 Angle: -45도, 펜 색상: 빨강 등을 추가한 예

Apply(혹은 Ok)를 통해 이렇게 생성된 해치는 아래와 같이 2 개의 선분 해치들로 구성된 것을 확인할 수 있습니다.



또한 Hatch Option 을 통해 외곽 + 해치의 가공 방식(해치를 먼저 가공할지, 외곽선을 먼저 가공하지 등) 변경할 수 있습니다.

15.2 폴리곤 해치



폴리곤(Polygon) 해치 옵션 설명

- **Hatch Joint:** 볼록한 각도의 접합부에서 오프셋을 관리하는 방식을 지정합니다. 오목한 접합부는 항상 Miter 접합으로 오프셋 됩니다.

- **Square:** 사각형 조인트 유형. 볼록한 접합부는 '스퀘어링' 모서리로 잘립니다. 그리고 이 스퀘어링 모서리의 중간점은 원래(또는 시작) 정점에서 정확히 오프셋 거리만큼 떨어져 있습니다.

- **Round:** 둥근 조인트 유형. 모든 볼록한 조인에 반경이 오프셋 거리인 원호가 적용되며, 원래 조인 정점이 원호의 중심이 됩니다.

- **Miter:** 마이터 조인트 유형. 가장자리는 먼저 원래(즉 시작) 가장자리 위치에서 지정된 거리만큼 평행하게 오프셋 됩니다. 이 오프셋된 가장자리는 인접한 가장자리 오프셋과 교차하는 지점까지 연장됩니다.

Hatch Order: 폴리곤 생성 순서를 지정합니다.

- **Ascending:** 오름차순 정렬됩니다. (내부에서 외부 순서로)

- **Descending:** 내림차순 정렬됩니다. (외부에서 내부 순서로)

Hatch Sort: 폴리곤들 간의 정렬 상태를 지정합니다.

- **None:** 생성된 순서를 그대로 이용합니다. (정렬 없음)
- **Nearest:** 가장 가까운(인접한) 폴리곤을 탐색하여 정렬합니다.
- **Global:** 폴리곤 들을 전역적으로 경로 최적화를 통해 정렬합니다.

Hatch Interval: 폴리곤 생성시 폴리곤 간 간격을 지정합니다. (0 보다 커야 합니다)

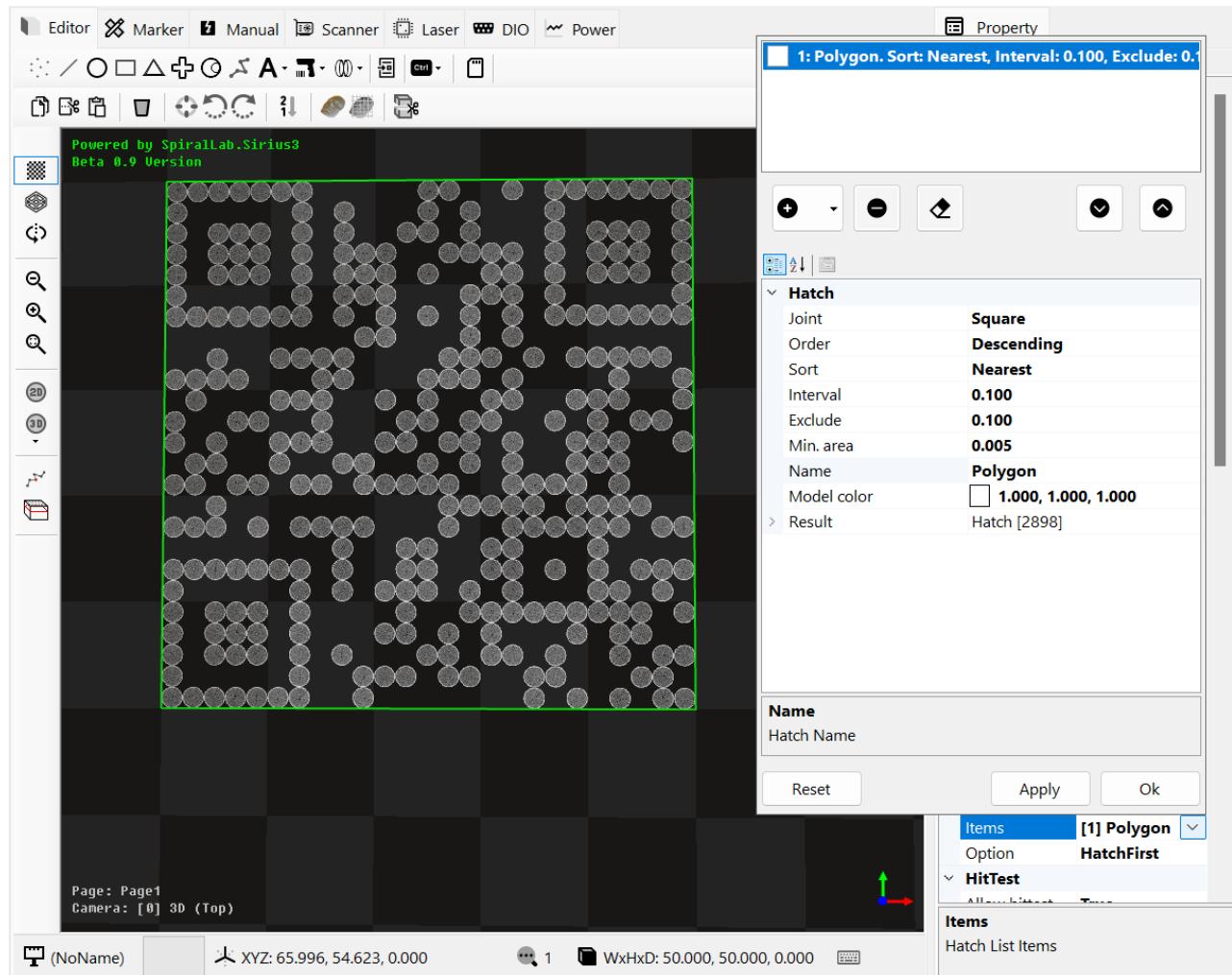
Hatch Exclude: 제외할 영역에 대한 외곽 거리 값입니다. 폐곡선이 해당 값만큼 축소된 후 해치 생성이 이루어집니다.

Hatch Min. Area: 폴리곤 생성시 제외한 최소 면적 크기입니다.

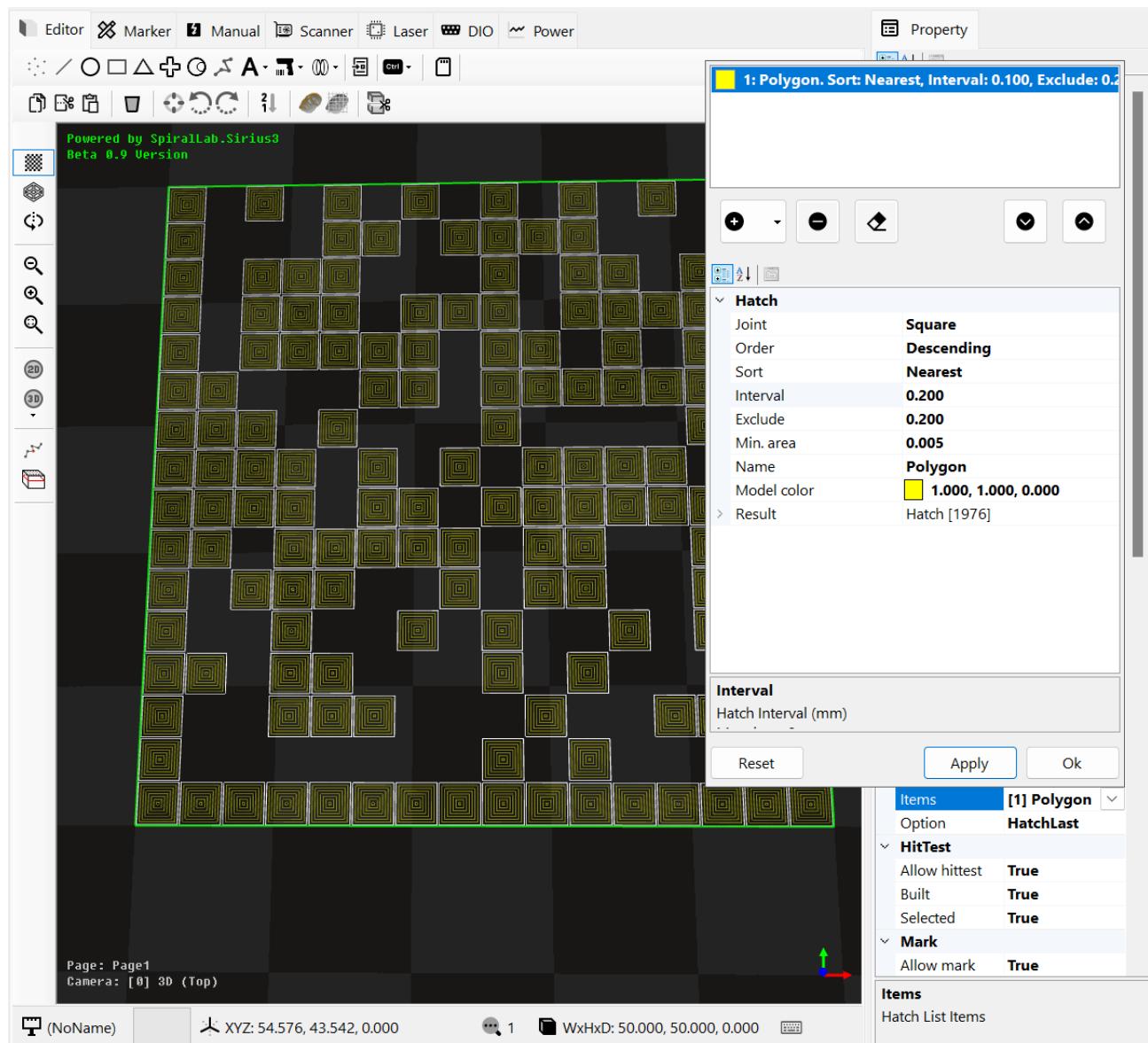
Hatch Model color: 해치에 사용할 펜 색상(Pen color)을 지정합니다. 색상 변경을 통해 레이저 가공시 서로 다른 스캐너 펜(Scanner pen) 속성들을 적용해 가공할 수 있습니다.

Hatch Result: 내부적으로 생성된 결과 저장용 객체입니다.

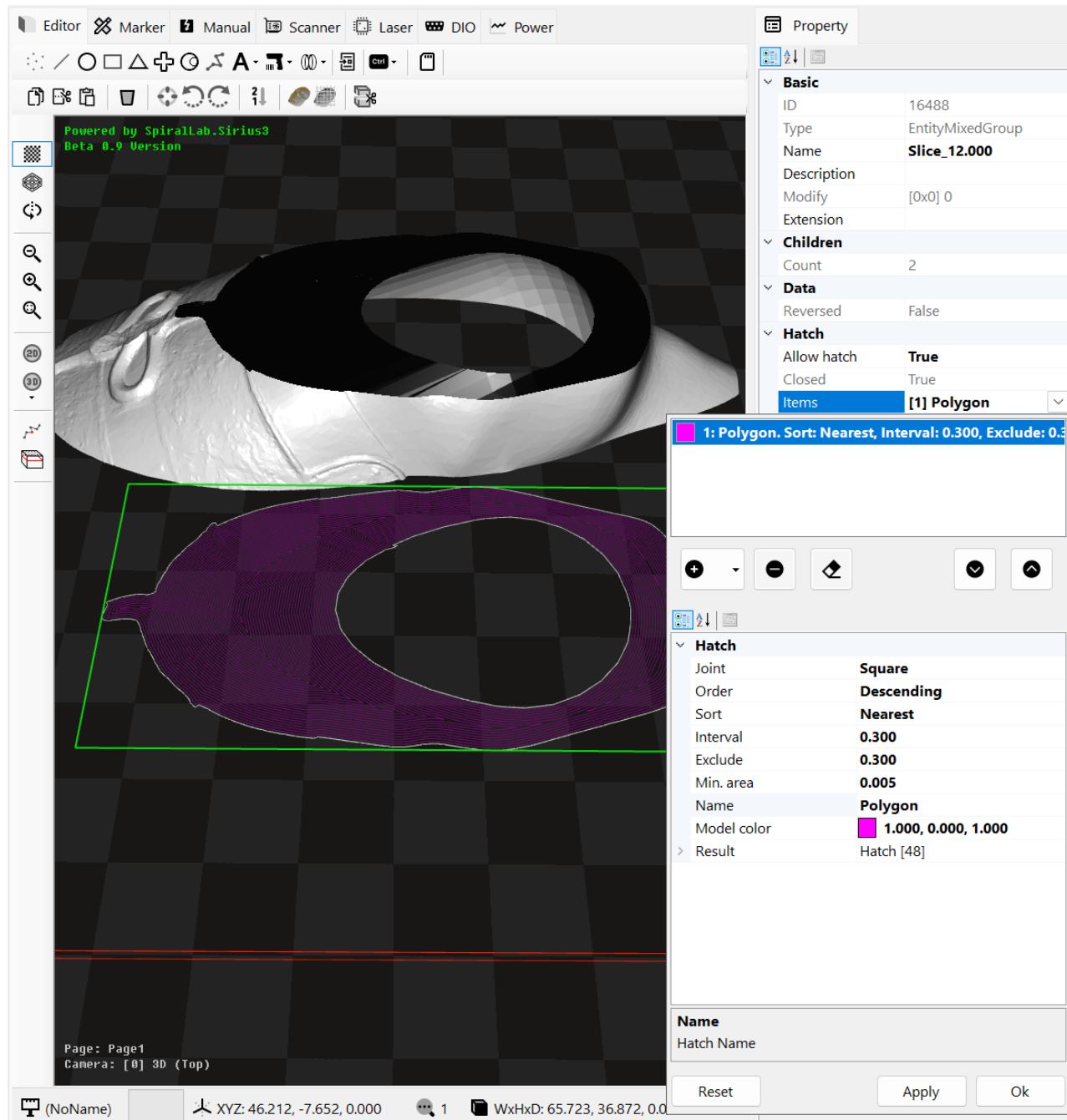
생성된 폴리곤 해치를 예제를 보면,



원 형태의 셀(Cell Circle)로 구성된 QR 바코드에 대해서 Descending 내림차순 (외부에서 내부로), Interval은 0.1 mm 등으로 생성한 예



사각형 형태의 셀(Cell Square)로 구성된 DataMatrix 바코드에 대해서 Descending 내림차순 (외부에서 내부로), Interval 은 0.1 mm 등으로 생성한 예



메쉬(Mesh)에 대해 Slice 후 생성된 2개의 폴리 라인(EntityPolyline2D)에 대해 폴리곤 해치를 적용한 모습

15.3 코드를 통한 해치 처리

해치를 지원하는 개체는 `IHatchable` 인터페이스를 상속 구현하고 있고, `HatchFactory`에서 제공하는 함수를 이용하면 다음과 같이 해치 객체를 생성 및 추가할 수 있습니다.

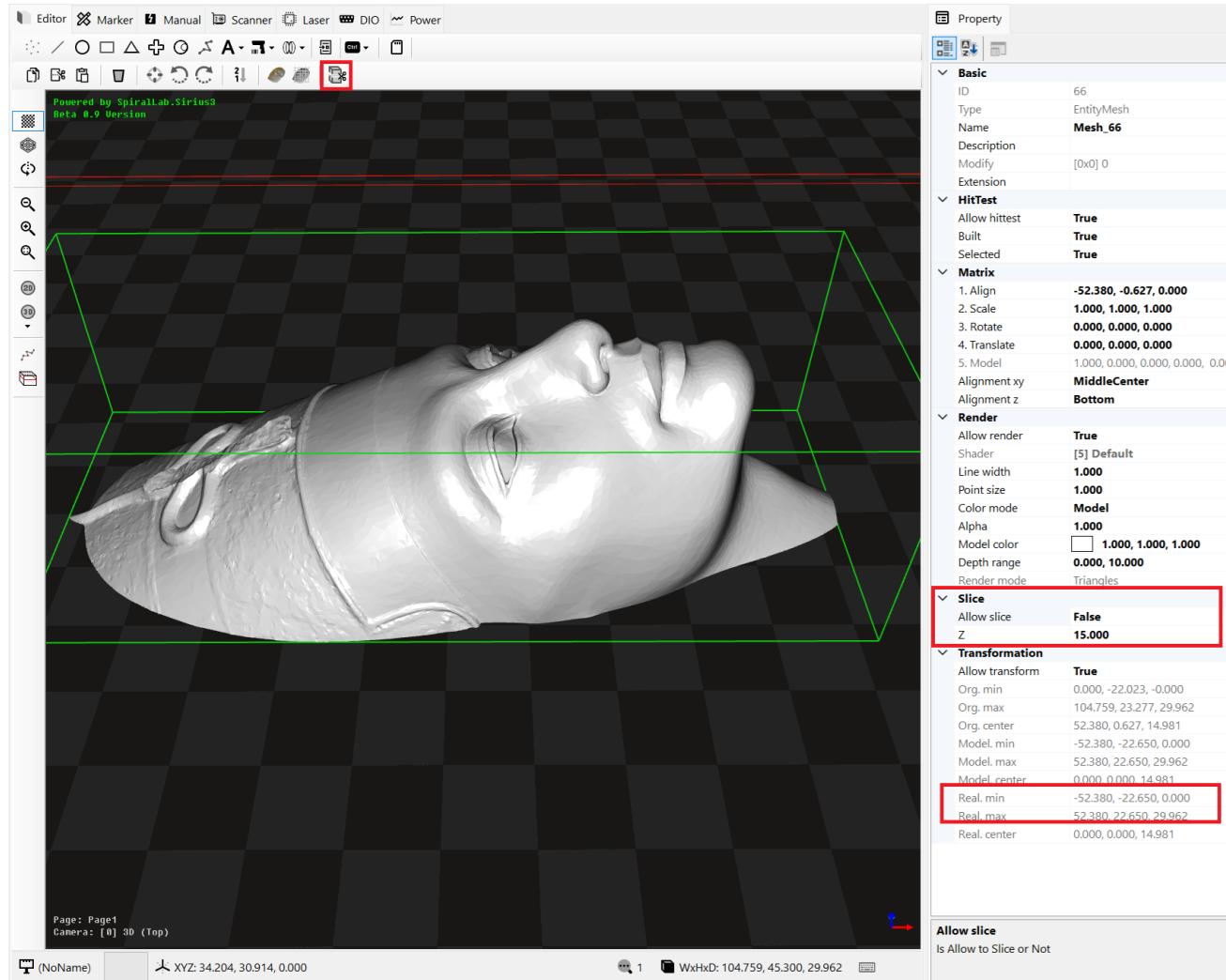
```
var hatch = HatchFactory.CreateLine(...);
//var hatch = HatchFactory.CreatePolygon(...);
hatch.ModelColor = Color.White;

var hatchable = entity as IHatchable;
hatchable.AddHatch(hatch); // hatchable.Hatches에 해치 객체 정보가 추가됨
document.ActRegen();
```

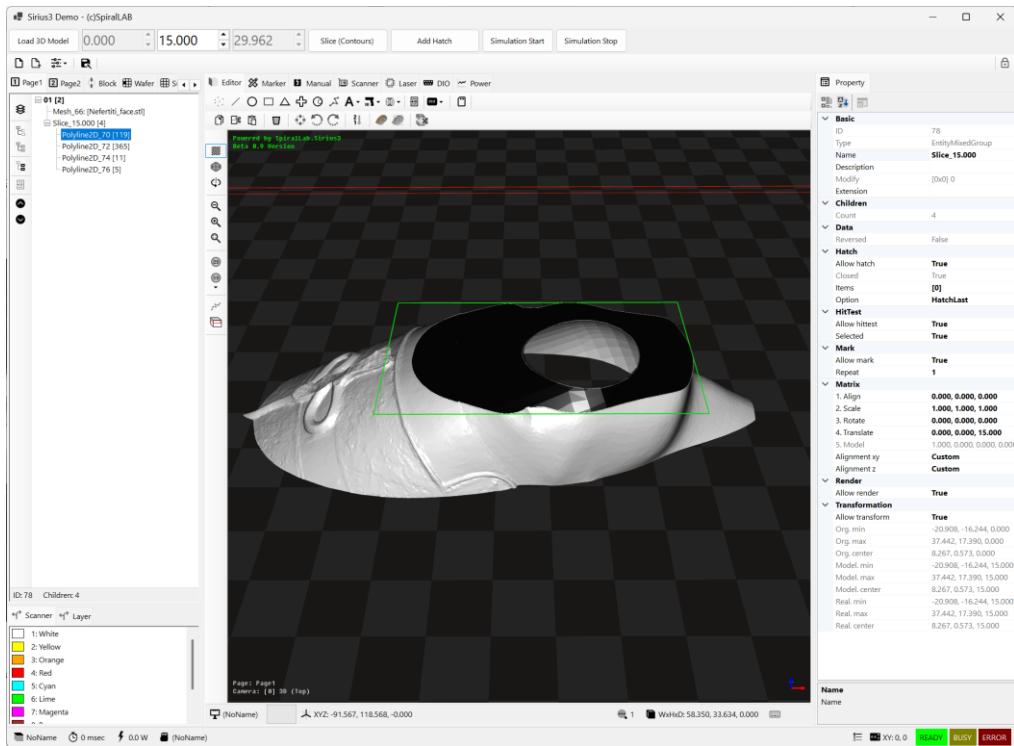
(참고) 자세한 사용법은 `editor_hatch` 예제 프로젝트의 코드를 참고해 주시기 바랍니다.

16. 슬라이서 (Slicer)

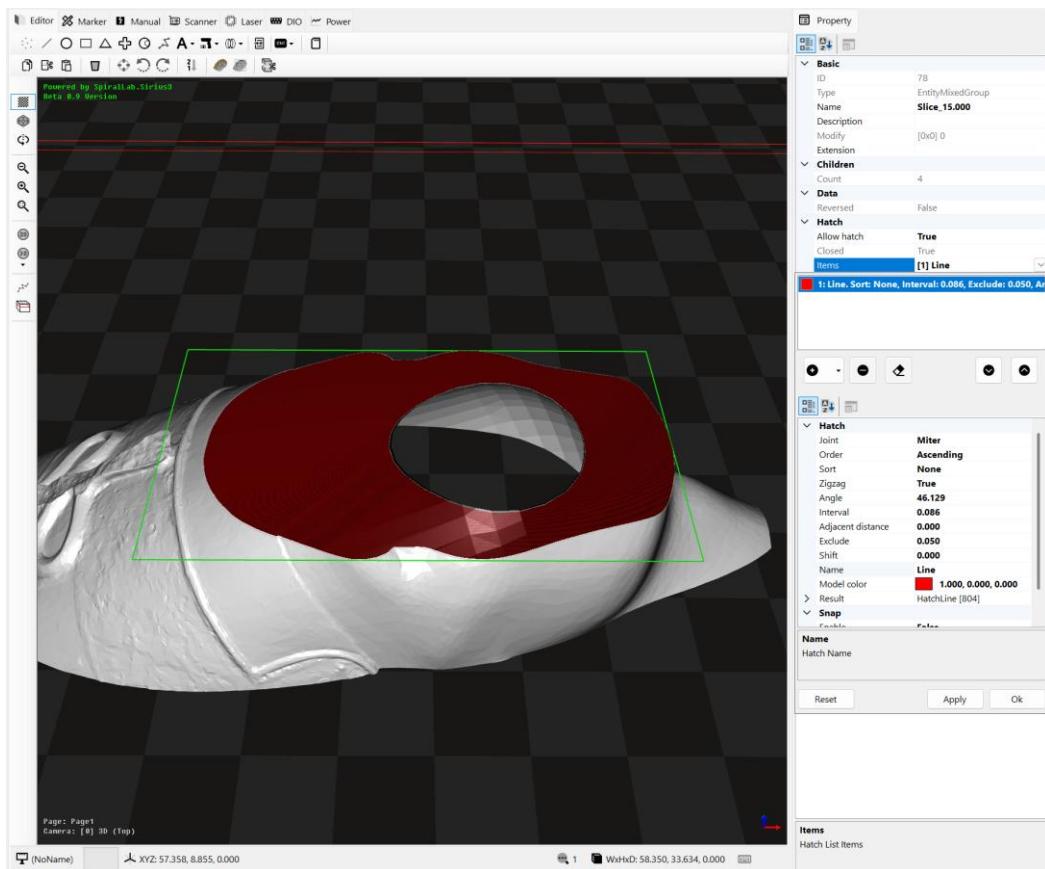
파일 가져오기(Import)를 통해 STL, OBJ, PLY 과 같은 3D 메쉬 모델(Mesh Model)을 가져와 특정 Z 위치로 평면을 자르고 그 경로(Contour)를 추출할 수 있습니다. 예를 들어 아래와 같은 메쉬 객체를 Import 하면 Z=0 위치에 바닥면이 자동으로 정렬(Align) 되어 로딩됩니다.



이후 Slice Z 값을 15mm로 설정(Real. Min ~ Real. Max의 Z 범위 내에 해당하는지 확인)하고, Allow slice 항목을 활성화(True)하게 되면, 지정된 Z=15 평면에서 메쉬가 잘린채 렌더링됩니다. 이때 Slice Mesh 버튼을 누르게 되면 해당 메쉬에 지정된 Z 평면위치를 사용해 잘린 평면을 구성하는 경로(Contour)들을 추출하고, 이를 폴리라인(Polyline) 들로 모아 하나의 그룹 객체를 생성해 추가하게 됩니다.



Slice Mesh 기능으로 평면으로 절단 후 생성된 Contour 그룹 개체

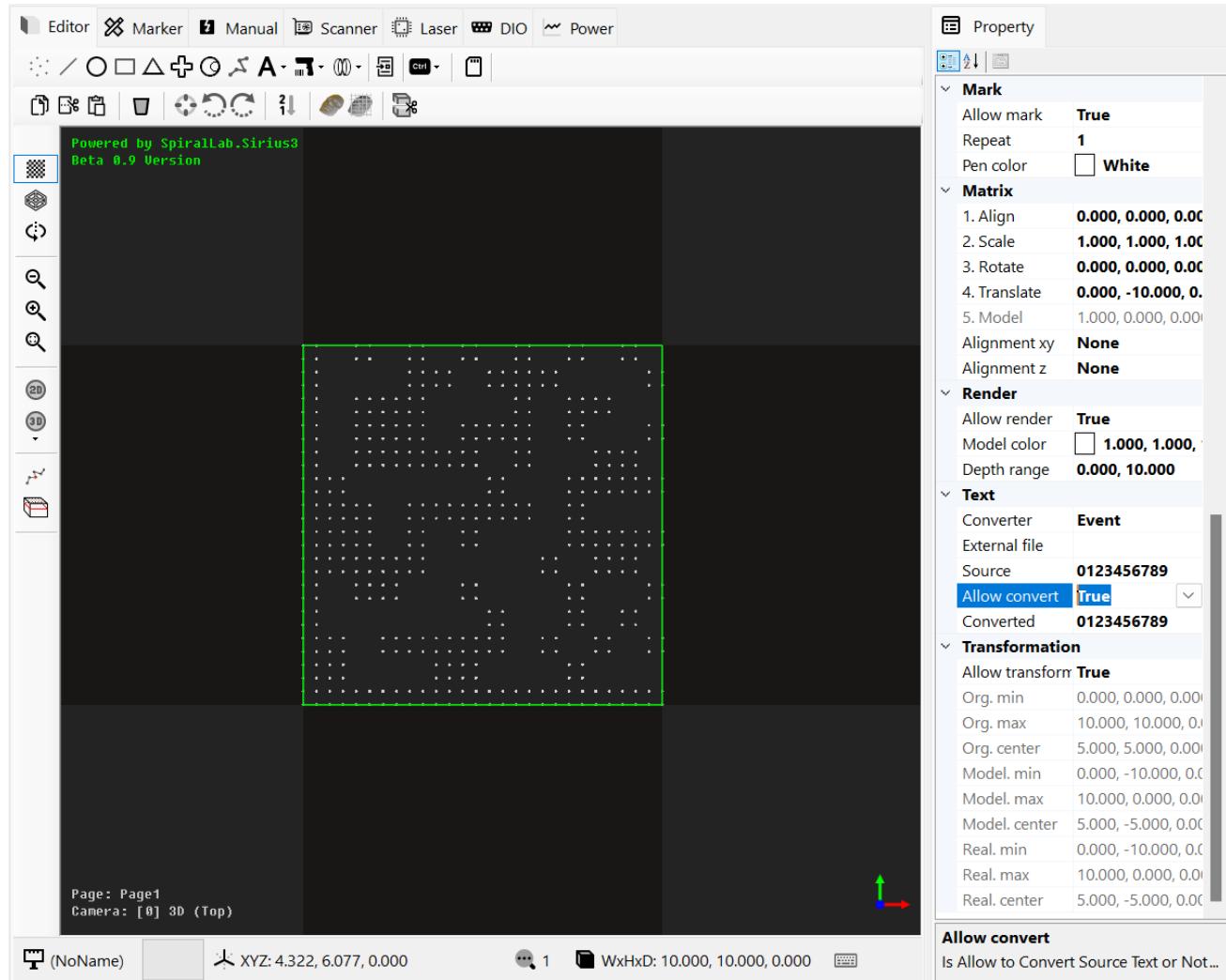


Contour 그룹 개체에 해치(Hatch)를 적용한 모습

3D 메수에 Slice Z 위치값을 변경하면서 절단면에 해치를 적용하고 이를 레이저 가공하는 방식을 반복적으로 활용하면 3D 레이저 프린터 가공 방식인 AM(Additive Manufacturing)이 가능해집니다.

17. 텍스트 데이터 변환 (Text Converter)

텍스트 데이터 변환을 지원하는 개체들²은 레이저 가공 직전 최종 데이터 변환이 적용됩니다. 이를 위해 아래와 같이 **Allow Convert** 기능을 활성화하고 변환기(Converter)를 선택해야 합니다. 또한 입력된 데이터는 **Source**, 최종 변환된 데이터는 **Converted** 항목에 반영되어 출력됩니다.



텍스트 데이터 변환기(Converter)는 4가지가 제공됩니다.

17.1 이벤트 (Event)

이벤트는 라이브러리 내부에서 제공되는 **IMaker.OnTextConvert** 이벤트에 함수를 등록해 사용하는 방식입니다. 이벤트 핸들러 함수에서 리턴(return)된 문자열을 사용하는 방식으로, 해당 코드를 직접 구현해 사용하므로, 복잡한 구현이 가능합니다.

코드 구현 예:

```
var marker = siriusEditor1.Marker;
marker.OnTextConvert += OnTextConvert;
...
```

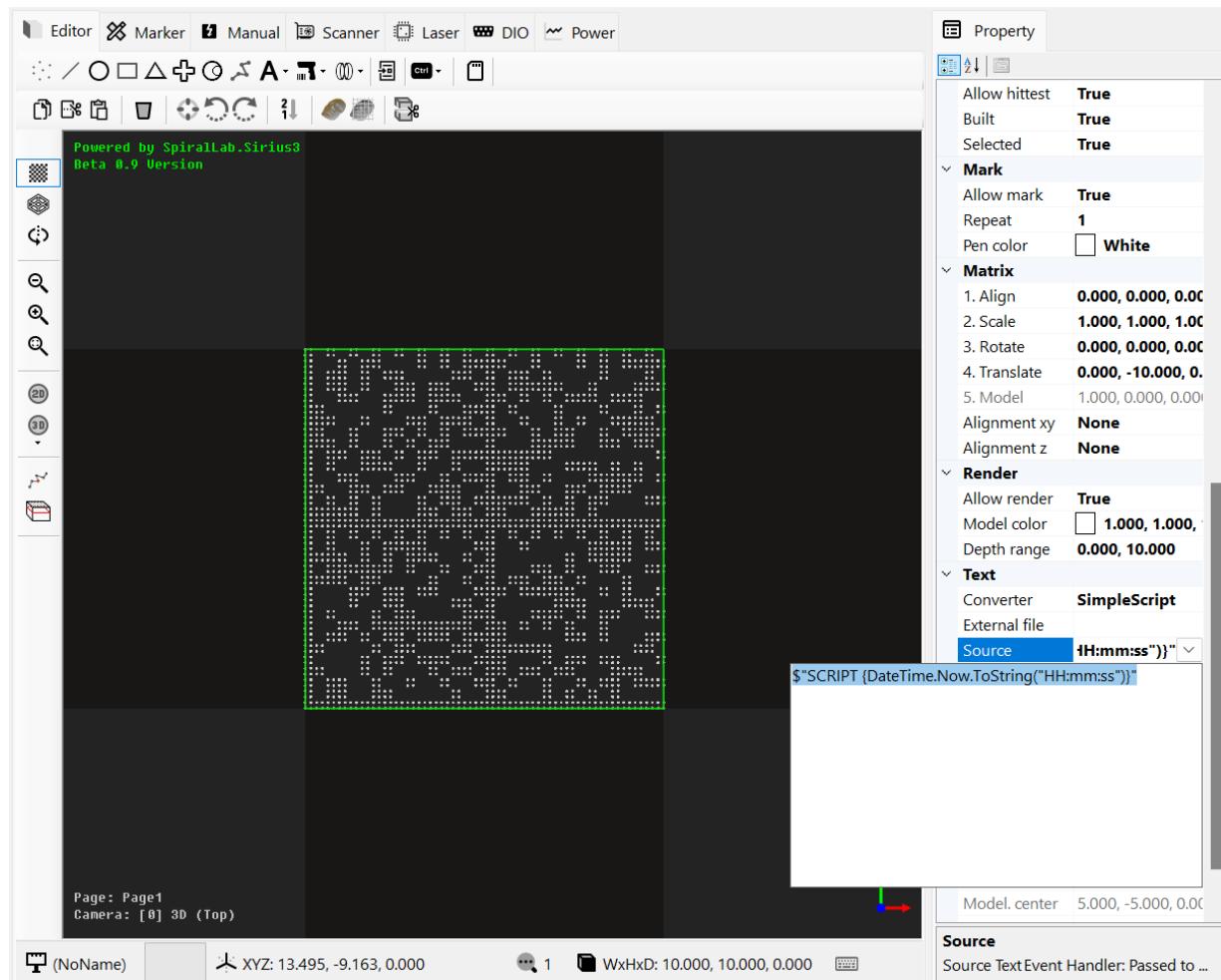
² 텍스트, 바코드 등의 개체(EntityText, EntitySiriusText, EntityCircularText, EntityBarcode1D, EntityQRCode, EntityDataMatrix 등)들 과 같이 **ITextConvertible** 를 상속 구현하는 객체들

```
string OnTextConvert(IMarker marker, ITextConvertible textConvertible)
{
    var entity = textConvertible as IEntity;
    switch (entity.Name)
    {
        case "MyBarcode":
            return $"EVENT {DateTime.Now.ToString("HH:mm:ss")}";
        default: // Not modified
            return textConvertible.SourceText;
    }
}
```

17.2 단순 스크립트 (Simple Script)

스크립트 변환기는 입력 데이터에 간이 C# 스크립트 코드를 입력하고, 레이저 가공 직전에 해당 스크립트 코드가 실행되어 리턴(return)된 문자열을 사용하는 방식입니다. 아래와 같이 매우 단순한 코드만 사용이 가능합니다.

C# 스크립트 구현 예: `$"SCRIPT {DateTime.Now.ToString("HH:mm:ss")}"`



17.3 파일 (File)

외부 텍스트 파일을 지정하고 그 내용을 사용하는 방식입니다. 레이저 가공 직전에 해당 텍스트 파일의 가장 첫번째 라인을 읽어 문자열을 레이저 가공에 사용하는 방식입니다. (해당 문자열은 자동으로 삭제됩니다)

때문에 파일 변환기 사용시에는 반드시 외부파일(External File)을 미리 지정해 두어야 합니다. 또한 파일의 문자열이 모두 소진(삭제)되면, 더 이상 가공이 진행되지 않습니다.

17.4 오프셋 (Offset)

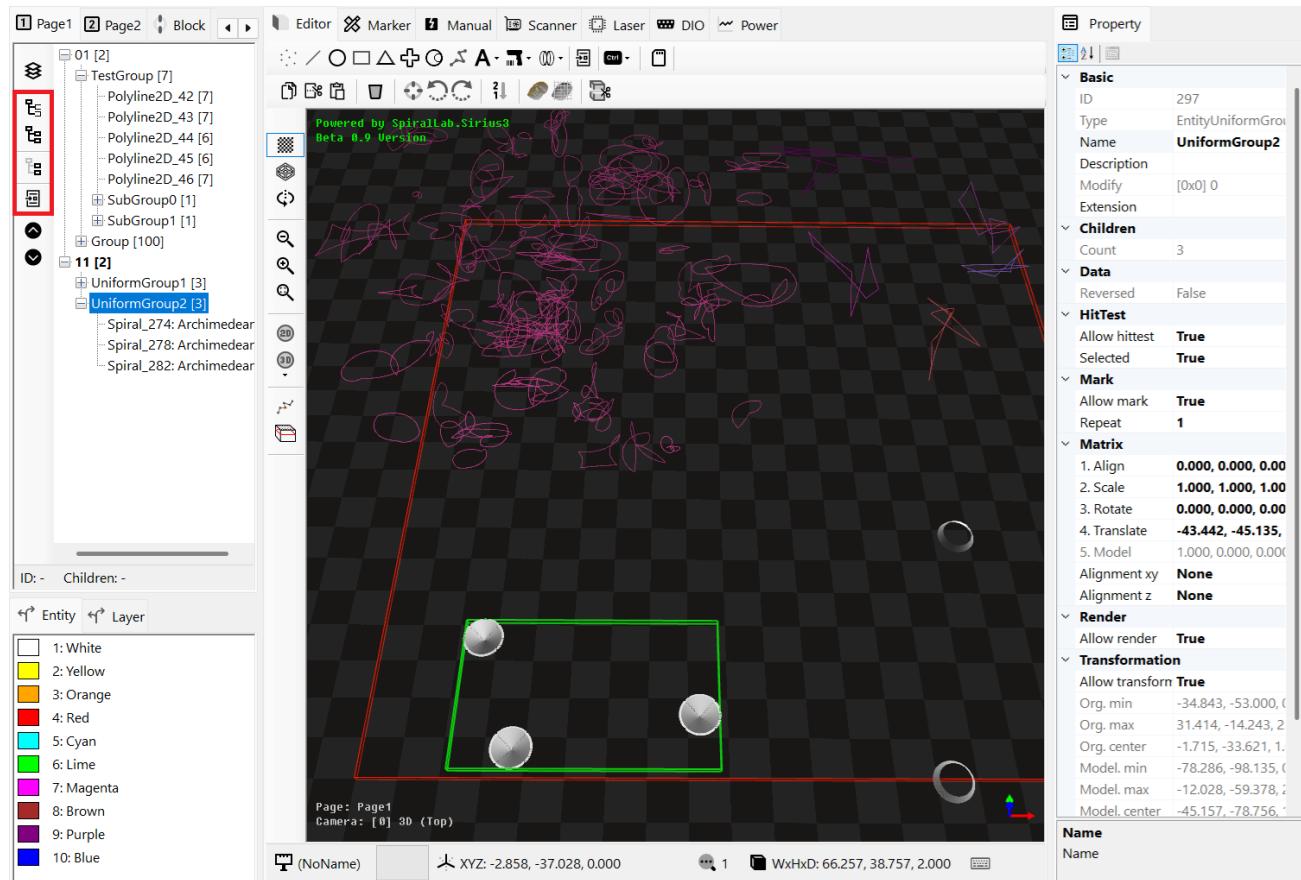
마커에서 가공 시작시 오프셋(Offset) 배열을 전달하면, 여러 위치에 반복적인 가공이 가능합니다. 가공이 진행중 오프셋(Offset) 객체의 `ExtensionData` 속성에 저장된 값을 읽어 텍스트 변환 가공해 사용하는 방식입니다.

(주의) 리턴(return) 된 문자열이 없으면 (`null` 혹은 `string.Empty`) `Marker` 의 가공이 중단됩니다.

(참고) 자세한 사용방법은 `editor_barcode_textconvert` 예제 프로젝트의 코드를 참고해 주시기 바랍니다.

18. 그룹(Group) 및 블록(Block)

다양한 개체(IEntity)들을 하나의 그룹(EntityGroup)으로 생성하기 위해서 페이지 트리뷰 왼쪽에는 다양한 그룹 개체 생성 기능이 제공됩니다.



1. Mixed Group: 선택된 개체(IEntity)들을 하나의 혼합 그룹으로 변환 생성합니다.

(참고1) 트리뷰(TreeView)의 동일한 레벨(깊이)의 노드들만 허용됩니다.

(참고2) 레이어(EntityLayer), 펜(EntityPen) 등의 개체들은 혼합될 수 없습니다.

2. Uniform Group: 선택된 개체(IEntity)들을 하나의 단일 그룹으로 변환 생성합니다.

(참고1) 트리뷰(TreeView)의 동일한 레벨(깊이)의 노드들만 허용됩니다.

(참고2) 레이어(EntityLayer), 펜(EntityPen) 등의 개체들은 혼합될 수 없습니다.

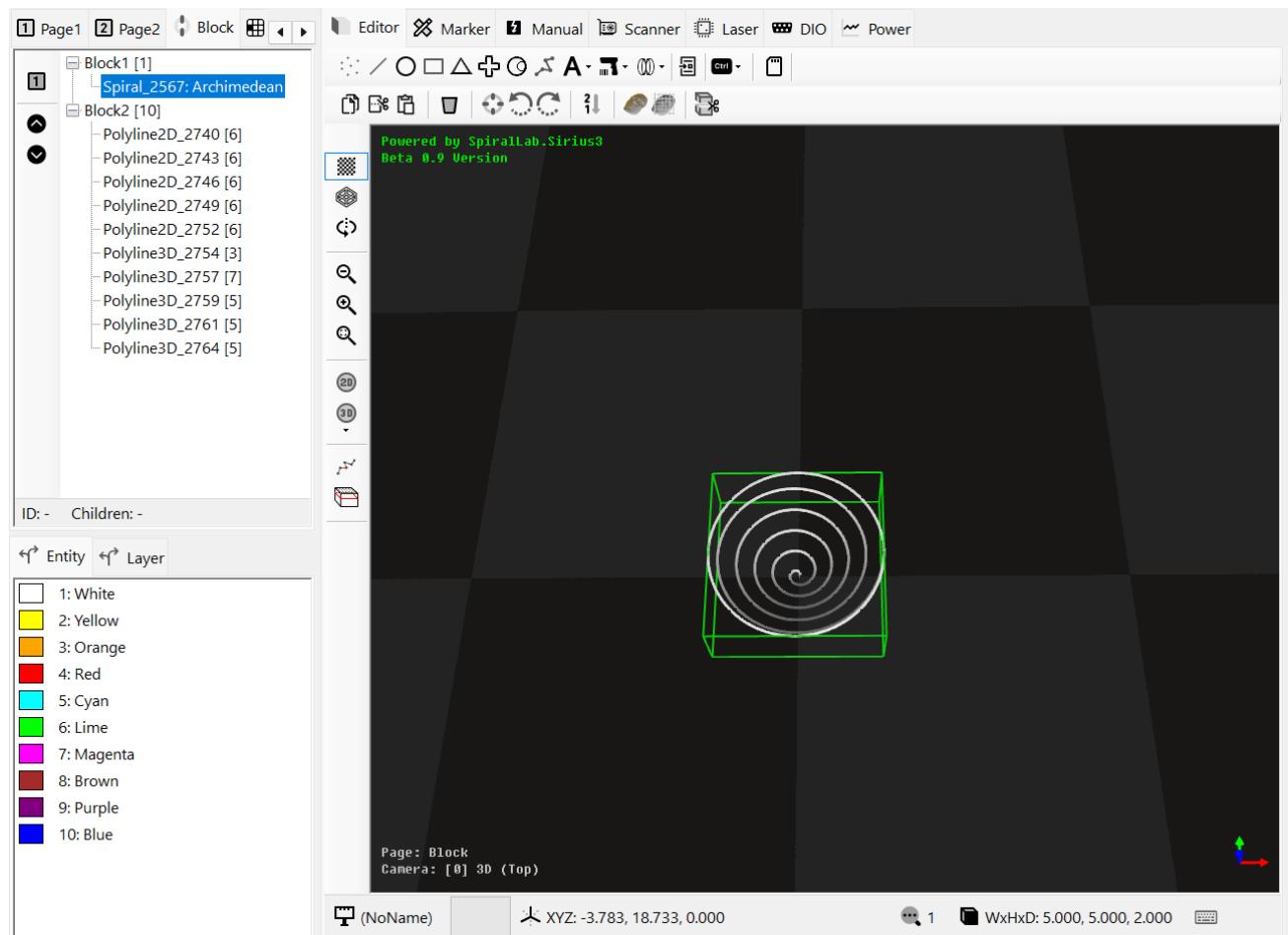
(참고3) 단일 그룹은 고속 렌더링 (하나의 VAO로 처리)을 위한 특수 목적용으로 제공되기 때문에, 각 개체들이 모두 동일한 렌더링 모드(RenderMode)를 가져야 합니다.

3. Ungroup: 선택된 그룹 개체를 해체해 개별 개체들로 다시 복구합니다.

4. Block: 선택된 개체들을 블록(EntityBlock) 개체로 변환 생성 합니다.

(참고1) 레이어(EntityLayer), 펜(EntityPen) 등의 개체들은 혼합될 수 없습니다.

(참고2) 생성된 블록 개체는 Block 페이지에 생성되며, 추후 블록 삽입(BlockInsert)을 통해 다수의 참조 개체로 사용할 수 있습니다. BlockInsert 된 참조 개체들은 다양한 변환(크기, 회전, 이동 등) 적용이 가능합니다.

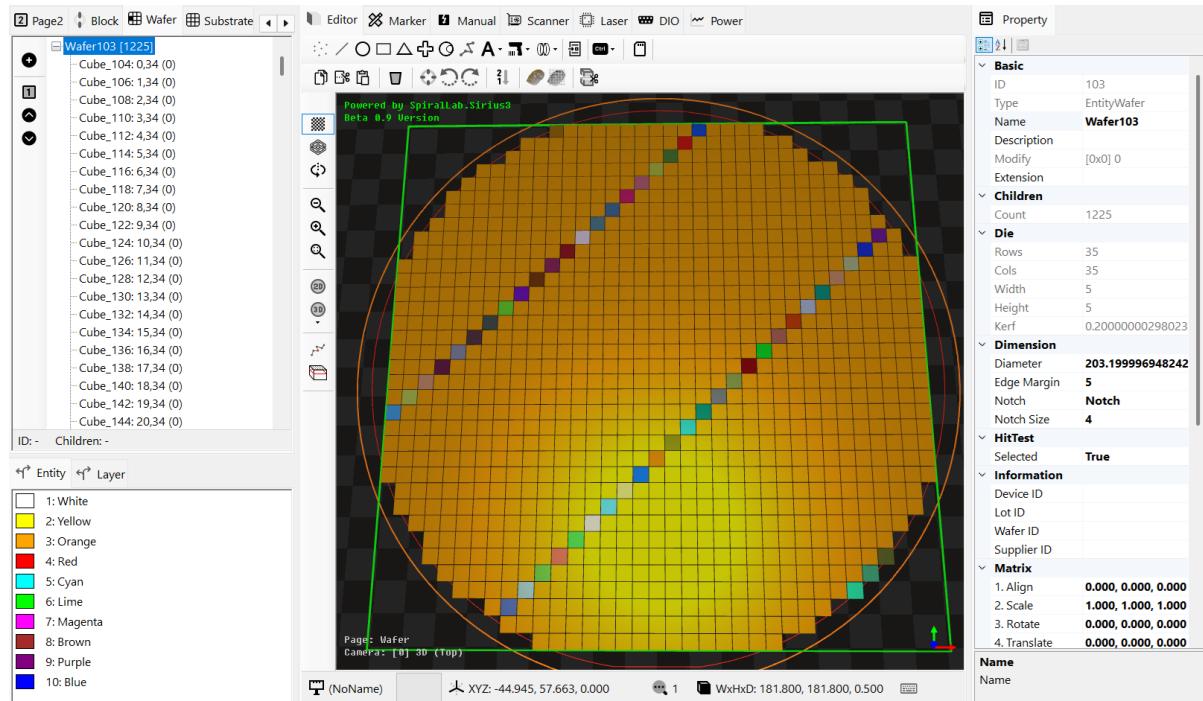


블록 페이지에서는 생성된 블록(EntityBlock) 목록이 나타나며, 그중에 제일 첫(상단) 위치에 있는 블록만 실제 화면에 렌더링 됩니다. 만약 다른 블록을 렌더링 시켜 편집 등의 작업이 필요하다면 좌측의 ‘1’ 버튼 (Set as First or Top)을 눌러 위치를 이동이 가능합니다.

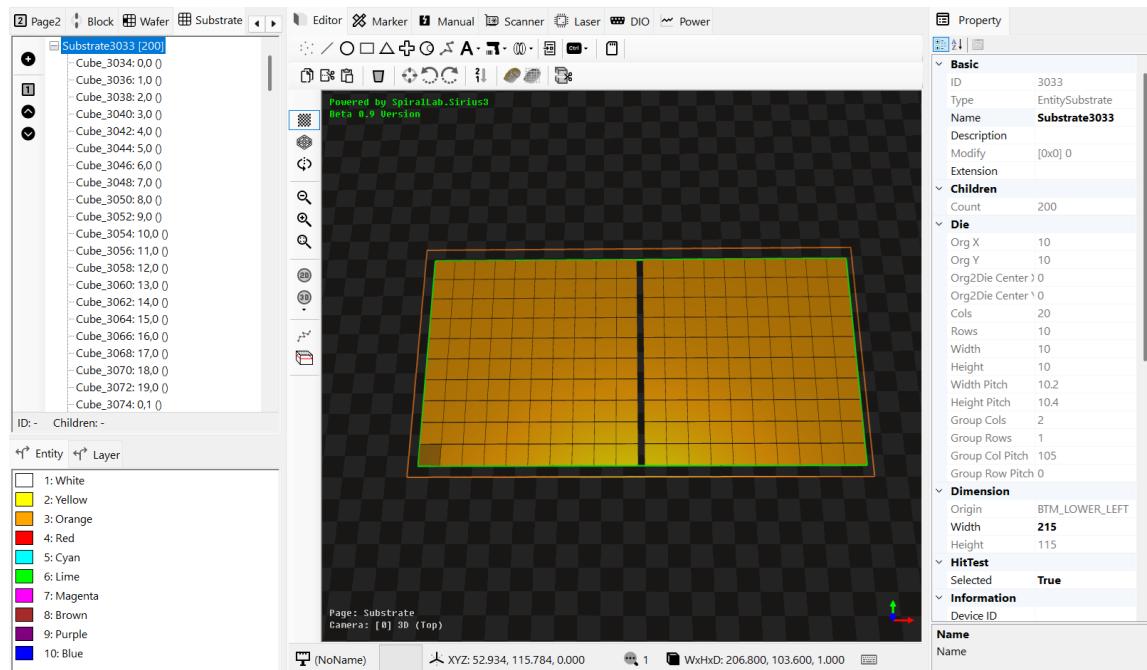
(주의) 블록 페이지에서 블록(EntityBlock) 개체를 삭제하면 이를 참고하고 있는 블록 삽입 (EntityBlockInsert) 개체들이 무효화되기 때문에, 불필요해진 개체들을 모두 사용자가 직접 삭제해 주어야 합니다.

19. 웨이퍼(Wafer), 기판(Substring) 맵(Map)

웨이퍼 (혹은 기판) 맵을 시각화하는 기능을 제공합니다. '+' 버튼을 누르면 신규 웨이퍼(혹은 기판) 맵이 추가됩니다. 그러나 대부분 특정 크기 정보를 가진 Die 들을 직접 생성하는 것이 일반적이므로, **Config**. **OnCreateWafer** 이벤트에 핸들러 함수를 등록해 직접 웨이퍼 (혹은 기판) 맵 생성이 가능합니다.

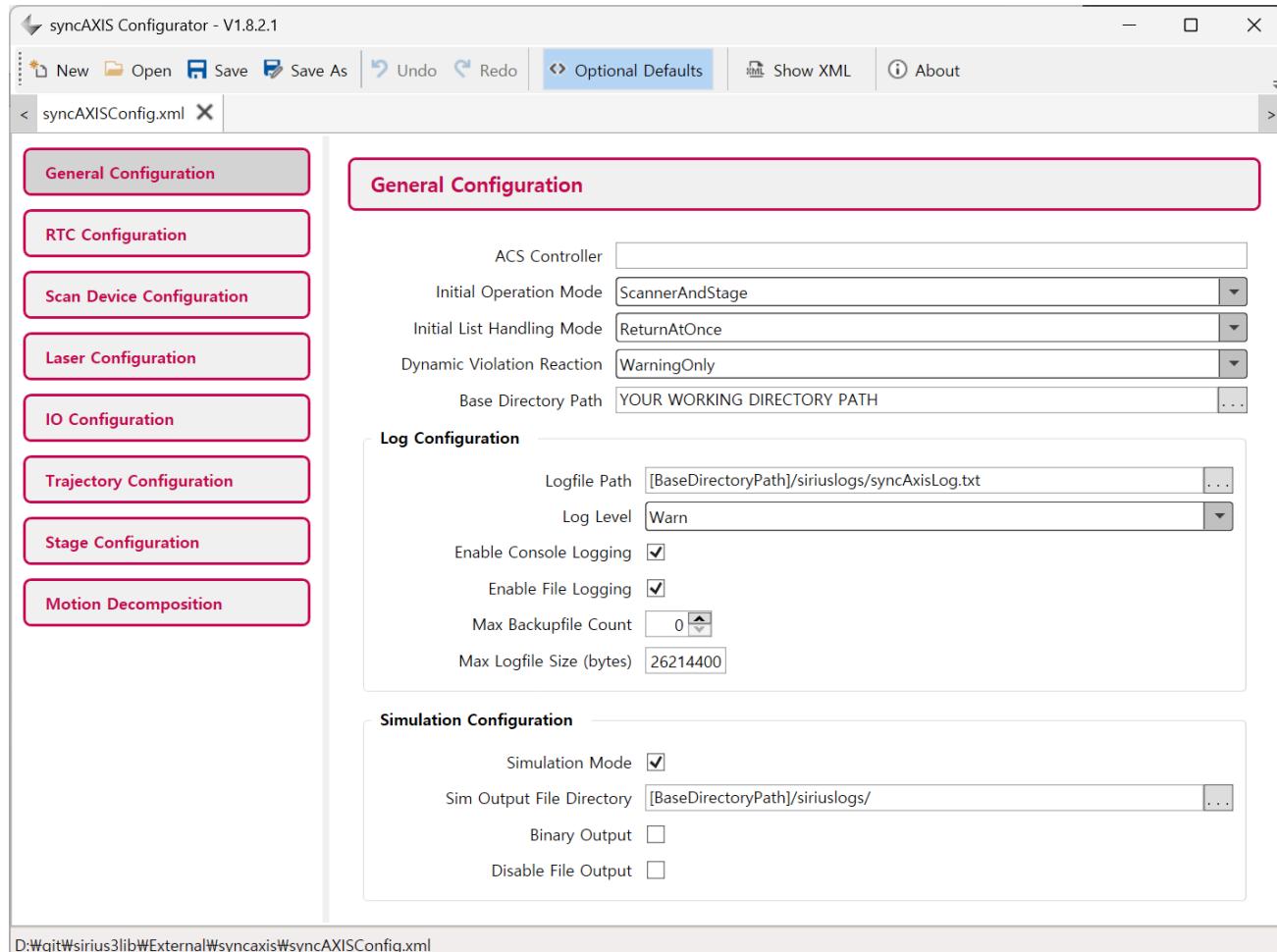


웨이퍼 (혹은 기판) 맵 페이지에서는 그중에 제일 첫(상단) 위치에 있는 맵 개체만 실제 화면에 렌더링 됩니다. 만약 다른 맵을 렌더링 시켜 편집 등의 작업이 필요하다면 좌측의 '1' 버튼 (Set as First or Top) 을 눌러 위치를 이동이 가능합니다.



20. syncAXIS (XL-SCAN) 사용법

syncAXIS 를 이용해 대면적 레이저 가공 기능을 사용하기 위해서는 RTC6 (SCANahead) 제어기 + ExcelliSCAN(or IntelliSCAN IV) 스캔 헤드 + ACS 모션 제어기 구성이 만족되어야 합니다. 또한 SPIIPLUS MMI Application Studio 설치 및 스테이지에 대한 정밀 튜닝 등을 모두 거쳐 준비가 모두 완료되었다면, syncAXIS_Configurator 프로그램을 이용해 아래와 같이 설정을 모두 확인 후 syncAXISConfig.xml 와 같은 설정 파일을 생성시킵니다. 이후 실제 구동을 위해서는 SCANLAB에서 구매한 syncAXIS 전용 라이센스 동글키가 시스템에 장착되어 있어야 합니다. (Simulation Mode이라도 동글키 필수)



이때 주의할 것은 **Base Directory Path** 경로 설정이 정확한지 여부, ACS 컨트롤러의 IP 주소, 스테이지의 물리적 한계(이동 범위 한계 및 속도, 가속도, 가가속도와 같은 역학 한계 등) 설정 등이며, 안전을 위해 동적 한계 침범시 처리(Dynamic Violation Reaction)시 중지 및 보고(Stop and Report) 가 추천됩니다. 또한 **Simulation Mode**로 우선 가공 경로에 대한 다양한 검증 및 테스트 이후 실제 가공시점에 **Hardware Mode**로 전환하는 것이 바람직합니다

시리우스3 에서는 syncAXIS 인스턴스를 다음과 같이 생성 및 초기화 합니다.

```
string configXmlFilePath = Path.Combine(Config.SyncAxisPath, "syncAXISConfig.xml");
var rtc = ScannerFactory.CreateRtc6SyncAxis(index, configXmlFilePath);
rtc.Initialize();
```

또한 syncAXIS 전용 마커인 **MarkerSyncAxis** 을 생성해 사용해야 합니다.

결과적으로 초기화가 성공하면 아래와 같은 로그가 출력되어야 합니다.

Time	Level	Message
10:14:39. 674	Information	GPU Vendor: ATI Technologies Inc., Renderer: AMD Radeon(TM) Graphics, OpenGL Version: 4.6.0 Compatibility Profile Context
10:14:39. 675	Information	Free GPU Texture Memory: 384MB
10:14:44. 529	Information	syncaxis [0]: handle id= 1. xml= D:\git\sirius3\bin\net481\syncaxis\syncAXISConfig.xml. v1.8.2
10:14:44. 553	Information	syncaxis [0]: motion mode has changed to Unfollow

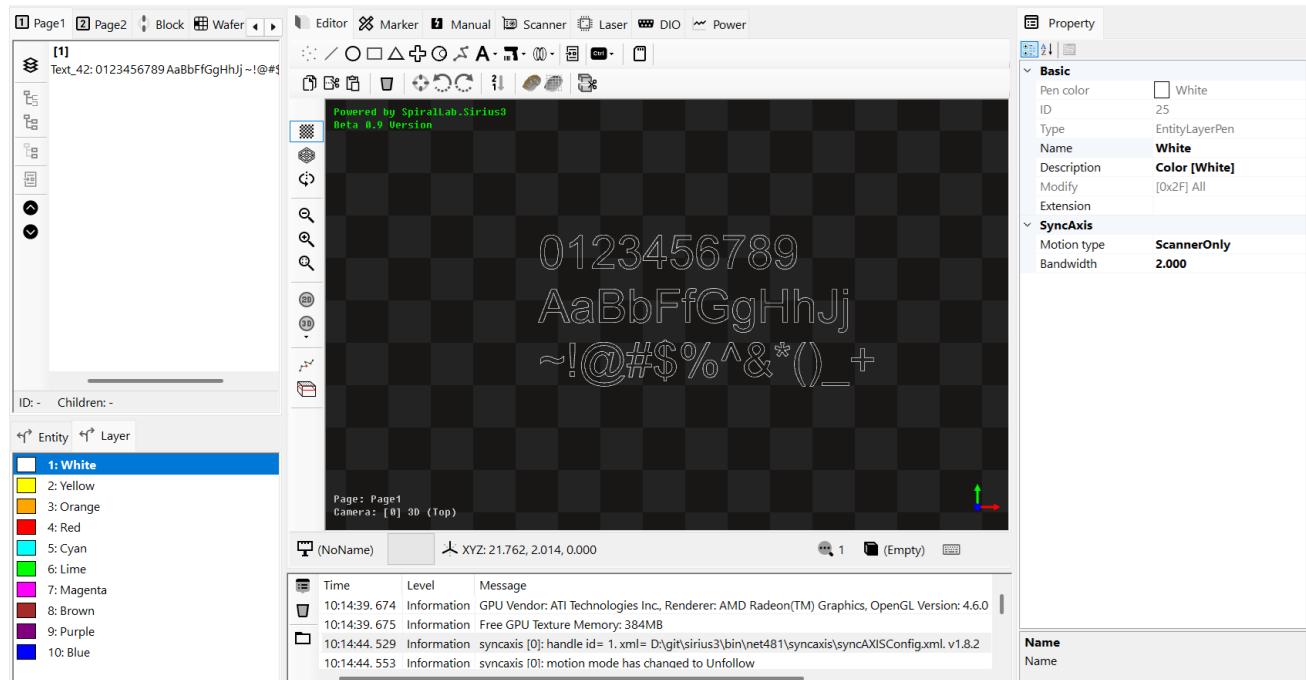
(참고1) 보정 파일은 **xml** 파일내에서 관리되므로 **CtlLoadCorrectionFile** 등의 함수들은 사용하지 않습니다.

(참고2) 디지털 입출력 포트는 확장1 포트(**Extension1**) 만 사용합니다.

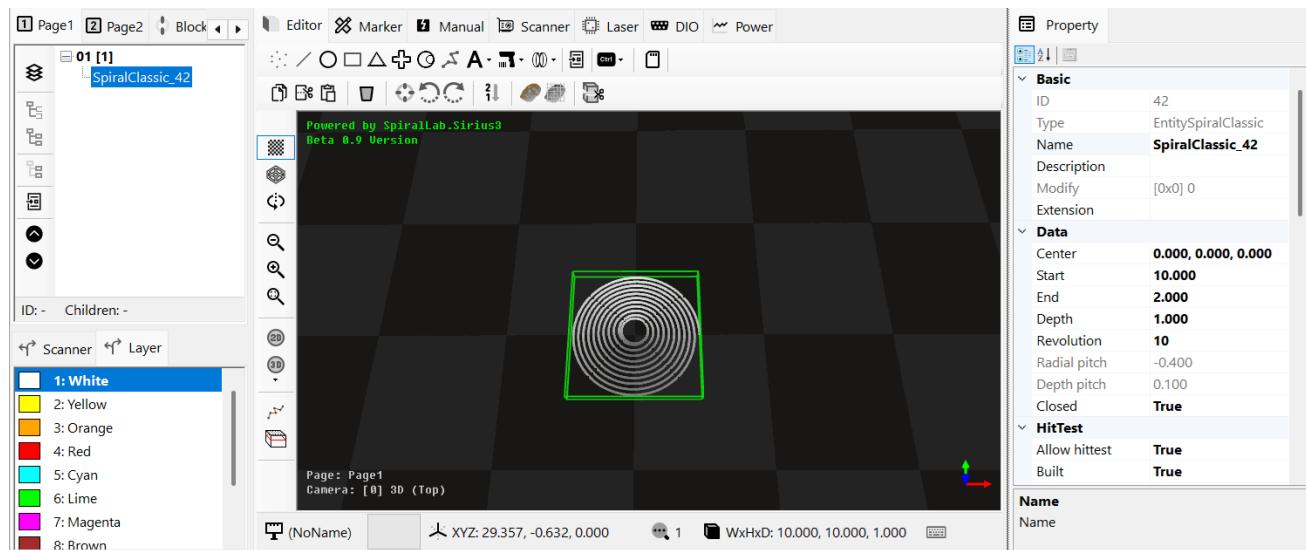
(참고3) 모든 3D 좌표(Z 좌표)는 Z=0 mm 으로 투영됩니다.

생성된 **Rtc6SyncAxis**를 편집기의 스캐너(**IScanner**) 항목에 지정하면 해당 제어기에 맞도록 다양한 속성들이 변경됩니다. 예를 들어 레이어 펜(**EntityLayerPen**)을 선택했을 경우 아래와 같이 **syncAXIS** 전용의 모션 타입, LPF 주파수 항목이 표시됩니다.

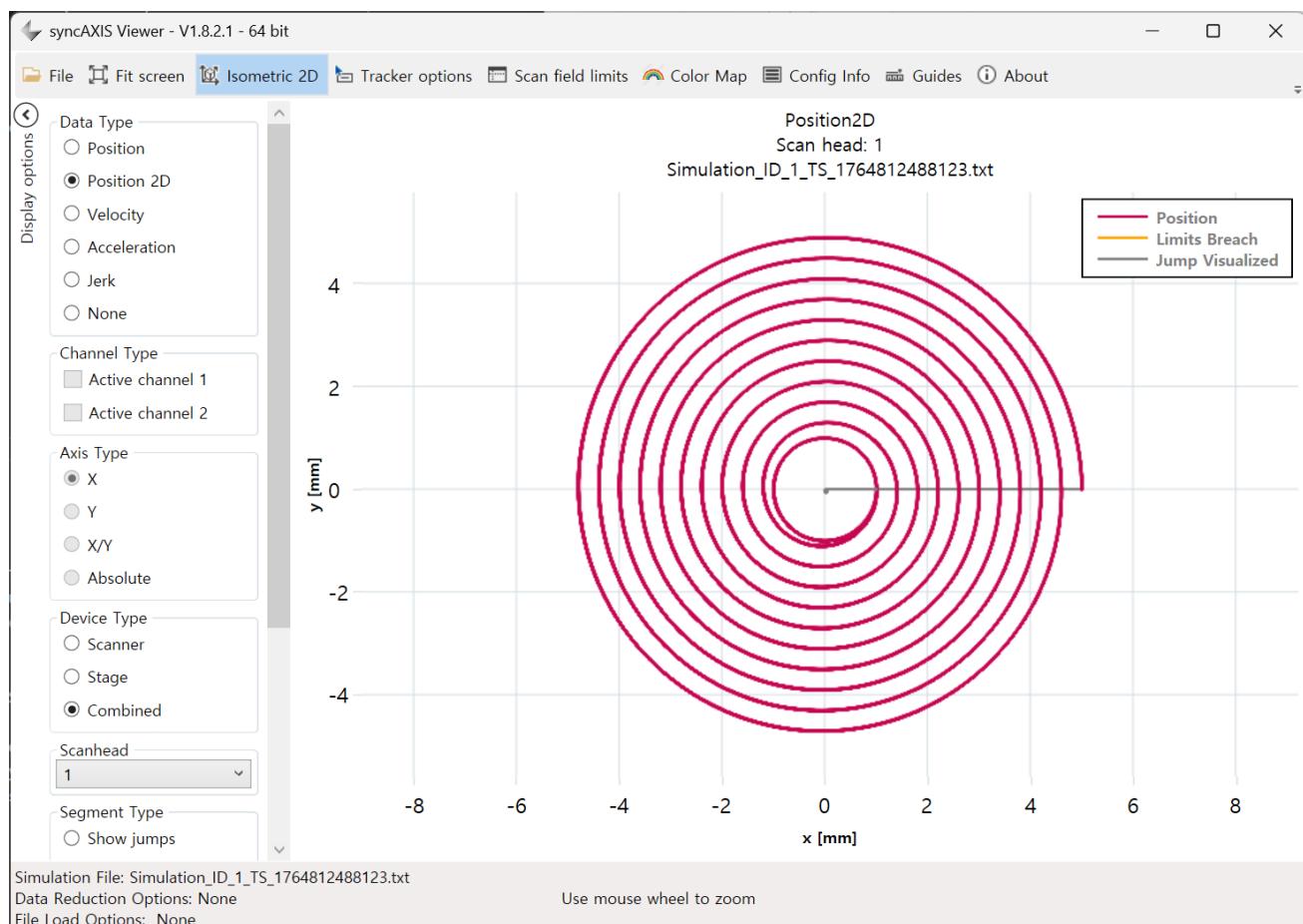
(참고) 개체 펜(**EntityPen**)의 일부 속성들도 **syncAXIS** 기능을 위해 출력이 변경됩니다.



아래에서는 간단한 나선 모양을 **Stage + Scanner** 모션 타입으로 시뮬레이션 가공을 진행한 결과 입니다.



시뮬레이션 모드(Simulation Mode)에서는 100kHz 샘플링 주기로, 스캐너와 스테이지의 경로(Trajectory)가 모두 수집되어 Sim Output File Directory에 기록됩니다. 이 출력 파일은 syncAXIS_Viewer 프로그램에 자동으로 전달되며, 시간 위치 그래프 분석, 가감속도 분석, 범위를 침범했는지 여부 등 다양한 동역학 항목을 분석할 수 있습니다.



또한 마커(Marker) 탭의 이력(History) 항목을 살펴보면, 좀더 구체적인 분석 정보가 제공됩니다.

SIRIUS3 라이브러리 사용자 매뉴얼

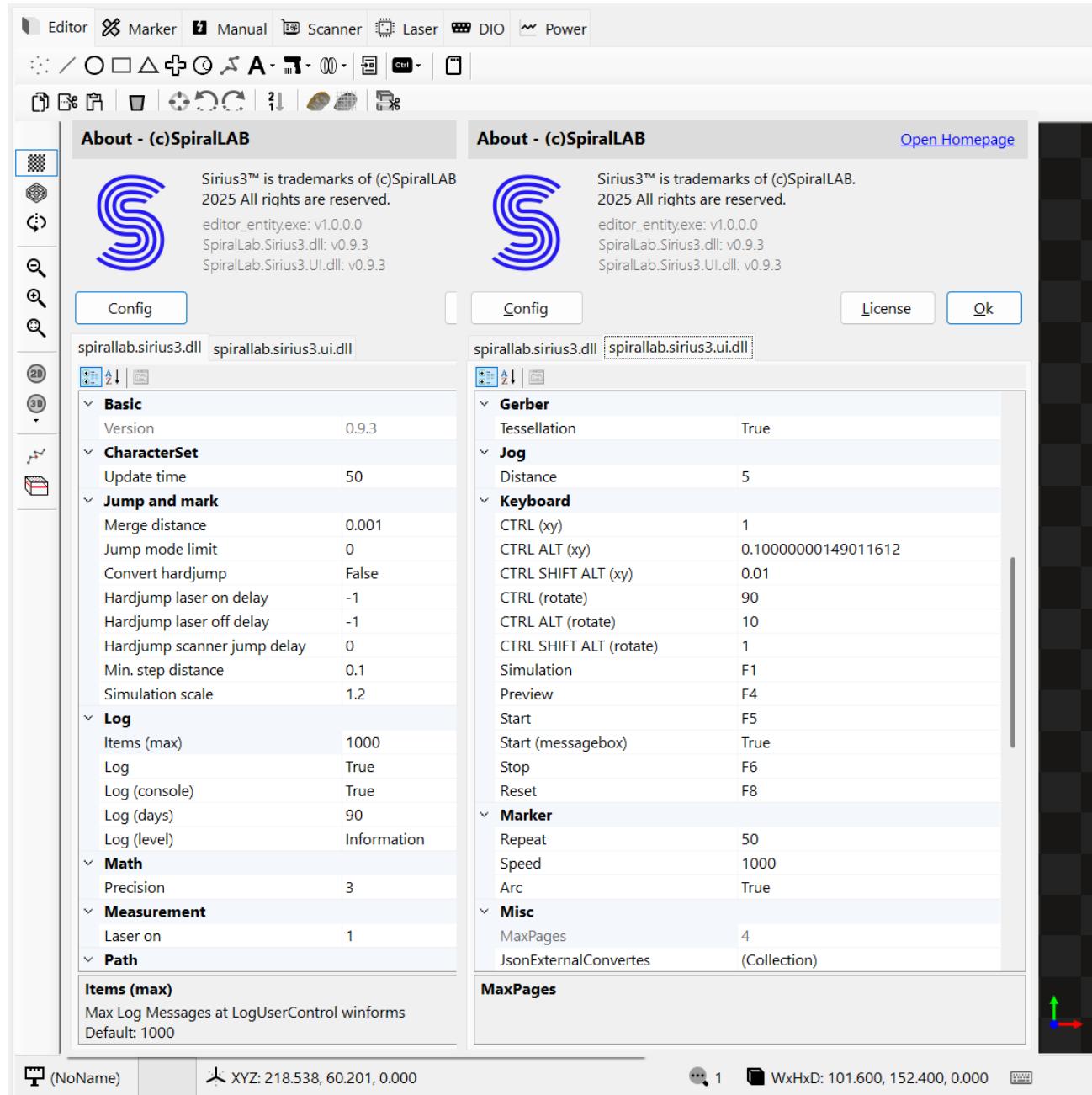
JobSyncAxis[] Array	
ID	ID: 1, Success, 10:41:28, 3.921s
Name	1
Description	
Bandwidth	2
Motion type	StageAndScanner
Config xml	D:\git\sirius3\lib\bin\net481\syncaxis\syncAXISCor
Relative coordinate system	False
Calculation	InProgress
Transfer	LoadedEnough
Execution	Finished
Result	Success
List counts	1986
Started	10:41:28.108
Ended	10:41:32.029
Execution	00:03.920
RTC	2.25773
Characteristic	
Scanner	
Position Max	3.33091130153022, 3.22174908542804
Velocity Max	107.461853397606, 100.533242138023
Acc Max	5793373.65365248, 7674742.9318269
Position Max + Laser On	3.33091130153022, 3.22174908542804
Min/Max X Position	-3.222 / 3.331
Min/Max Y Position	-3.218 / 3.222
Min/Max X Position + Laser On	-3.222 / 3.331
Min/Max Y Position + Laser On	-3.218 / 3.222
Stage	
Position Max	1.73491809911825, 2.29295437125394
Velocity Max	33.3771119446123, 36.4651926179738
Acc Max	689.723735902703, 703.748499475409
Jerk Max	50000.0145464696, 50000.031897035
Position Max + Laser On	1.69436003415024, 2.29295437125394
Min/Max X Position	-1.694 / 1.735
Min/Max Y Position	-1.567 / 2.293
Min/Max X Position + Laser On	-1.694 / 1.682
Min/Max Y Position + Laser On	-1.567 / 2.293
Motion MicroSteps	225773
Min/Max Mark Speed	98.581 / 100.000
Min	98.580899312304467
Max	100.00000000000001
Inserted Skywritings	0
Active channel count	0
Active channel	
Head count	1
Head1 offset	0.000, 0.000, 0.000, 0.000
Head2 offset	0.000, 0.000, 0.000, 0.000
Head3 offset	0.000, 0.000, 0.000, 0.000
Head4 offset	0.000, 0.000, 0.000, 0.000
Scanner	Pos: 12.337 %
Position	12.337, 11.932
X	12.337
Y	11.932
Stage count	1
Stage1	Pos: 1.529%, Acc: 7.037%, Jerk: 50.000%
Position	1.157, 1.529
X	1.157
Y	1.529

스캐너 헤드가 이동한 최대 좌표, 속도, 가속도 정보 및 동적 한계를 어느 정보 비율까지 사용했는지 등의 정보

스테이지가 이동한 최대 좌표, 속도, 가속도, 가가속도 정보 및 동적 한계를 어느 정보 비율까지 사용했는지 등의 정보

21. 환경 설정(Config)

라이브러리의 환경 설정값을 통해 다양한 기능을 처리할 수 있습니다. `spirallab.sirius3.dll` 모듈에서 제공되는 설정값은 `SpiralLab.Sirius3.Config` 으로 편집이 가능하며, `spirallab.sirius3.ui.dll` 모듈에서 제공되는 설정값은 `SpiralLab.Sirius3.UI.Config` 으로 편집이 가능합니다. 또한 편집기에서는 `About -> Config` 메뉴를 통해 시작화된 상태로 설정이 가능합니다. `Config` 설정값에 대한 자세한 설명은 별도의 개발자 매뉴얼을 참고해 주시기 바랍니다.



- 빈 페이지 -