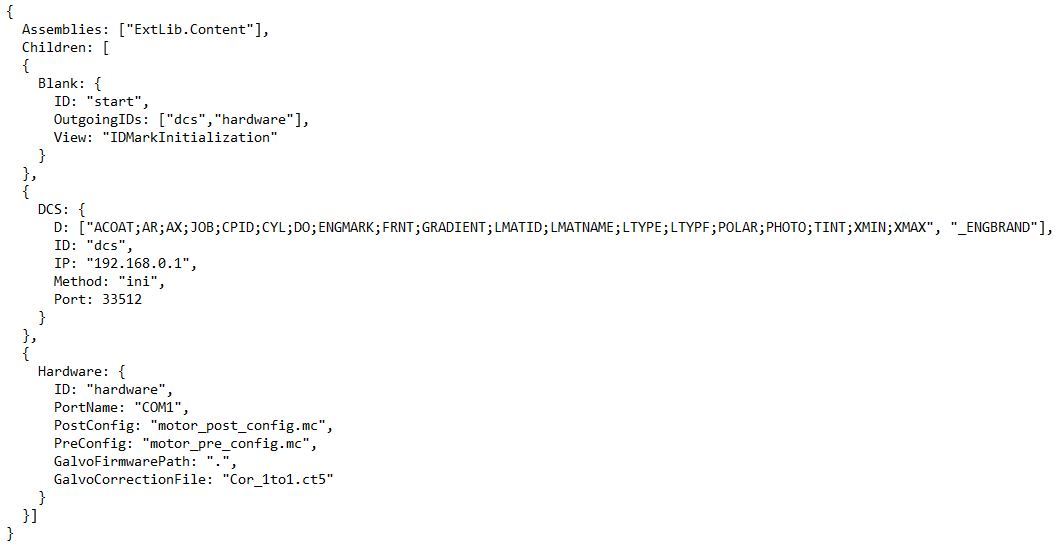
# PROCESS PLAYER

**What has this SDK been created for?** The answer is rather simple: to computerize equipment within a short time and using less power. A team of 2-5 employees spends a year or more for the things that can be done 2-3 times faster. How can you win in the competitive battle? The answer is that simple – to offer a better quality and give more, compared with what competitors can offer at the same price. Both, quality and that what you offer depend on the time you have for developing, testing, functionality; but time is money, not only yours but of your customers, too. Finally, you try to look for a compromise between time and quality. ProcessPlayer, while in search of this compromise, allows you to choose more quality, and this means you can do everything better than your competitors. So, it was just a short introduction.

**Main purpose:** simple, fast and intuitive description and process execution. The process of technological equipment, machine, physical model, or in general any process that can be described as a state machine.

**How does it work?** ProcessPlayer is an engine which executes a set of instructions in one or more script files. Each file is a script which contains the description of operation logic, either of a process as a whole or some part of it. This process script describes the elements, which represent the interface of some device, or just describes the logic of combining the elements with each other. Besides, it defines data transmission links between the elements, and sequences of their execution. For instance, process operation logic can be defined by the following elements, forming the first group: Any, Collector, CommandButton, Counter, Decision, and etc. Each of these elements specifies how a process state is going to change regarding the data they have received. The second group includes the elements the function of which is to organize an interface with devices or just a few libraries to work with files, databases, or doing certain calculations, and etc.

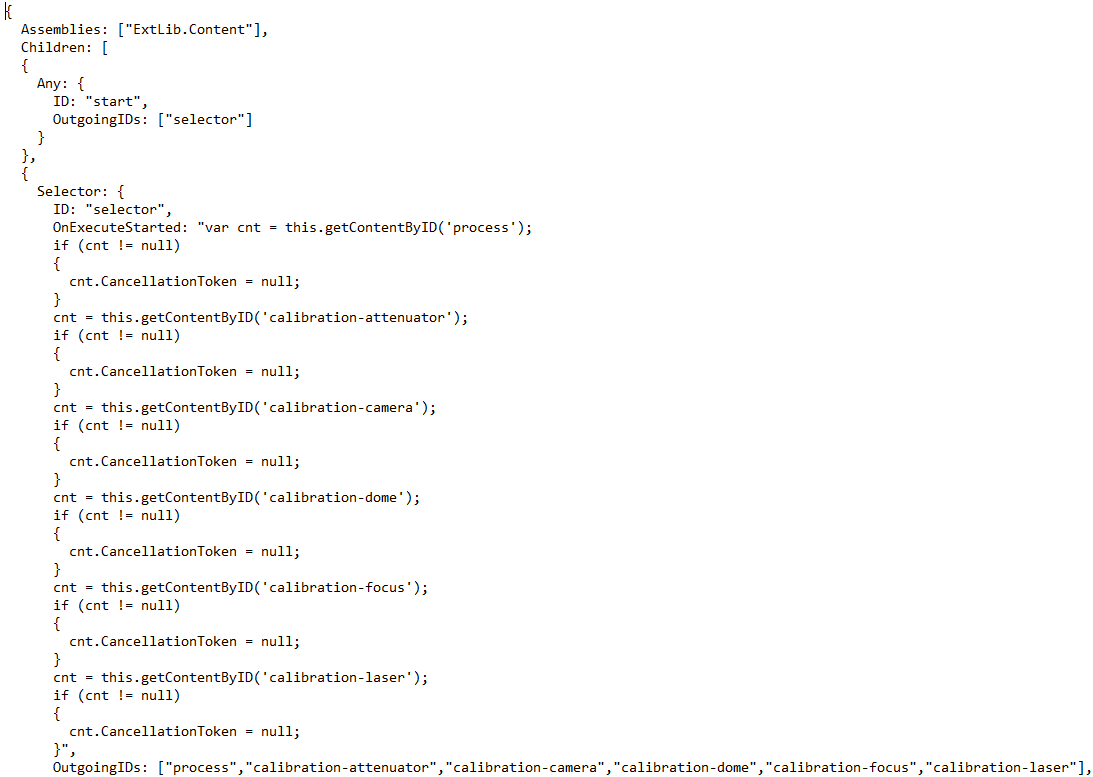


Each element from a script realizes an OO model, i.e. it is assigned with properties, methods and events, which can be defined in the script. If such element has a write property, its value can be defined in the script. Additionally, each element may have indicated a method, which will be used every time the element is being executed. The script, for instance, may have a number of elements with the same name, each of them having own method needed to realize the process.



Each element has the events defined, which indicate the element – its execution proper – started or finished, received data from one of ingoing elements, and etc. The realization or code describing the reaction on an event, is written using a language very similar to JScript, but with some exclusions. In the text of a procedure it is possible to use all methods and properties of elements available in the context.

Now, it is time we understand how all this work altogether. When started up, ProcessPlayer receives a command line listing all the scripts to be executed. Each script is parsed and pre-processed in the pre-processor for linking several scripts together, should there be #include instructions indicated in the script text. Then, the process script is serialized in an array of objects, the links and subordinations being defined between these objects. A prerequisite for any script is the presence of an element or elements not referred to by any other element, i.e. they are not present in the OutgoingsIDs array, with no one element of the script. There is one more mandatory condition – each element must have a unique identifier, regardless of the fact whether it is subordinated to another element or not. ID can be any string of chars, authorized for JSON. ProcessPlayer defines one or more elements having "open" inputs and calls for an ExecuteAsync method, specific to each of them, thus starting up the process. We should clarify that before you start up the process, it will be preceded by initialization and diagnostics, if defined for the elements. As one can see from the postfix in the ExecuteAsync name, this is an asynchronous method, and therefore all the elements will run asynchronously. Upon asynchronous execution, the element will transfer the data it has formed or which it received on its input, to all the elements it is linked with, i.e. to which it refers in its OutgoingsIDs list. As soon as any of the elements form the script receives input data, i.e. its input buffer content being modified, then, subject to TriggerMode set for this element, ExecuteAsync is called for, if the input buffer is full (TriggerMode = All mode), or if the input buffer has been changed (TriggerMode = Any mode). If such open-input element, for instance, refers to several elements, then following its execution it will transmit its results to each of them, and they start execution in parallel. The composition of elements in a script and connections between these elements may be arbitrary. Take for a example the list of elements forming the function logics of a script: Any, Blank, Collector, CommandButton, Copier, Counter, Decision, Group, Loop, Root, Selector, ToggleSwitch, UserContent, Wait. Service elements, which do not make decisions but serve in data transformation or in execution of certain procedures, are as follows: RdToRdCalculation, RdToRdFilter, RdToRdMapping, RdToRdSchema, Script, ToRdMSSQL, ToRdMYSQL, ToRdOracle. But these elements are not enough to realize the specifics, which is peculiar in any process. For instance, a device works with a barcode reader, so we need a Barcode element, which will wait for and receive data from a scanner or receive data put manually. To add the process realization with functionality, it is necessary to create own library, to which we'll add the classes inherited from the ProcessContent class. ProcessContent is a basic class for all elements of our script. For it is the class with the implemented methods in charge of initialization, diagnostics, and methods of an element. To implement Barcode, we need to add the Barcode class, inherit it from ProcessContent and redefine one or more ExecuteAsync, Initialize, Diagnostics methods. Examples of how to implement new classes can be found here. And, finally, one more thing to do with external libraries. Library must be described in the script for ProcessPlayer to see it. Root is the first element of a script, always, with its name hidden. This element has a property called Assemblies, in which we count all libraries, separating them with comas, to be used by ProcessPlayer. So, now we are a little more familiar with how ProcessPlayer is functioning.



Next point – ProcessPlayer communication with graphic interface. The simplest way to realize ProcessPlayer is a console application. As ProcessPlayer is written in С#, GUI can be WPF or Windows Forms. All you need to do is to have a set of scenes changing each other depending on the current process status. You will also need to link properties of graphic elements and properties of process elements. Each element of a process has the Views property, in which you indicate the names of scenes to be displayed when executing this element. GUI may be realized in a custom way. It may be a container, which will switch the context of scenes or it may be a static GUI, the behavior of which will be subject to the states of elements in a process, or may be combined. Due to ProcessPlayer being independent from GUI, it follows that the same process can be realized using multiple GUIs, each having different behavior and constraints. Some examples of GUI realization can be found here.

**Key advantages of using SDK:**

1. The whole process can be described abstractly, which significantly facilitates its creation, management, and understanding not only by the developer. **ProcessPlayer is a tool that allows the programmer to create and debug applications without much effort.**
2. The same library of elements representing the interfaces with devices or the library designed to work with files, databases, etc. can be used to describe different processes. **With more libraries gathered and interfaces realized in them, the process of developing any application goes faster**.
3. Due to the fact that ProcessPlayer is independent from GUI, any application will be developed in two stages. The first stage will include application logic development and debugging, and GUI on the next stage, both these stages being performed in parallel, if necessary. **The independence from GUI for process execution enables defining different GUIs for the same application, thus facilitating application prototyping**.
4. ProcessPlayer is just an open source SDK which can be debugged using standard Visual Studio environment. Moreover, the programmer can create various scripts for full or partial application debugging. **This is the same as testing Unit in .NET**.
5. Software product developers need not spend their time on figuring out how to embed methods in separate threads, or how to synchronize them, or how to set global context, and etc. All these things have been realized in ProcessPlayer. **All the developer have to do is the element that will be an interface with a device or functional class. In practice, this is mainly overriding of just two overloaded methods**.
6. ProcessPlayer usage is not limited only by creating applications for equipment automation. **Originally, it was designed for Web applications automation, and for applications that operate databases, that is why using it exclusively in equipment automation would be wrong**. I am convinced that the scope of application for ProcessPlayer will be only increasing.
7. Application itself can override scripts used by ProcessPlayer for application start-up, **so it means we can create an active intelligent system, a system with elements of artificial intelligence**.

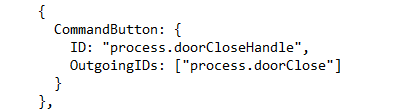
**Description of standard elements:**

**Any** element transmits data from its buffer to all elements it refers to every time when its buffer changes. I.e. if any element, which refers to Any, stops execution, it triggers Any element instantaneous execution.

**Blank** element does nothing but can be used to define the actual process state, other elements initialization or simply as trigger in the process scheme.

**CommandButton** element is used to interface the active script with GUI. For instance, if script elements' execution should depend on some event, then this element can be linked with a GUI button, and unless it is not pressed, this element will be in waiting mode. The element has one property:

* Command – RelayCommand type.

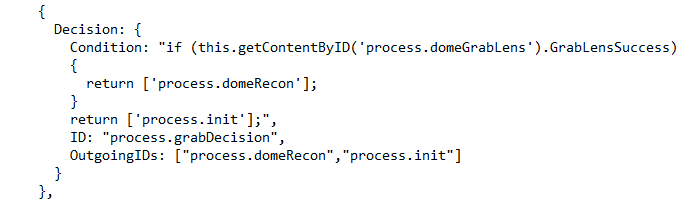


**Copier** element copies its buffer and, without modifying it, transmits its data to all element it refers to. The input buffer is a vocabulary, the key to which is an ID of the element sending data, and the value is an array of data. For any element, except Copier, the data from the output buffer is a vocabulary having just one element with the current ID. Often we need to keep data identified that is why using this element helps solve this task.

**Counter** element refers to two elements. If the value of the internal counter for element execution is less than the set value, data will be transferred to the first element of the list, or to the second in a row otherwise. This element is used as a counter of the number of executions per process cycle. The element has two properties:

* Count – actual value of the counter.
* Limit – maximum value of the counter.

**Decision** is a decision making element in a process scheme. This element has the Condition property, in which the decision procedure is defined, which element or elements Decision refers to must be then executed.



**Group** element contains a group of elements. As soon as the element is executed, it transmits data from its input buffer to all associated elements with open inputs. Group waits for data from all associated elements with open inputs. The element has one property:

* IsIsolated – specifies the behavior of a group when an exclusion happens inside this group. If IsIsolated = Тruе, then when an exclusion takes place, the parent process continues, and the execution is assigned to all elements the group refers to. If IsIsolated = False, then when an exclusion takes place, the parent process is interrupted.

**Loop** element starts up the elements it refers to so far as the condition returns true. The element has one property:

* Condition – in the property the procedure is set to be used by the element in defining whether or not the cycle must be continued.

**Root** is a service element to be used in a process just once and will be created automatically.

**Script** element contains a range of methods which help alter the script content. Element properties can be added, deleted or modified; you can also add or delete elements from script. Using this element makes it possible to modify attribute values after calibration, save and restore session states, and etc. The element has no properties.

**Selector** is a decision making element in a process scheme. Unlike the Decision element, in which the decision is made according to the Condition procedure, Selector receives commands containing parameters, with elements listed to be executed. The element has one property:

* Command – RelayCommand type.

**UserContent** is an element in which the results of work are calculated in a script itself. This element helps perform simple calculations or data processing in a script, thus giving the application more mobility. The element has one property:

* Perform – in the property the procedure is set to be used by the element in calculating the result.

**Wait** element makes a pause in a process. Using this element can be handy when it is necessary to slow down the process, with the aim to view the results of calculations in the script to control running times of elements, and etc. The element has one property:

* WaitTimeout – element hold-up expressed in milliseconds.

**Script listing:**

{

Group: {

Comment: "Main process group",

ID: "process",

IsIsolated: true,

OutgoingIDs: ["reset"],

Children: [

{

Any: {

ID: "process.start",

OnExecuteStarted: "var cnt = this.getContentByID('process.jobCounter');

if (cnt != null)

{

cnt.resetInputData();

}",

OutgoingIDs: ["process.init"]

}

},

{

Any: {

ID: "process.init",

OnExecuteStarted: "// initial reset for every process

var cnt = this.getContentByID('process.barcode');

if (cnt != null)

{

cnt.Code = '';

}

cnt = this.getContentByID('counterReloc');

if (cnt != null)

{

cnt.Count = 0;

}

cnt = this.getContentByID('process.rotation');

if (cnt != null)

{

cnt.resetInputData();

}

cnt = this.getContentByID('process.anglexy');

if (cnt != null)

{

cnt.resetInputData();

}

cnt = this.getContentByID('process.reconCounter');

if (cnt != null)

{

cnt.Count = 0;

}",

OutgoingIDs: ["process.doorOpen"],

Views: ["Blank"]

}

},

{

Door: {

ID: "process.doorOpen",

SetOpen: true,

OutgoingIDs: ["process.jobCounter"]

}

},

{

Decision: {

ID: "process.jobCounter",

IgnoreCalls: ["process.dcs"],

Condition: "var cnt = this.getInputData()['process.dcs'];

if (cnt == null || cnt.Length == 0)

{

globals['Lens'] = 0;

return ['process.barcode'];

}

var DO = '';

var packet = cnt[0].Data;

var count = packet.Count;

var i = 0;

while (i < count)

{

if (packet[i].Item1 == 'DO')

{

DO = trim(packet[i].Item2[0]);

break;

}

i += 1;

}

if (DO == 'B')

{

if (globals['Lens'] == 0)

{

globals['Lens'] = 1;

return ['process.recognition'];

}

else

{

globals['Lens'] = 0;

return ['process.barcode'];

}

}

return ['process.barcode'];",

OutgoingIDs: ["process.barcode","process.recognition"]

}

},

{

Barcode: {

ConfigGroup: 1,

ConfigNumber: 1,

EnterManualy: true,

ID: "process.barcode",

IP: "192.168.0.79",

OutgoingIDs: ["process.dcs"],

Port: 50003,

Views: ["ProcessBarcode"]

}

},

{

DCS: {

ID: "process.dcs",

IP: "192.168.0.1",

Method: "exec",

OnExecuteStarted: "var cnt = this.getInputData()['process.barcode'];

if (cnt != null && cnt.Length > 0)

{

this.Job=toInt(cnt[0].Data);

}",

OutgoingIDs: ["process.recognition","process.engrController","process.engrave","process.jobCounter","process.reconAcq"],

Port: 33512,

Views: ["ProcessDCS"]

}

},

{

Recognition: {

Condition: "return ['process.doorCloseHandler'];",

ID: "process.recognition",

OutgoingIDs: ["process.doorCloseHandler","endProcess"],

TriggerMode: 1,

Views: ["ProcessDCS"]

}

},

{

CommandButton: {

ID: "process.doorCloseHandle",

OutgoingIDs: ["process.doorClose"]

}

},

{

Door: {

ID: "process.doorClose",

SetOpen: false,

OutgoingIDs: ["process.trampWork"],

Views: ["ProcessLoading"]

}

},

{

Trampolin: {

ID: "process.trampWork",

SetClamp: true,

SetWork: true,

OutgoingIDs: ["process.trampRelease"]

}

},

{

Trampolin: {

ID: "process.trampRelease",

SetClamp: false,

OutgoingIDs: ["process.domeGrabLens"]

}

},

{

Dome: {

ID: "process.domeGrabLens",

Method: "grablens",

OutgoingIDs: ["process.trampClear"]

}

},

{

Trampolin: {

ID: "process.trampClear",

SetClamp: true,

SetWork: false,

OutgoingIDs: ["process.trampClearRelease"]

}

},

{

Trampolin: {

ID: "process.trampClearRelease",

SetClamp: false,

OutgoingIDs: ["process.grabDecision"]

}

},

{

Decision: {

Condition: "if (this.getContentByID('process.domeGrabLens').GrabLensSuccess)

{

return ['process.domeRecon'];

}

return ['process.init'];",

ID: "process.grabDecision",

OutgoingIDs: ["process.domeRecon","process.init"]

}

},

{

Dome: {

ID: "process.domeRecon",

Method: "zmove",

OutgoingIDs: ["process.mirrorShow"],

ZRel: 10

}

},

{

ReconMirror: {

ID: "process.mirrorShow",

SetActive: true,

OutgoingIDs: ["process.reconMotor"]

}

},

{

Recon: {

AutoResetOutputs: true,

ID: "process.reconMotor",

Method: "motor",

OutgoingIDs: ["process.reconLED"],

ReconMotorFreqMin: 700,

ReconIP: "192.168.0.80",

ReconPort: 50020

}

},

{

Recon: {

AutoResetOutputs: true,

CameraINI: "camera.ini",

CameraResX: 1680,

CameraResY: 1680,

ID: "process.reconLED",

Method: "led",

OutgoingIDs: ["process.reconCounter"],

ReconLensBackgroundOptimal: 128,

ReconLEDMax: 100

}

},

{

Counter: {

AutoResetOutputs: true,

ID: "process.reconCounter",

Limit: 5,

OutgoingIDs: ["process.reconAcq","process.reconError"],

TriggerMode: 1

}

},

{

Recon: {

AttemptCount: 1,

DomeRadius: 165,

DomeOffsetX: 17.0608332552163,

DomeOffsetY: 10.1198836288012,

EngrAreaOffsetMM: 30,

EngrAreaRadiusMM: 17,

EngrOffsetMM: 17,

LogoDiameter: 42,

LogosDistance: 488,

LogosDistanceMM: 34,

ID: "process.reconAcq",

Method: "recognize",

OnExecuteStarted: "this.Image = null;",

OutgoingIDs: ["process.engrave","process.rotation","process.anglexy","process.viewRecon"],

ReconRatio: 14.352941176470588235294117647059

}

},

{

Blank: {

AutoResetOutputs: true,

ID: "process.viewRecon",

OutgoingIDs: ["process.engrController"],

Views: ["ProcessRecon"]

}

},

{

EngrController: {

AutoResetOutputs: true,

ID: "process.engrController",

OnExecuteStarted: "var input = this.getInputData();

this.DCSData = input['process.dcs'];

this.Image = null;

this.ReconData = input['process.viewRecon'];",

OutgoingIDs: ["process.anyEngrave","process.viewEngrController"]

}

},

{

Blank: {

AutoResetOutputs: true,

ID: "process.viewEngrController",

OutgoingIDs: ["process.anglexy"],

Views: ["ProcessEngrController"]

}

},

{

PositionController: {

ID: "process.anglexy",

Method: "anglexy",

OnExecuteStarted: "var input = this.getInputData();

this.ReconData = input['process.reconAcq'];",

OutgoingIDs: ["process.rotation","process.rotateAndRecognize","process.reposition","process.mirrorHide"]

}

},

{

Any: {

AutoResetOutputs: true,

ID: "process.anyEngrave",

OutgoingIDs: ["process.engrave"]

}

},

{

Group: {

Comment: "Engrave group",

ID: "process.engrave",

Children: [

{

Copier: {

ID: "process.engrave.start",

OutgoingIDs: ["process.engrave.anglexz","process.engrave.check","process.engrave.reconCapture","process.engrave.shooter"],

Views: ["ProcessShooterPreload"]

}

},

{

Recon: {

Comment: "temporary element for capture image",

ID: "process.engrave.reconCapture",

Method: "capture",

OnExecuteStarted: "this.Image = null;",

OutgoingIDs: ["process.engrave.check","process.engrave.mirrorHide","process.engrave.shooter"]

}

},

{

ReconMirror: {

AutoResetOutputs: true,

ID: "process.engrave.mirrorHide",

SetActive: false,

OutgoingIDs: ["process.engrave.anglexz"]

}

},

{

PositionController: {

DomeR: 10,

Domer: 2,

ID: "process.engrave.anglexz",

Method: "anglexz",

OnExecuteStarted: "var input = this.getInputData();

this.DCSData = input['process.dcs'];

this.ReconData = input['process.reconAcq'];",

OutgoingIDs: ["process.engrave.domeAlign","process.engrave.domeRestore"]

}

},

{

Dome: {

AutoResetOutputs: true,

ID: "process.engrave.domeAlign",

Method: "position",

OnExecuteStarted: "var input = this.getInputData();

var dict = input['process.engrave.anglexz'][0].Data;

this.RotateAngle = dict['angle'];

this.ZRel = 10 + dict['dz'];",

OutgoingIDs: ["process.engrave.shooter"]

}

},

{

Shooter: {

FontDir: "D:\\LSU193",

ID: "process.engrave.shooter",

LaserX0: 300,

LaserY0: 0,

OnExecuteStarted: "var input = this.getInputData();

this.Bmp = input['process.engrave.reconCapture'][0].Data['bmp'];

this.DCSData = input['process.dcs'];

this.ReconData = input['process.reconAcq'];",

OutgoingIDs: ["process.engrave.check","process.engrave.viewEngraving"]

}

},

{

Blank: {

AutoResetOutputs: true,

ID: "process.engrave.viewEngraving",

OutgoingIDs: ["process.engrave.domeRestore"],

Views: ["ProcessShooter"]

}

},

{

Dome: {

ID: "process.engrave.domeRestore",

Method: "position",

OutgoingIDs: ["process.engrave.mirrorShow"],

ZRel: 10

}

},

{

ReconMirror: {

AutoResetOutputs: true,

ID: "process.engrave.mirrorShow",

SetActive: true,

OutgoingIDs: ["process.engrave.check"]

}

},

{

Recon: {

ID: "process.engrave.check",

Method: "check",

OnExecuteStarted: "var input = this.getInputData();

this.DCSData = input['process.dcs'];

this.ReconData = input['process.reconAcq'];

this.ShooterData = input['process.engrave.shooter'];"

}

}],

OutgoingIDs: ["process.mirrorHide"]

}

},

{

Group: {

Comment: "Rotation lens group.",

Children: [

{

Copier: {

ID: "process.rotation.start",

OutgoingIDs: ["process.rotation.domeRotate","process.rotation.engrDisplay"]

}

},

{

Dome: {

ID: "process.rotation.domeRotate",

Method: "rotate",

OnExecuteStarted: "this.RotateAngle = this.getInputData()['process.anglexy'][0].Data['angle'];",

OutgoingIDs: ["process.rotation.reconCapture"]

}

},

{

Recon: {

AutoResetOutputs: true,

ID: "process.rotation.reconCapture",

Method: "capture",

OnExecuteStarted: "this.Image = null;",

OutgoingIDs: ["process.rotation.engrDisplay"]

}

},

{

EngrDisplay: {

ID: "process.rotation.engrDisplay",

OnExecuteStarted: "var input = this.getInputData();

this.BitmapData = input['process.rotation.reconCapture'];

this.Image = null;

this.ReconData = input['process.reconAcq'];"

}

}],

ID: "process.rotation",

OnExecuteStarted: "var cnt = this.getChildrenByID('process.rotation.engrDisplay');

if (cnt != null)

{

cnt.BitmapData = null;

cnt.Image = null;

}

cnt = this.getChildrenByID('process.rotation.domeRotate');

if (cnt != null)

{

cnt.RotateAngle = 0;

}",

OutgoingIDs: ["process.anyEngrave"],

Views: ["ProcessRotationAndCapture"]

}

},

{

Group: {

Comment: "Group with just lens rotation. After execution it jumped to process.recognition.",

Children: [

{

Dome: {

ID: "process.rotateAndRecognize.domeRotate",

Method: "rotate",

OnExecuteStarted: "this.RotateAngle = this.getInputData()['process.anglexy'][0].Data['angle'];"

}

}],

ID: "process.rotateAndRecognize",

OnExecuteStarted: "var cnt = this.getChildrenByID('process.rotateAndRecognize.domeRotate');

if (cnt != null)

{

cnt.RotateAngle = 0;

}",

OutgoingIDs: ["process.reconCounter"],

TriggerMode: 1,

Views: ["ProcessRotation"]

}

},

{

Group: {

Comment: "Group with lens rotation and shifting. After execution it jumped to process.recognition.",

Children: [

{

Copier: {

ID: "process.reposition.start",

OutgoingIDs: ["process.reposition.domeRotate","process.reposition.trampLoad"]

}

},

{

Dome: {

ID: "process.reposition.domeRotate",

Method: "rotate",

OnExecuteStarted: "this.RotateAngle = this.getInputData()['process.anglexy'][0].Data['angle'];",

OutgoingIDs: ["process.reposition.mirrorHide"]

}

},

{

ReconMirror: {

ID: "process.reposition.mirrorHide",

SetActive: false,

OutgoingIDs: ["process.reposition.counter"]

}

},

{

Counter: {

ID: "process.reposition.counter",

Limit: 2,

OutgoingIDs: ["process.reposition.domeSafePos","process.reposition.end"],

TriggerMode: 1

}

},

{

Dome: {

AutoResetOutputs: true,

ID: "process.reposition.domeSafePos",

Method: "zmove",

OutgoingIDs: ["process.reposition.trampLoad"],

ZRel: 45

}

},

{

Trampolin: {

ID: "process.reposition.trampLoad",

SetClamp: true,

SetWork: true,

OnExecuteStarted: "this.WorkOffset = this.getInputData()['process.anglexy'][0].Data['dx'];",

OutgoingIDs: ["process.reposition.domeDropLens"]

}

},

{

Dome: {

ID: "process.reposition.domeDropLens",

Method: "droplens",

OutgoingIDs: ["process.reposition.trampShift"]

}

},

{

Trampolin: {

ID: "process.reposition.trampShift",

SetClamp: true,

SetWork: true,

OutgoingIDs: ["process.reposition.trampShiftRelease"]

}

},

{

Trampolin: {

ID: "process.reposition.trampShiftRelease",

SetClamp: false,

OutgoingIDs: ["process.reposition.domeGrabLens"]

}

},

{

Dome: {

ID: "process.reposition.domeGrabLens",

Method: "grablens",

OutgoingIDs: ["process.reposition.trampClear"]

}

},

{

Trampolin: {

ID: "process.reposition.trampClear",

SetClamp: true,

SetWork: false,

OutgoingIDs: ["process.reposition.trampClearRelease"]

}

},

{

Trampolin: {

ID: "process.reposition.trampClearRelease",

SetClamp: false,

OutgoingIDs: ["process.reposition.grabDecision"]

}

},

{

Decision: {

Condition: "if (this.getContentByID('process.reposition.domeGrabLens').GrabLensSuccess)

{

return ['process.reposition.domeRecon'];

}

var cnt = this.getContentByID('process.reposition.trampLoad');

if (cnt != null)

{

var dx = cnt.getInputData()['process.anglexy'][0].Data['dx'];

cnt.getInputData()['process.anglexy'][0].Data['dx'] = -dx;

}

return ['process.reposition.counter'];",

ID: "process.reposition.grabDecision",

OutgoingIDs: ["process.reposition.domeRecon","process.reposition.counter"]

}

},

{

Dome: {

ID: "process.reposition.domeRecon",

Method: "zmove",

OutgoingIDs: ["process.reposition.mirrorShow"],

ZRel: 10

}

},

{

ReconMirror: {

ID: "process.reposition.mirrorShow",

OutgoingIDs: ["process.reposition.end"],

SetActive: true

}

},

{

Any: {

ID: "process.reposition.end"

}

}],

ID: "process.reposition",

OnExecuteStarted: "var cnt = this.getChildrenByID('process.reposition.counter');

if (cnt != null)

{

cnt.Count = 0;

}

cnt = this.getChildrenByID('process.reposition.domeRotate');

if (cnt != null)

{

cnt.RotateAngle = 0;

}

cnt = this.getChildrenByID('process.reposition.trampLoad');

if (cnt != null)

{

cnt.WorkOffset = 0;

}",

OutgoingIDs: ["process.grabDecision2"],

Views: ["ProcessReposition"]

}

},

{

Decision: {

Condition: "if (this.getContentByID('process.reposition.domeGrabLens').GrabLensSuccess)

{

var cnt = this.getContentByID('process.reposition.counter');

if (cnt != null && cnt.Count >= 2)

{

var cnt = this.getContentByID('process.rotateAndRecognize.domeRotate');

if (cnt != null)

{

cnt.getInputData()['process.anglexy'][0].Data['angle'] = 90;

return ['process.rotateAndRecognize'];

}

}

return ['process.reconCounter'];

}

return ['process.init'];",

ID: "process.grabDecision2",

OutgoingIDs: ["process.reconCounter","process.rotateAndRecognize","process.init"]

}

},

{

CommandButton: {

ID: "process.reconError",

OnExecuteStarted: "this.Status = 'Engravings on the lens were not found or their quality to low. Engrave process stopped.';",

OutgoingIDs: ["process.mirrorHide"],

Views: ["ProcessError"]

}

},

{

ReconMirror: {

ID: "process.mirrorHide",

SetActive: false,

OutgoingIDs: ["process.domeDropLens"],

TriggerMode: 1,

Views: ["ProcessUnloading"]

}

},

{

Dome: {

ID: "process.domeDropLens",

Method: "droplens",

OutgoingIDs: ["process.trampLoad"]

}

},

{

Trampolin: {

ID: "process.trampLoad",

SetClamp: true,

SetWork: false,

OutgoingIDs: ["process.trampLoadRelease"]

}

},

{

Trampolin: {

ID: "process.trampLoadRelease",

SetClamp: false,

OutgoingIDs: ["process.init"]

}

},

{

ToggleSwitch: {

ID: "process.deadlock"

}

}]

}

}