VDMA 40301

ICS 35.240.50; 81.100

OPC UA for Flat Glass

OPC UA for Flat Glass

VDMA 40301:2022-03 is identical with OPC 40301 (Release 1.0)

Document comprises 65 pages

VDMA

Contents

Figures

Tables

Forewords

This specification was created by a joint working group of the OPC Foundation and VDMA. It is adopted identically as VDMA Specification.

VDMA Forum Glass Technology

The VDMA represents around 3300 German and European companies in the mechanical engineering industry. The industry represents innovation, export orientation, medium-sized companies and employs around four million people in Europe, more than one million of them in Germany. The Forum Glass Technology is a network of machine and plant manufacturers along the process chain of the glass industry. It supervises technical working groups, carries out standardization, is active for its members in international markets and represents the interests of companies in the glass industry.

OPC Foundation

OPC is the interoperability standard for the secure and reliable exchange of data and information in the industrial automation space and in other industries. It is platform independent and ensures the seamless flow of information among devices from multiple vendors. The OPC Foundation is responsible for the development and maintenance of this standard.

OPC UA is a platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework. This multi-layered approach accomplishes the original design specification goals of:

– Platform independence: from an embedded microcontroller to cloud-based infrastructure

– Secure: encryption, authentication, authorization and auditing

– Extensible: ability to add new features including transports without affecting existing applications

– Comprehensive information modelling capabilities: for defining any model from simple to complex

1 Scope

The OPC 40301 - 40329 specifications contain OPC UA Companion Specifications of several glass industry sectors and are developed by members of VDMA and/or the OPC Foundation. OPC UA is a machine to machine communication technology to transmit characteristics of products (e.g. manufacturer name, device type or components) and process data (e.g. temperatures, pressures or feed rates). To enable vendor unspecific interoperability the description of product characteristics and process data has to be standardized utilizing technical specifications, the OPC UA companion specifications.Glass industry machinery has a broad range of special application from bottles and tableware to flat glass products for buildings, for window glasses and technology applications such as photovoltaic elements, automotive or display glass (see also chapter 5).

Thus, glass producers as processors are required besides a variety of machinery, a considerable number of specialized manufacturers of such machines. Integrating these machines into an effective production ecosystem requires a considerable amount of interfacing between the machines and equally to production planning and MES software.

Parallel to this challenge the requirements of transparent data transfer between automation layer are a basic demand of the RAMI 4.0 model. In order to allow machinery builders for glass fabricants and processors to evolve into industry 4.0 and simplify integration, a standardization of communication is required. OPC UA is considered to be one of the main languages for standardized communication, however, the variety of possibilities requires confinement to the requirements of the glass industry. A Companion Specification defining information models and exchange methods for the glass industry is an appropriate response to those requirements.

OPC 40301 describes the interface between a flat glass processing/assembling machine and manufacturing execution systems (MES), or enterprise resource planning (ERP) for data exchange.

The target of OPC 40301 is to provide a unique interface for flat glass processing machines, e.g., cutting machines and higher order systems from different manufacturers to ensure compatibility.

The following functionalities are covered:

- Machine identification, including machine status, information for job planning and machine capability, information on logged-in users;
- Job handling including job management, job status, calls to deal with jobs

2 Normative references

OPC 10000-1, *OPC Unified Architecture - Part 1: Overview and Concepts*

http://www.opcfoundation.org/UA/Part1/

OPC 10000-2, *OPC Unified Architecture - Part 2: Security Model*

http://www.opcfoundation.org/UA/Part2/

OPC 10000-3, *OPC Unified Architecture - Part 3: Address Space Model*

http://www.opcfoundation.org/UA/Part3/

OPC 10000-4, *OPC Unified Architecture - Part 4: Services*

http://www.opcfoundation.org/UA/Part4/

OPC 10000-5, *OPC Unified Architecture - Part 5: Information Model*

http://www.opcfoundation.org/UA/Part5/

OPC 10000-6, *OPC Unified Architecture - Part 6: Mappings*

http://www.opcfoundation.org/UA/Part6/

OPC 10000-7, *OPC Unified Architecture - Part 7: Profiles*

http://www.opcfoundation.org/UA/Part7/

OPC 10000-100, *OPC Unified Architecture - Part 100: Devices*

http://www.opcfoundation.org/UA/Part100/

OPC 40001-1, *OPC UA for Machinery - Part 1: Basic Building Blocks*

http://www.opcfoundation.org/UA/Machinery/

3 Terms, definitions and conventions

3.1 Overview

It is assumed, that basic concepts of OPC UA information modelling and OPC UA for Machinery are understood in this specification. It will use these concepts to describe the glass technology Information Model. For the purposes of this document, the terms and definitions given in OPC 10000-1, OPC 10000-3, OPC 10000-4, OPC 10000-5, OPC 10000-7, OPC 10000-100 as well as the following apply.

Note that OPC UA terms and terms defined in this specification are *italicized* in the specification.

3.2 OPC UA for glass technology terms terms

3.2.1 Coating

Layer structure usually applied onto the glass surface by a CVD or PVD process to influence the spectral properties of the component such as transmission and reflection. Such coating is essential for energy conserving glazing where IR radiation may be controlled by this layer. There are several coating classes which are distinguished as follows:

- Hard Coated (HC): This describes a type of coating or covering a glass surface in a way that the resulting surface is rather resistant against damage, at least at a similar level as compared to standard Floatglass. In a composite, the surface may be exposed to the environment.
- Soft Coated (SC): This describes a type of coating covering a glass surface resulting in a surface that is more vulnerable by environmental conditions such as moist, dirt etc. Soft coated glass panes require more care in processing.
- Coated with foil protection (FC): This describes glass panes where the coating (typically a soft coating) is protected against environmental influence by a foil that might have to be removed when the glass is to be processed.

3.2.2 Job

A job is defined as the concrete implementation of one or more programs or recipes by using a given order to produce something.

3.2.3 Job Group

A set of jobs.

3.2.4 Jumbo sheets, Raw glass panes

Large format glass panes as produced from float tanks at flat glass production plant. The size is typically 6000*3210 mm.

3.2.5 Structure Alignment

Specifies how the surface structure is aligned.

3.2.6 Significant Side

The term “Significant Side” is used if there is a necessity to distinguish which side of a glass pane is “up” or “down” or “front” or “back” while processing the glass pane. Typical examples are:

- When producing flat glass, one side is floating on the tin bath, the other is exposed to the gas fire heating the chamber.
- If the glass is coated (a thin metallic layer is applied), it is essential to know which side is coated
- Glass having patterns or other surface treatments

3.2.7 User Profile

A User Profile contains the meta data of a logged in user.

3.3 OPC UA for glass technology terms (Insulating Glass Units)

3.3.1 Cavity

Space between glass panes, width depends on spacer. Filled with a gas or gas mixture to provide demanded thermal properties.

3.3.2 Gas Filling

The gas filling refers to gas in the space between glass of an Insulating Glass Unit IGU. Typically, the gas used is Argon or Xenon, both gases which allow good thermal resistance.

3.3.3 Perimeter Protection

Used to protect the second sealant of an Insulating Glass Unit. Typically, an aluminum tape is used to wrap the edge of the unit.

3.3.4 Primary Sealing

Applied to the spacer to prevent ingress of moisture and loss of gases between spacer and glass.

Note: See also Secondary Sealing

3.3.5 Sealant Depth

The sealant depth is the minimum dimension from the spacer to the outer edge of the silicone secondary seal. This dimension may also be referred to as the “bite” or “height” of the insulating glass sealant.

3.3.6 Secondary Sealing

Applied to IGU after assembling. Provides resistance against moisture and loss of gas. Provides additional structural strength. Possible materials: polyurethane, polysulfide, butyl, silicone, polyisobutylene

Note: See also Primary Sealing

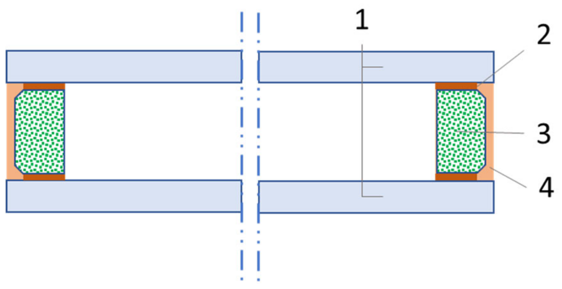
3.3.7 Spacer

A spacer is used in IGU’s to physically separate the individual glass panes. Spacer can be of different material such as for example:

- Metal spacers out of aluminum or stainless steel
- Carbon based spacers either rigid or flexible
- TPS (Thermoplastic spacer): a gel like liquid that is formed to spacer size when applied to the glass.
- Other (e.g. elastic spacer)

3.3.8 Insulating Glass Unit (IGU)

Unit of 2 or more panes of glass. Gas filling is used to set thermal transfer properties. Spacer defines the distance between the panes and also accounts for thermal properties. Sealants prevent loss of gas filling and entrance of humidity. Figure 1 shows a sectional drawing of an IGU with the different elements.



Key:

1 glass panes

2 primary sealing

3 spacers with desiccant

4 secondary sealing

Figure 1 – Sectional drawing of an IGU

3.3.9 IGU Line

Production line for IGU´s.

3.3.10 Laminated Glass

Type of glass according to prEN12543-1:2020, e.g., glass that is made of two or more panes of glass joined together by a layer of plastic, or polyvinyl butyral (PVB).

3.4 Abbreviated terms

AC	Alarm and Condition
CSV	Character-separated values
CVD	Chemical vapor deposition
DCS	Distributed Control Systems
DI	Device integration
ERP	Enterprise resource planning
FC	Foil coated
HC	Hard coated
HLS	Higher level system
ID	Identifier
IG	Insulating glass
IGU	Insulating glass unit
M	Mandatory
MES	Manufacturing execution system
O	Optional
OPC	Open platform communications
PVB	Polyvinyl butyral
PVD	Physical vapor deposition
RAMI 4.0	Reference architecture model industry 4.0
SC	Soft coated
SI	International system of units
TPS	Thermoplastic spacer
UA	Unified architecture
UD	User defined
URI	Uniform Resource Identifier
URL	Uniform resource locator
UTC	Coordinated universal time

3.5 Conventions used in this document

3.5.1 Conventions for Node descriptions

3.5.1.1 Node definitions

Node definitions are specified using tables (see Table 2).

Attributes are defined by providing the *Attribute* name and a value, or a description of the value.

References are defined by providing the *ReferenceType* name, the *BrowseName* of the *TargetNode* and its *NodeClass*.

– If the *TargetNode* is a component of the *Node* being defined in the table the *Attributes* of the composed *Node* are defined in the same row of the table.

– The *DataType* is only specified for *Variables*; “[<number>]” indicates a single-dimensional array, for multi-dimensional arrays the expression is repeated for each dimension (e.g. [2][3] for a two-dimensional array). For all arrays the *ArrayDimensions* is set as identified by <number> values. If no <number> is set, the corresponding dimension is set to 0, indicating an unknown size. If no number is provided at all the *ArrayDimensions* can be omitted. If no brackets are provided, it identifies a scalar *DataType* and the *ValueRank* is set to the corresponding value (see OPC 10000-3). In addition, *ArrayDimensions* is set to null or is omitted. If it can be Any or *ScalarOrOneDimension*, the value is put into “{<value>}”, so either “{Any}” or “{*ScalarOrOneDimension*}” and the *ValueRank* is set to the corresponding value (see OPC 10000-3) and the *ArrayDimensions* is set to null or is omitted. Examples are given in Table 1.

Table 1 – Examples of DataTypes

Notation	Data-Type	Value-Rank	Array-Dimensions	Description
0:Int32	0:Int32	-1	omitted or null	A scalar Int32.
0:Int32[]	0:Int32	1	omitted or {0}	Single-dimensional array of Int32 with an unknown size.
0:Int32[][]	0:Int32	2	omitted or {0,0}	Two-dimensional array of Int32 with unknown sizes for both dimensions.
0:Int32[3][]	0:Int32	2	{3,0}	Two-dimensional array of Int32 with a size of 3 for the first dimension and an unknown size for the second dimension.
0:Int32[5][3]	0:Int32	2	{5,3}	Two-dimensional array of Int32 with a size of 5 for the first dimension and a size of 3 for the second dimension.
0:Int32{Any}	0:Int32	-2	omitted or null	An Int32 where it is unknown if it is scalar or array with any number of dimensions.
0:Int32{ScalarOrOneDimension}	0:Int32	-3	omitted or null	An Int32 where it is either a single-dimensional array or a scalar.

– The TypeDefinition is specified for *Objects* and *Variables*.

– The TypeDefinition column specifies a symbolic name for a *NodeId*, i.e. the specified *Node* points with a *HasTypeDefinition Reference* to the corresponding *Node*.

– The *ModellingRule* of the referenced component is provided by specifying the symbolic name of the rule in the *ModellingRule* column. In the *AddressSpace*, the *Node* shall use a *HasModellingRule Reference* to point to the corresponding *ModellingRule Object*.

If the *NodeId* of a *DataType* is provided, the symbolic name of the *Node* representing the *DataType* shall be used.

Note that if a symbolic name of a different namespace is used, it is prefixed by the *NamespaceIndex* (see 3.5.2.2).

Nodes of all other *NodeClasses* cannot be defined in the same table; therefore only the used *ReferenceType*, their *NodeClass* and their *BrowseName* are specified. A reference to another part of this document points to their definition.

Table 2 illustrates the table. If no components are provided, the DataType, TypeDefinition and ModellingRule columns may be omitted and only a Comment column is introduced to point to the *Node* definition.

Table 2 – Type Definition Table

Attribute	Value				
Attribute name	Attribute value. If it is an optional Attribute that is not set "--" will be used.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
ReferenceType name	*NodeClass* of the target *Node*.	*BrowseName* of the target *Node*.	*DataType* of the referenced *Node*, only applicable for *Variables*.	*TypeDefinition* of the referenced *Node*, only applicable for *Variables* and *Objects*.	Additional characteristics of the *TargetNode* such as the *ModellingRule* or *AccessLevel*.
NOTE Notes referencing footnotes of the table content.					

Components of *Nodes* can be complex that is containing components by themselves. The *TypeDefinition*, *NodeClass* and *DataType* can be derived from the type definitions, and the symbolic name can be created as defined in 3.5.3.1. Therefore, those containing components are not explicitly specified; they are implicitly specified by the type definitions.

The Other column defines additional characteristics of the Node. Examples of characteristics that can appear in this column are show in Table 3.

Table 3 – Examples of Other Characteristics

Name	Short Name	Description
0:Mandatory	M	The Node has the Mandatory ModellingRule.
0:Optional	O	The Node has the Optional ModellingRule.
0:MandatoryPlaceholder	MP	The Node has the MandatoryPlaceholder ModellingRule.
0:OptionalPlaceholder	OP	The Node has the OptionalPlaceholder ModellingRule.
ReadOnly	RO	The *Node AccessLevel* has the *CurrentRead* bit set but not the *CurrentWrite* bit.
ReadWrite	RW	The Node AccessLevel has the CurrentRead and CurrentWrite bits set.
WriteOnly	WO	The Node AccessLevel has the *CurrentWrite* bit set but not the *CurrentRead* bit.

If multiple characteristics are defined they are separated by commas. The name or the short name may be used.

3.5.1.2 Additional References

To provide information about additional *References*, the format as shown in Table 4 is used.

The components of the *ObjectType* have additional references which are defined in Table 4.

Table 4 – <some> Additional References

SourceBrowsePath	Reference Type	Is Forward	TargetBrowsePath
SourceBrowsePath is always relative to the *TypeDefinition*. Multiple elements are defined as separate rows of a nested table.	*ReferenceType* name	True = forward *Reference*	TargetBrowsePath points to another *Node*, which can be a well-known instance or a *TypeDefinition*. You can use *BrowsePaths* here as well, which is either relative to the *TypeDefinition* or absolute. If absolute, the first entry needs to refer to a type or well-known instance, uniquely identified within a namespace by the *BrowseName*.

References can be to any other *Node*.

3.5.1.3 Additional sub-components

To provide information about sub-components, the format as shown in Table 5 is used.

Table 5 – <some>Type Additional Subcomponents

BrowsePath	Reference	NodeClass	BrowseName	DataType	TypeDefinition	Others
BrowsePath is always relative to the *TypeDefinition*. Multiple elements are defined as separate rows of a nested table	NOTE Same as for Table 2					

3.5.1.4 Additional Attribute values

The type definition table provides columns to specify the values for required Node *Attributes* for *InstanceDeclarations*. To provide information about additional *Attributes*, the format as shown in Table 6 is used.

Table 6 – <some>Type Attribute values for child nodes

BrowsePath	<Attribute name> Attribute
BrowsePath is always relative to the TypeDefinition. Multiple elements are defined as separate rows of a nested table	The values of attributes are converted to text by adapting the reversible JSON encoding rules defined in OPC 10000-6. If the JSON encoding of a value is a JSON string or a JSON number then that value is entered in the value field. Double quotes are not included. If the DataType includes a NamespaceIndex (QualifiedNames, NodeIds or ExpandedNodeIds) then the notation used for BrowseNames is used. If the value is an Enumeration the name of the enumeration value is entered. If the value is a Structure then a sequence of name and value pairs is entered. Each pair is followed by a newline. The name is followed by a colon. The names are the names of the fields in the DataTypeDefinition. If the value is an array of non-structures then a sequence of values is entered where each value is followed by a newline. If the value is an array of Structures or a Structure with fields that are arrays or with nested Structures then the complete JSON array or JSON object is entered. Double quotes are not included.

There can be multiple columns to define more than one *Attribute*.

3.5.2 NodeIds and BrowseNames

3.5.2.1 NodeIds

The *NodeIds* of all *Nodes* described in this standard are only symbolic names. Annex A defines the actual *NodeIds*.

The symbolic name of each *Node* defined in this document is its *BrowseName*, or, when it is part of another *Node*, the *BrowseName* of the other *Node*, a ".", and the *BrowseName* of itself. In this case "part of" means that the whole has a *HasProperty* or *HasComponent Reference* to its part. Since all *Nodes* not being part of another *Node* have a unique name in this document, the symbolic name is unique.

The *NamespaceUri* for all *NodeIds* defined in this document is defined in Annex A. The *NamespaceIndex* for this *NamespaceUri* is vendor-specific and depends on the position of the *NamespaceUri* in the server namespace table.

Note that this document not only defines concrete *Nodes*, but also requires that some *Nodes* shall be generated, for example one for each *Session* running on the *Server*. The *NodeIds* of those *Nodes* are *Server*-specific, including the namespace. But the *NamespaceIndex* of those *Nodes* cannot be the *NamespaceIndex* used for the *Nodes* defined in this document, because they are not defined by this document but generated by the *Server*.

3.5.2.2 BrowseNames

The text part of the *BrowseNames* for all *Nodes* defined in this document is specified in the tables defining the *Nodes*. The *NamespaceUri* for all *BrowseNames* defined in this document is defined Annex A.

For *InstanceDeclarations* of *NodeClass Object* and *Variable* that are placeholders (*OptionalPlaceholder* and *MandatoryPlaceholder ModellingRule*), the *BrowseName* and the *DisplayName* are enclosed in angle brackets (<>) as recommended in OPC 10000-3. If the *BrowseName* is not defined by this document, a namespace index prefix is added to the *BrowseName* (e.g., prefix '0' leading to '0:EngineeringUnits' or prefix '2' leading to '2:DeviceRevision'). This is typically necessary if a *Property* of another specification is overwritten or used in the OPC UA types defined in this document. Table 94 provides a list of namespaces and their indexes as used in this document.

3.5.3 Common Attributes

3.5.3.1 General

The *Attributes* of *Nodes*, their *DataTypes* and descriptions are defined in OPC 10000-3. Attributes not marked as optional are mandatory and shall be provided by a *Server*. The following tables define if the *Attribute* value is defined by this specification or if it is server-specific.

For all *Nodes* specified in this specification, the *Attributes* named in Table 7 shall be set as specified in the table.

Table 7 – Common Node Attributes

Attribute	Value
DisplayName	The *DisplayName* is a *LocalizedText*. Each server shall provide the *DisplayName* identical to the *BrowseName* of the *Node* for the LocaleId "en". Whether the server provides translated names for other LocaleIds is server-specific.
Description	Optionally a server-specific description is provided.
NodeClass	Shall reflect the *NodeClass* of the *Node.*
NodeId	The *NodeId* is described by *BrowseNames* as defined in 3.5.2.1.
WriteMask	Optionally the *WriteMask Attribute* can be provided. If the *WriteMask Attribute* is provided, it shall set all non-server-specific *Attributes* to not writable. For example, the *Description Attribute* may be set to writable since a *Server* may provide a server-specific description for the *Node*. The *NodeId* shall not be writable, because it is defined for each *Node* in this specification.
UserWriteMask	Optionally the *UserWriteMask Attribute* can be provided. The same rules as for the *WriteMask Attribute* apply.
RolePermissions	Optionally server-specific role permissions can be provided.
UserRolePermissions	Optionally the role permissions of the current Session can be provided. The value is server-specifc and depend on the *RolePermissions Attribute* (if provided) and the current *Session*.
AccessRestrictions	Optionally server-specific access restrictions can be provided.

3.5.3.2 Objects

For all *Objects* specified in this specification, the *Attributes* named in Table 8 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 8 – Common Object Attributes

Attribute	Value
EventNotifier	Whether the *Node* can be used to subscribe to *Events* or not is server-specific.

3.5.3.3 Variables

For all *Variables* specified in this specification, the *Attributes* named in Table 9 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 9 – Common Variable Attributes

Attribute	Value
MinimumSamplingInterval	Optionally, a server-specific minimum sampling interval is provided.
AccessLevel	The access level for *Variables* used for type definitions is server-specific, for all other *Variables* defined in this specification, the access level shall allow reading; other settings are server-specific.
UserAccessLevel	The value for the *UserAccessLevel Attribute* is server-specific. It is assumed that all *Variables* can be accessed by at least one user.
Value	For *Variables* used as *InstanceDeclarations,* the value is server-specific; otherwise it shall represent the value described in the text.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is server-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *Variable*.
Historizing	The value for the *Historizing Attribute* is server-specific.
AccessLevelEx	If the *AccessLevelEx Attribute* is provided, it shall have the bits 8, 9, and 10 set to 0, meaning that read and write operations on an individual *Variable* are atomic, and arrays can be partly written.

3.5.3.4 VariableTypes

For all *VariableTypes* specified in this specification, the *Attributes* named in Table 10 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 10 – Common VariableType Attributes

Attributes	Value
Value	Optionally a server-specific default value can be provided.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behaviour is server-specific. If the *ValueRank* specifies an array of a specific dimension (i.e. *ValueRank* > 0) then the *ArrayDimensions Attribute* shall be specified in the table defining the *VariableType*.

3.5.3.5 Methods

For all *Methods* specified in this specification, the *Attributes* named in Table 11 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 11 – Common Method Attributes

Attributes	Value
Executable	All *Methods* defined in this specification shall be executable (*Executable Attribute* set to “True”), unless it is defined differently in the *Method* definition.
UserExecutable	The value of the *UserExecutable Attribute* is server-specific. It is assumed that all *Methods* can be executed by at least one user.

4 General information to glass technology and OPC UA

4.1 Introduction to glass technology

4.1.1 Overview

The glass industry is one of the base material industries that, on the one hand has a long tradition, and, on the other hand, is crucial for all modern applications. It ranges from, table ware, bottles for beverages and to perfume up to flat and bended glass for facades, windows, cars and technological applications such as for televisions and mobile devices. Figure 2 gives an overview of the glass domain. This specification only describes the flat glass domain. Other part will be defined OPC UA information models for other domains.

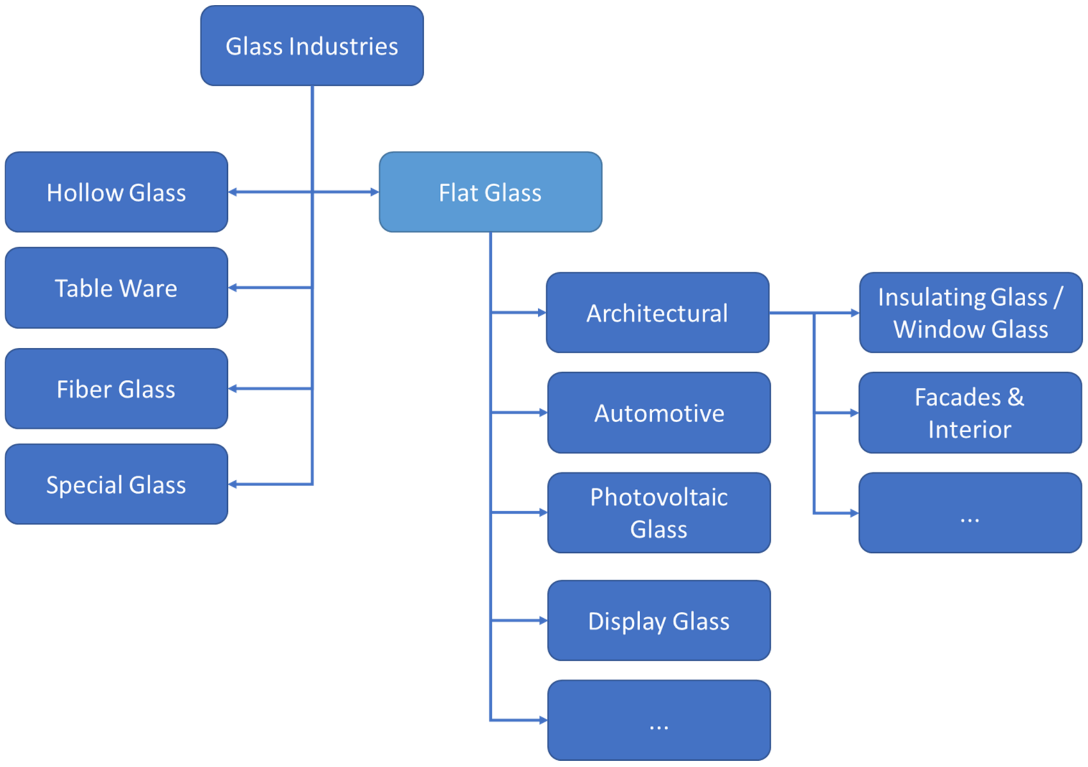


Figure 2 – Overview glass industries domains

Focusing on flat glass, the principal process steps for flat glass products are shown in simplified form in Figure 3.

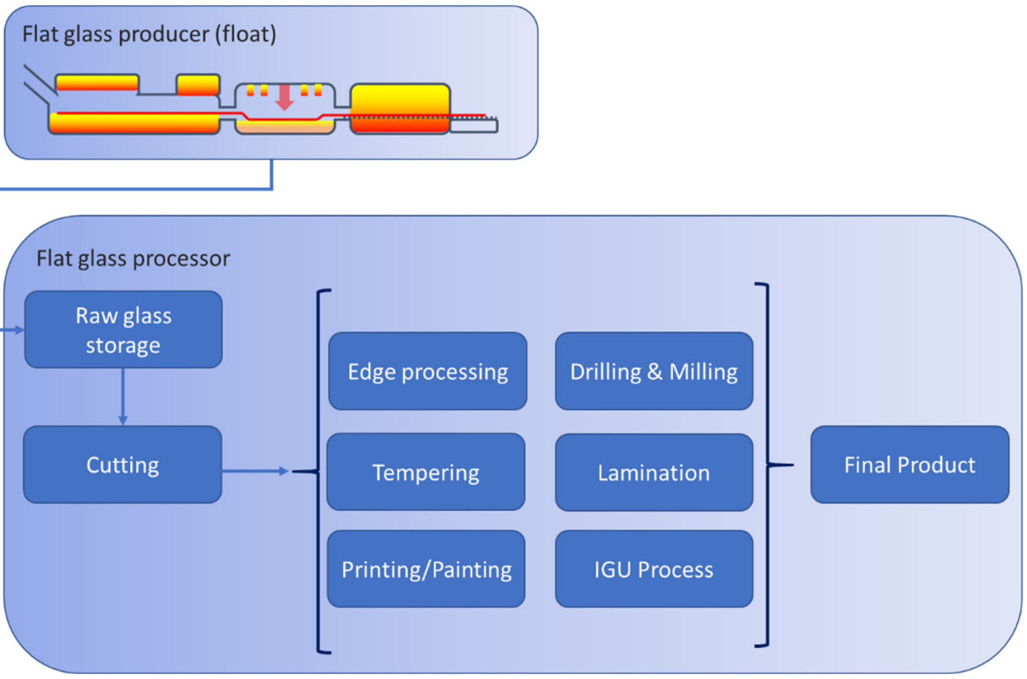


Figure 3 – Typical processing steps in flat glass production

Flat glass can be produced in two ways. The first is by the use of the float process, a second possibility is the rolled glass process. Float glass producers hold large glass tanks to melt and pull glass in the desired thickness and application of required coating. Such glass with a thickness between 2 and 20mm, typically cut to size of 6000 x 3210 mm and stocked on racks for transport, is typically the raw material for the flat glass processors. The flat glass processer transforms the raw glass into divers glass products of various sizes and shapes such as window glass, splash backs, shopfronts, stairs, tables etc. In any case, the cutting process is the most universal process that is required for basically all raw glass processings and therefore the example of flat glass cutting is used in the subsequent chapters as reference object. The most important processes are described in more detail below:

- **Cutting: mechanical flat glass cutting is a sequenced process in which:**
  - a raw glass pane is transported from storage to the cutting table
  - the cutting pattern is inscribed onto the glass surface, e.g., by means of a very small sharpened stainless-steel wheel
  - the glass is broken along the prescribed lines by applicating a bending force, either manual or automatic
  - optional, there may be a grinding head mounted, allowing to eliminate existing glass coating on prescribed areas
  - special cutting for previously laminated glass, where as an additional process after the glass breaking the foil attaching the 2 glass panes is separated mostly by a thermomechanical process.
- **Processing:** Flat glass processing can include :
  - Edge works, such as edge seaming, grinding and or polishing
  - Surface works, such as drilling holes or generating cut outs, required e.g. for inserting of hinges and handles as required for shower doors.
  - Heat treatment, like tempering to change tension and behaviour of glass, as e.g. required for car side windows (tempered glass which in case of break dissipates in small cullets)
  - Bending (heating plus application of gravity or pressing force) to change the glass´shape into curved glass
- **Assembly:** multiple glass panes may be assembled to (semi-) products

- Lamination, in which multiple glass panes are glued together by means of interlaying thermoplastic foils. For this process the glass is first cleaned, then stacked with the interlayers and further processed in a thermopressure process to initiate the robustness of the product and the required transparency.
- Insulating Glass production: for windows and facade elements in which two or more glass panes are assembled holding glass spacers between the glass elements. The resulting volume is in general filled with inert gas such as Argon to ensure high insulation values.

4.1.2 Relevance to OPC UA Model

For OPC UA modelling it is required to respect main characteristics of flat glass processing:

- Glass processing is very individual and a good example of industrial batch size of a production.
- As a result, all machinery must be able to receive information on single glass element base in contrast to mass production.
- Glass processing is very often based on intensive co-working between machine and human. Thus, processes with non-digitized interfaces shall be possible.

4.1.3 Different kinds of processing:

As the OPC UA model shall be universal for flat glass processing, resulting major requirements are deducted.

One of them is that we have different types of transformation processes:

1. **One to many:** Cutting, where raw glass panes are cut into multiple glass elements
2. **One to one:** Mainly single glass element transformation processes such as edge and surface works, heat treatment, painting
3. **Many to one:** Classical assembly processes such as lamination and insulating glass fabrication

4.1.3.1 Processing requirements:

- In order to establish an OPC UA Model suitable to the different processes, the main requirements of information transport and modelling, established over years in the glass industry, shall be integrated. A job is defined as concrete work order for one or more glass elements. For the transformation, the job may reference to preadjusted work recipes in the respective machines or production lines recipes by means
- Jobs can be grouped in a job group (as a set of job in a fix order)
- The order of jobs and of job groups may be changed
- Each job group and job have a status (see ProductionStateMachine)
- Each transformation process is one job, e.g.,
  - Cutting 100 raw glass panes requires 100 jobs as each pane will be cut in an individual pattern.
  - IG Line production of a lot requires an individual setting for each glass element produced.

4.2 Introduction to OPC Unified Architecture

4.2.1 What is OPC UA?

OPC UA is an open and royalty free set of standards designed as a universal communication protocol. While there are numerous communication solutions available, OPC UA has key advantages:

– A state of art security model (see OPC 10000-2).

– A fault tolerant communication protocol.

– An information modelling framework that allows application developers to represent their data in a way that makes sense to them.

OPC UA has a broad scope which delivers for economies of scale for application developers. This means that a larger number of high-quality applications at a reasonable cost are available. When combined with semantic models such as flat glass, OPC UA makes it easier for end users to access data via generic commercial applications.

The OPC UA model is scalable from small devices to ERP systems. OPC UA *Servers* process information locally and then provide that data in a consistent format to any application requesting data - ERP, MES, PMS, Maintenance Systems, HMI, Smartphone or a standard Browser, for examples. For a more complete overview see OPC 10000-1.

4.2.2 Basics of OPC UA

As an open standard, OPC UA is based on standard internet technologies, like TCP/IP, HTTP, Web Sockets.

As an extensible standard, OPC UA provides a set of *Services* (see OPC 10000-4) and a basic information model framework. This framework provides an easy manner for creating and exposing vendor defined information in a standard way. More importantly all OPC UA *Clients* are expected to be able to discover and use vendor-defined information. This means OPC UA users can benefit from the economies of scale that come with generic visualization and historian applications. This specification is an example of an OPC UA *Information Model* designed to meet the needs of developers and users.

OPC UA *Clients* can be any consumer of data from another device on the network to browser based thin clients and ERP systems. The full scope of OPC UA applications is shown in **Figure 4**.

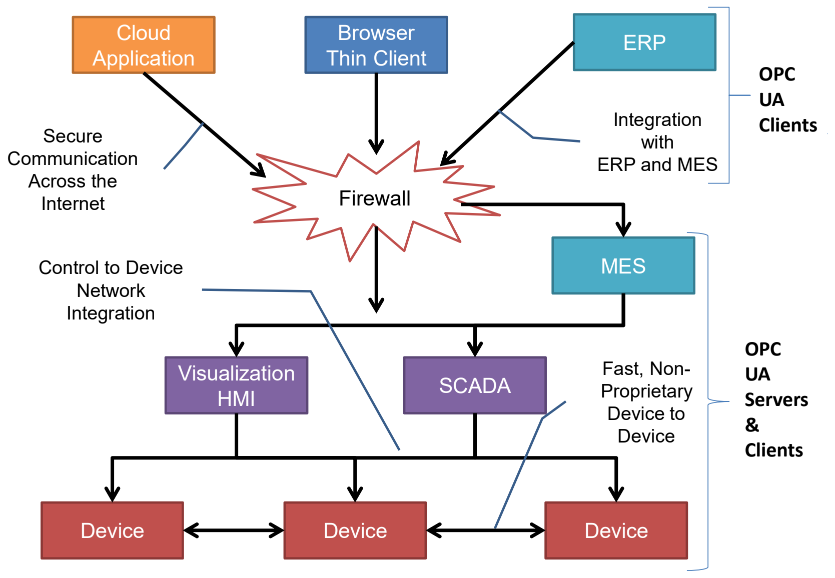

Cloud Application
Browser Thin Client
ERP
OPC UA Clients
Secure Communication Across the Internet
Integration with ERP and MES
Firewall
MES
Control to Device Network Integration
Visualization HMI
SCADA
Fast, Non-Proprietary Device to Device
OPC UA Servers & Clients
Device
Device
Device

Figure 4 – The Scope of OPC UA within an Enterprise

OPC UA provides a robust and reliable communication infrastructure having mechanisms for handling lost messages, failover, heartbeat, etc. With its binary encoded data, it offers a high-performing data exchange solution. Security is built into OPC UA as security requirements become more and more important especially since environments are connected to the office network or the internet and attackers are starting to focus on automation systems.

4.2.3 Information modelling in OPC UA

4.2.3.1 Concepts

OPC UA provides a framework that can be used to represent complex information as *Objects* in an *AddressSpace* which can be accessed with standard services. These *Objects* consist of *Nodes* connected by *References*. Different classes of *Nodes* convey different semantics. For example, a *Variable Node* represents a value that can be read or written. The *Variable Node* has an associated *DataType* that can define the actual value, such as a string, float, structure etc. It can also describe the *Variable* value as a variant. A *Method Node* represents a function that can be called. Every *Node* has a number of *Attributes* including a unique identifier called a *NodeId* and non-localized name called as *BrowseName*. An *Object* representing a ‘Reservation’ is shown in Figure 5.

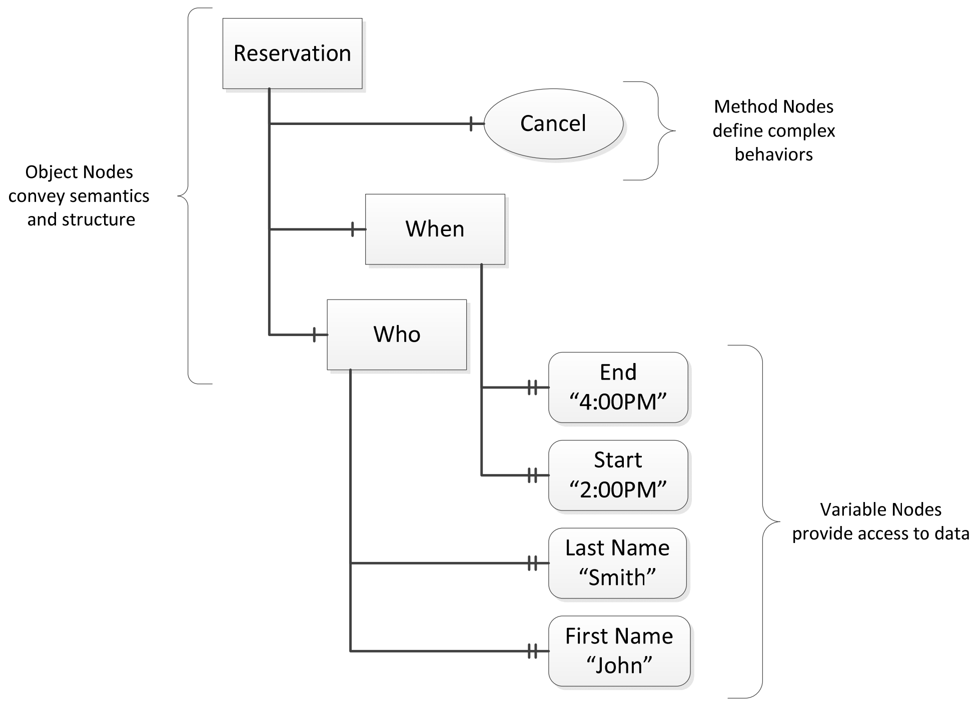

Reservation
Cancel
Method Nodes define complex behaviors
Object Nodes convey semantics and structure
When
Who
End "4:00PM"
Start "2:00PM"
Last Name "Smith"
First Name "John"
Variable Nodes provide access to data

Figure 5 – A Basic Object in an OPC UA Address Space

Object and *Variable Nodes* represent instances and they always reference a *TypeDefinition* (*ObjectType* or *VariableType*) *Node* which describes their semantics and structure. Figure 6 illustrates the relationship between an instance and its *TypeDefinition*.

The type *Nodes* are templates that define all of the children that can be present in an instance of the type. In the example in Figure 6 the PersonType *ObjectType* defines two children: First Name and Last Name. All instances of PersonType are expected to have the same children with the same *BrowseNames*. Within a type the *BrowseNames* uniquely identify the children. This means *Client* applications can be designed to search for children based on the *BrowseNames* from the type instead of *NodeIds*. This eliminates the need for manual reconfiguration of systems if a *Client* uses types that multiple *Servers* implement.

OPC UA also supports the concept of sub-typing. This allows a modeller to take an existing type and extend it. There are rules regarding sub-typing defined in OPC 10000-3, but in general they allow the extension of a given type or the restriction of a *DataType*. For example, the modeller may decide that the existing *ObjectType* in some cases needs an additional *Variable*. The modeller can create a subtype of the *ObjectType* and add the *Variable*. A *Client* that is expecting the parent type can treat the new type as if it was of the parent type. Regarding *DataTypes*, subtypes can only restrict. If a *Variable* is defined to have a numeric value, a sub type could restrict it to a float.

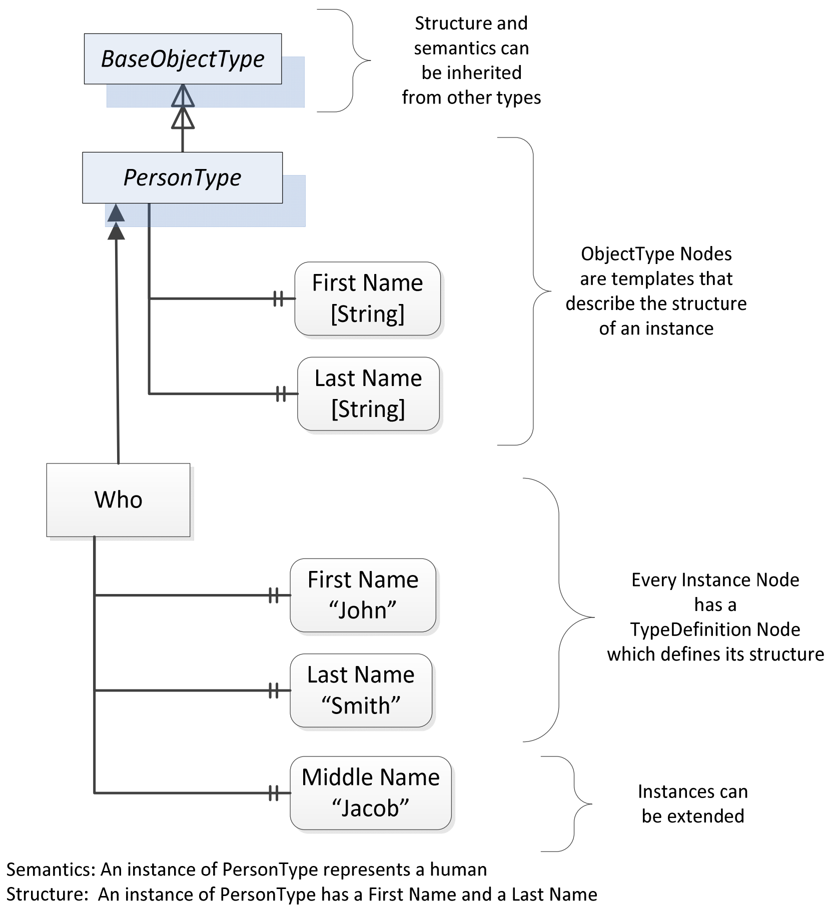

BaseObjectType
Structure and semantics can be inherited from other types
PersonType
First Name [String]
Last Name [String]
ObjectType Nodes are templates that describe the structure of an instance
Who
First Name "John"
Last Name "Smith"
Every Instance Node has a TypeDefinition Node which defines its structure
Middle Name "Jacob"
Instances can be extended
Semantics: An instance of PersonType represents a human
Structure: An instance of PersonType has a First Name and a Last Name

Figure 6 – The Relationship between Type Definitions and Instances

References allow *Nodes* to be connected in ways that describe their relationships. All *References* have a *ReferenceType* that specifies the semantics of the relationship. *References* can be hierarchical or non-hierarchical. Hierarchical references are used to create the structure of *Objects* and *Variables*. Non-hierarchical are used to create arbitrary associations. Applications can define their own *ReferenceType* by creating subtypes of an existing *ReferenceType*. Subtypes inherit the semantics of the parent but may add additional restrictions. Figure 7 depicts several *References,* connecting different *Objects*.

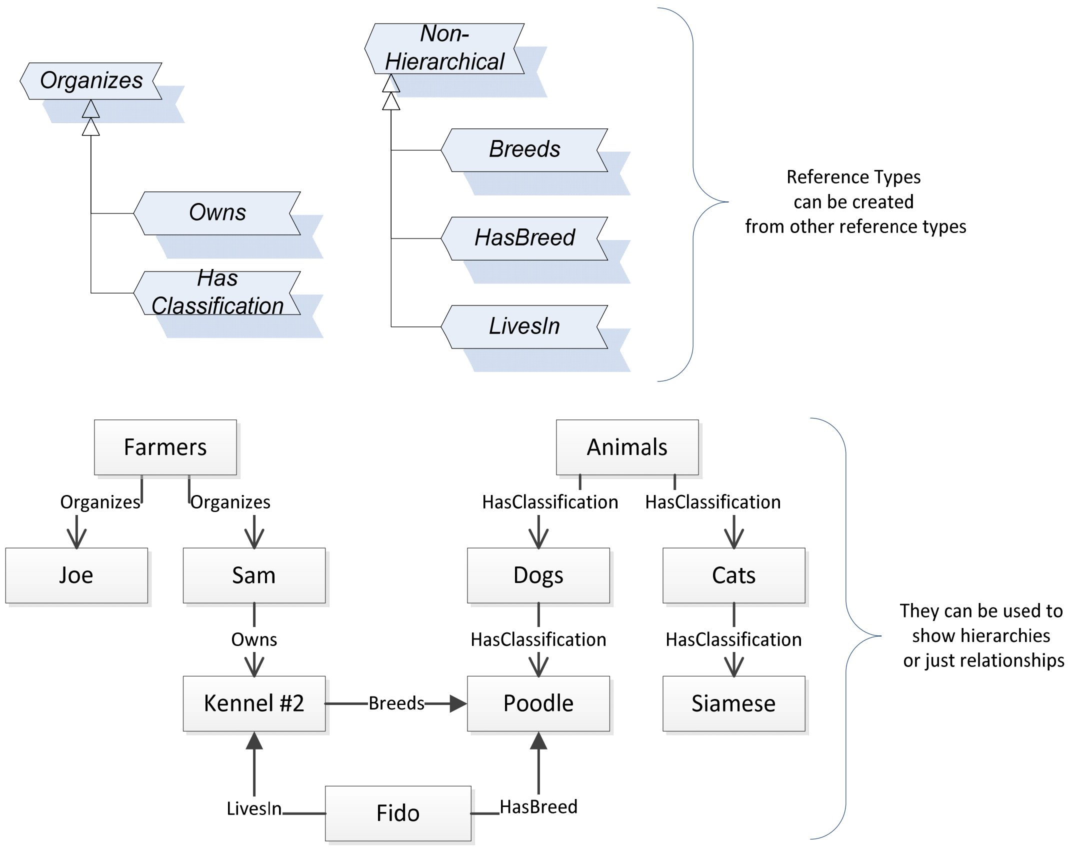


Figure 7 – Examples of References between Objects

The figures above use a notation that was developed for the OPC UA specification. The notation is summarized in Figure 8. UML representations can also be used; however, the OPC UA notation is less ambiguous because there is a direct mapping from the elements in the figures to *Nodes* in the *AddressSpace* of an OPC UA *Server*.

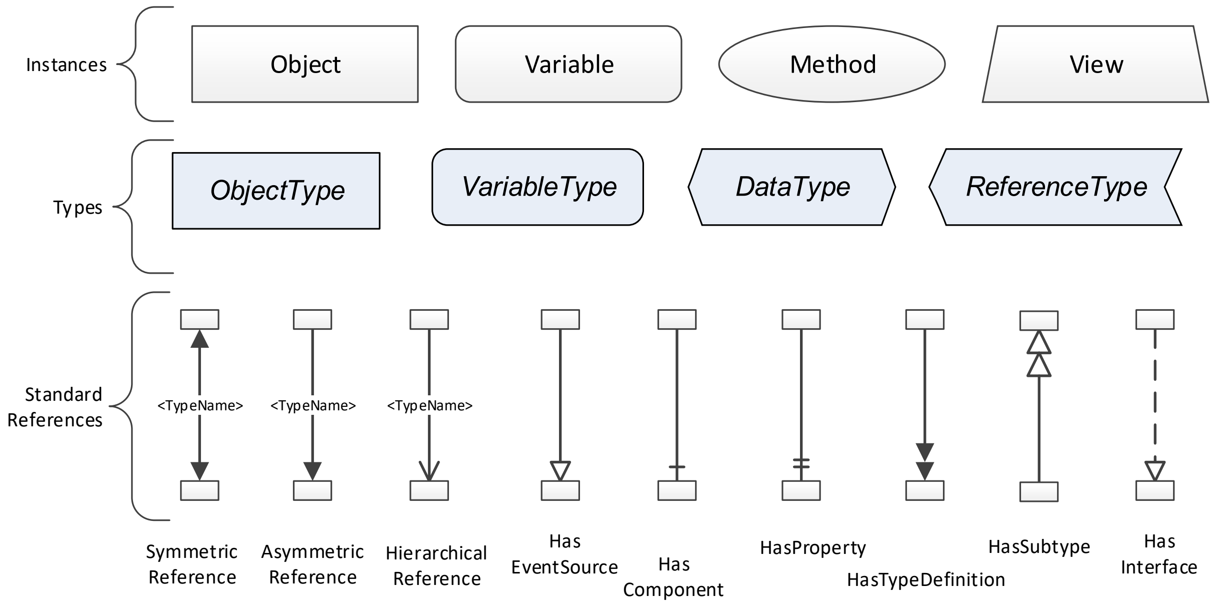


Figure 8 – The OPC UA Information Model Notation

A complete description of the different types of Nodes and References can be found in OPC 10000-3 and the base structure is described in OPC 10000-5.

OPC UA specification defines a very wide range of functionality in its basic information model. It is not required that all *Clients* or *Servers* support all functionality in the OPC UA specifications. OPC UA includes the concept of *Profiles*, which segment the functionality into testable certifiable units. This allows the definition of functional subsets (that are expected to be implemented) within a companion specification. The *Profiles* do not restrict functionality, but generate requirements for a minimum set of functionality (see OPC 10000-7)

4.2.3.2 Namespaces

OPC UA allows information from many different sources to be combined into a single coherent *AddressSpace*. Namespaces are used to make this possible by eliminating naming and id conflicts between information from different sources. Each namespace in OPC UA has a globally unique string called a NamespaceUri which identifies a naming authority and a locally unique integer called a NamespaceIndex, which is an index into the *Server's* table of *NamespaceUris*. The *NamespaceIndex* is unique only within the context of a *Session* between an OPC UA *Client* and an OPC UA *Server*- the *NamespaceIndex* can change between *Sessions* and still identify the same item even though the NamespaceUri's location in the table has changed. The *Services* defined for OPC UA use the *NamespaceIndex* to specify the Namespace for qualified values.

There are two types of structured values in OPC UA that are qualified with *NamespaceIndexes*: NodeIds and *QualifiedNames*. NodeIds are locally unique (and sometimes globally unique) identifiers for *Nodes*. The same globally unique *NodeId* can be used as the identifier in a node in many *Servers* – the node's instance data may vary but its semantic meaning is the same regardless of the *Server* it appears in. This means *Clients* can have built-in knowledge of of what the data means in these *Nodes*. OPC UA *Information Models* generally define globally unique *NodeIds* for the *TypeDefinitions* defined by the *Information Model*.

QualifiedNames are non-localized names qualified with a Namespace. They are used for the *BrowseNames* of *Nodes* and allow the same names to be used by different information models without conflict. *TypeDefinitions* are not allowed to have children with duplicate *BrowseNames*; however, instances do not have that restriction.

4.2.3.3 Companion Specifications

An OPC UA companion specification for an industry specific vertical market describes an *Information Model* by defining *ObjectTypes*, *VariableTypes*, *DataTypes* and *ReferenceTypes* that represent the concepts used in the vertical market, and potentially also well-defined Objects as entry points into the AddressSpace.

5 Use cases

5.1 Basic

5.1.1 Machine identification to control systems

Control systems shall be able to identify a glass processing machine in a standardized way within an accessible environment. For this purpose, a minimum amount of information specified in the Companion Specification is transmitted from the machine to the control system. This includes basic data about the machine-like serial number, manufacturer, production data, etc. Also, information that is useful for job planning is included, so that the machine is only supplied with suitable jobs. Furthermore, the control system is informed of the storage location of the machine documentation. The ObjectTypes from OPC UA for Machinery are used. Detailed description of the Types can be found within MachineryIdentificationType (OPC 40001-1) and GlassMachineIdentificationType in section 7.1.

5.1.2 Request of machine status

Control systems shall be able to request the current status of a specific glass processing machine. The machine supplies at least the dynamic information specified in the Companion Specification with the job state machine and events. In addition, the machine status of OPC 40001-1 is to be added in the future to obtain an overall status of the machine.

5.1.3 Request of machine documentation

The machine provides its own documentation. This documentation is either stored directly on the machine or somewhere external, in which case an URL is provided. If a user or another party (e.g. document management system) needs the machine documentation, it can be requested via OPC UA from the server of the machine. This information can be found in the identification section (see 7.1.3).

5.1.4 Identification of logged in users

It is possible to query a list of the currently logged in users (local and connected via OPC UA) and their characteristics (e.g. language, access level) via the OPC UA server of the machine. This information can be found in the identification section (see 7.1.4).

5.1.5 Identification of the capabilities of a machine

The control system can query the process capabilities of the glass processing machine from the OPC UA server. In the case of an IGU line, for example, this can be the maximum possible window size. Furthermore, active and temporarily inactive capabilities are distinguished. For example, if a cutting table can basically cut 10 mm thick glass, but the currently used cutting wheel can only cut up to 5 mm thick. This use case is not implemented in the current version, but should be defined in further versions.

5.2 Job

5.2.1 Job Management (insert, modify and delete)

A glass processing machine can be given a new job by a control system. This is done by calling the InsertJob-Method and transferring all information that is relevant to the job but has not yet been released for production. These jobs can be modified or deleted by calling the corresponding methods. If the job is currently in Running or Interrupted state, the modification must be confirmed or rejected by an operator at the machine. For deletion of the job while in Running or Interrupted status, the processing must be terminated by operator interaction (Aborted or Ended status) to enable the deletion. This information can be found in section 7.2.

5.2.2 Request the current job list

The control system can request the current job list from the OPC UA server of a glass processing machine. This information can be used to plan further jobs or to optimize the sequence. This information can be found in section 7.2.2

5.2.3 Current status of a job

The control system can request the current status of a specific job from the OPC UA server of a glass processing machine. This information can be used to plan further jobs or to optimize the sequence. This information can be found in section 7.2.5.

5.2.4 Release job

The control system can give the OPC UA server of a glass processing machine the production release for a specific job. This job can then be started by the operator at the machine or by the machine itself (full automation). This information can be found in section 7.2.4.2.

5.2.5 Suspend Job

The control system can revoke the production release for a specific job from the OPC UA server of a glass processing machine. If the job is currently in Running status, processing must be interrupted by user interaction (Interrupted status) to allow the release to be withdrawn. This information can be found in section 7.2.4.3

5.2.6 Machine or operator triggered event

The glass processing machine is expected to offer all current errors and warnings over the machine interface. These errors and warnings shall be mapped to OPC UA event types accordingly. This information can be found in section 8.

6 Glass technology Information Model overview

This chapter introduces the "OPC UA Information Model for Glass technology – flat glass".

This *Information Model* provides the necessary *ObjectTypes* to model a glass machine interface in a structure as illustrated in Figure 9 There are *ObjectTypes* that are used to identify the machine (*GlassMachineIdentificationType*), to manage the production *(Production Type)* on the machine and to get the manuals for operation or maintenance (ManualsFolderType).

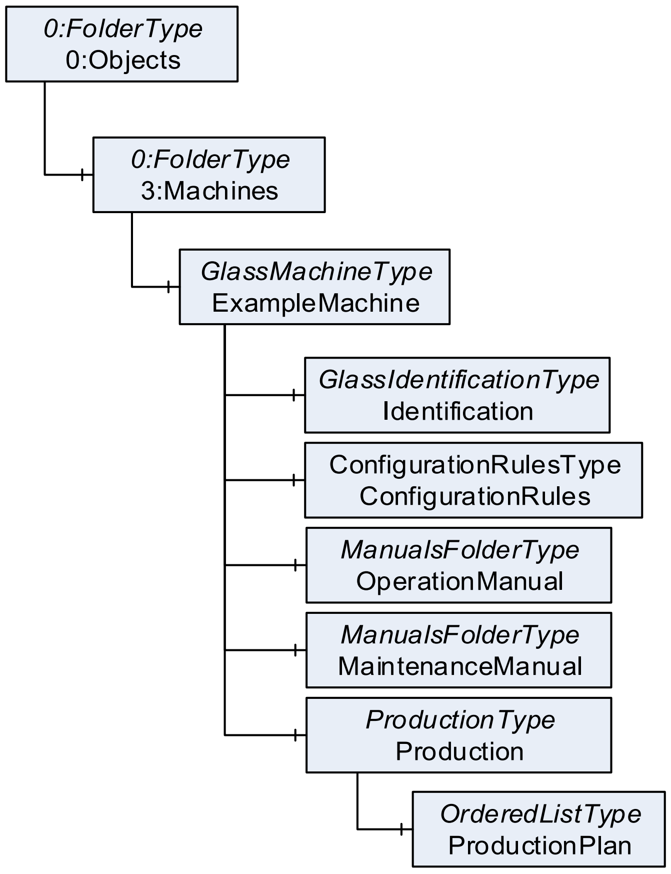


Figure 9 – Instance Example for "OPC UA Information model for Glass Machines"

The *ObjectType* hierarchy of this Companion Specification is shown within Figure 10 to Figure 14. Objects from external specifications are positioned within greyish-green boxes.

Figure 10 shows the inheritance relations of the *GlassMachineType* and all direct children. In this picture the optional components are shown with a dashed line.

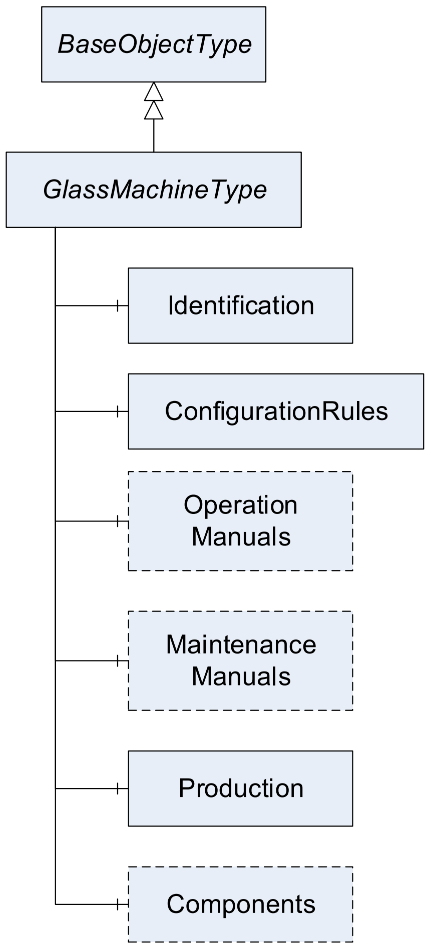


Figure 10 – Overview of the Glass Machine with all children

Figure 11 shows the inheritance hierarchy of all ObjectTypes used in the *glass technology Identification* component. This relates to the document structure in section 7.1.2.

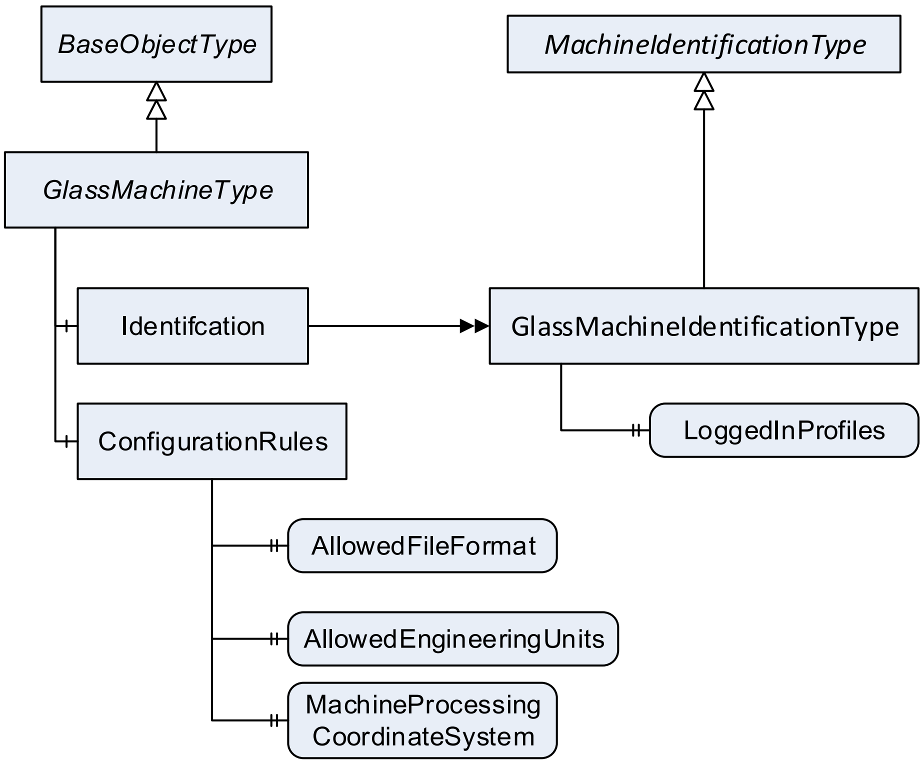


Figure 11 – Overview of the Identification part of the Glass Machine

Figure 12 shows the inheritance hierarchy of all ObjectTypes used in the glass technology as manual component. This relates to the document structure in seciton 7.1.3

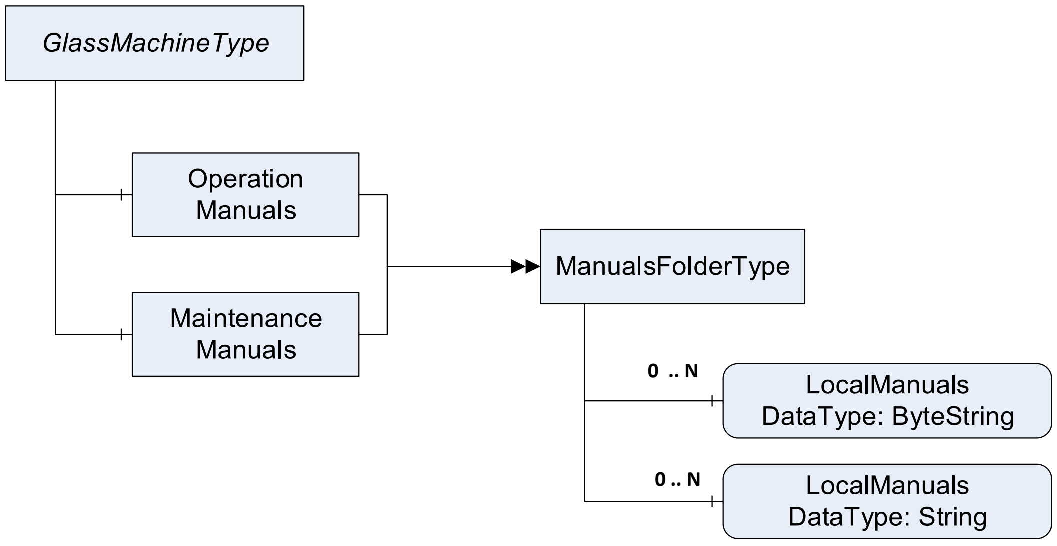


Figure 12 – Overview of the manual part of the Glass Machine

Figure 13 shows the inheritance hierarchy of all types used in the *GlassMachineType's Production* component to the level of the JobType. The inheritance hierarchy of the subtypes of the JobType and their relationship to the different Materials is shown in Figure 14. This conforms to the structure of the section *Production.*

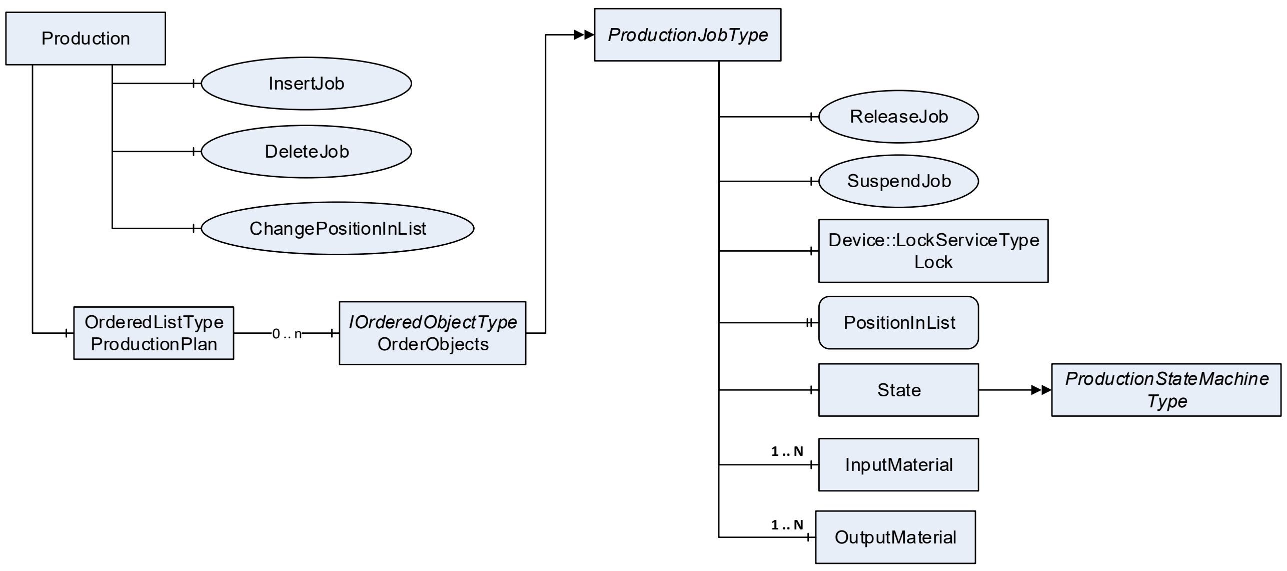


Figure 13 – Overview of the Production of the Glass Machine

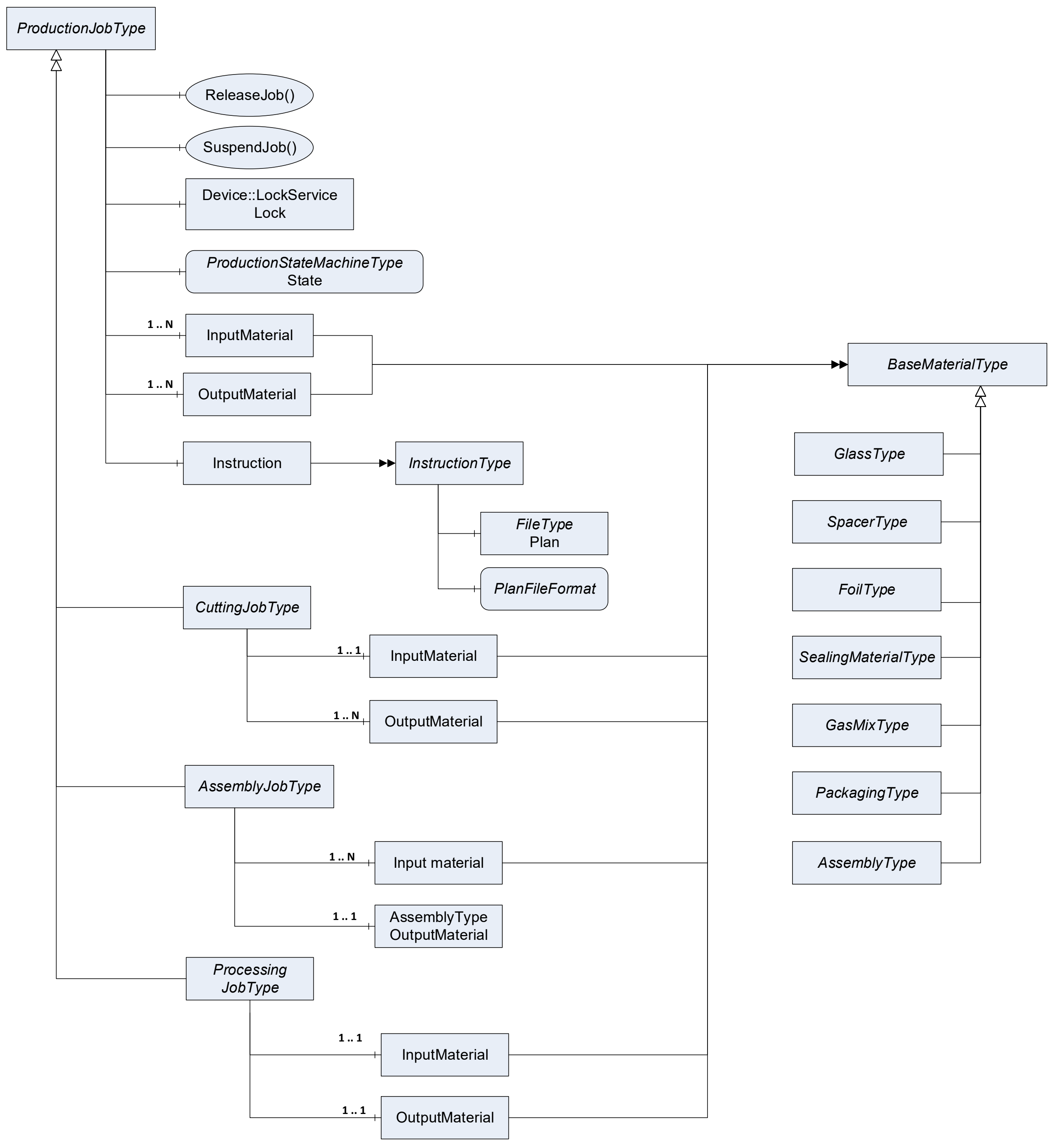


Figure 14 – Overview of the different Jobs and the relationship with Material

7 OPC UA ObjectTypes

7.1 General Types

7.1.1 GlassMachineType Definition

The GlassMachineType provides the information of the machine and is formally defined in Table 12. A Machine or system can contain other subsystems (e.g. other machines or devices). Such a complex system can be modelled with the component structure from OPC UA Machinery (see OPC 40001-1, section 11).

This GlassMachine object can be further divided into subtypes using the modular device structure from OPC UA for Machinery.

Table 12 – GlassMachineType Definition

Attribute	**Value**				
BrowseName	GlassMachineType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseObjectType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
HasAddIn	Object	2:Identification		GlassMachineIdentificationType	Mandatory
0:HasComponent	Object	MaintenanceManuals		ManualFolderType	Optional
0:HasComponent	Object	OperationManuals		ManualFolderType	Optional
0:HasComponent	Object	Production		ProductionType	Mandatory
HasAddIn	Object	Components		3:MachineComponentsType	Optional
0:HasComponent	Object	ConfigurationRules		ConfigurationRulesType	Mandatory

Identification contains the information to identify the glass machine. For more information see GlassMachineIdentificationType and OPC 40001-1 (MachineIdentificationType).

MaintenanceManuals contains the manuals or the references to the manuals for the maintenance process.

OperationManuals contains the manuals or the references to the manuals for the operation process.

Production contains the information and methods that relate to the production. This includes the states of the current jobs and methods for insert, remove, release, suspend a job.

Components contains components or submachines of the glass machine.

ConfigurationRules contains the properties that describe the machine configuration.

7.1.2 GlassMachineIdentificationType Definition

The GlassMachineIdentificationType provides the information about the identification process and is formally defined in Table 13.

Table 13 – GlassMachineIdentificationType Definition

Attribute	**Value**				
BrowseName	GlassMachineIdentificationType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the 3:MachineIdentificationType defined in this OPC UA 400001-1 , i.e. inheriting the InstanceDeclarations of that Node.					
0:HasProperty	Variable	LoggedInProfiles	UserProfileType[]	0:PropertyType	Optional

LoggedInProfiles contains a List of logged (local and via OPC UA) in user profiles at the machine.

7.1.3 ManualFolderType Definition

The ManualFolderType provides information about the manuals and is formally defined in Table 14.

Table 14 – ManualFolderType Definition

Attribute	**Value**				
BrowseName	ManualFolderType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the FolderType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	<LocalManuals>		0:FileType	OptionalPlaceholder
0:HasProperty	Variable	ExternalManuals	LimitedString64[]	0:PropertyType	Optional

<LocalManuals> contains all manuals, which are stored on the machine control memory and can be accessed via OPC UA.

ExternalManuals contains URIs (by RFC 3986) of manuals, which are stored on external systems. Example: https://example.com/manual/5789; ftp://example.com/manual/234985923

7.1.4 ConfigurationRulesType Definition

The ConfigurationRulesType provides information about the configuration of the machine. This includes all nodes in the OPC UA address space that are related to the machine. It also contains the possible file format and units, which can be handeled by the server. It is formally defined inTable 14.

Table 15 – ConfigurationRulesType Definition

Attribute	**Value**				
BrowseName	ConfigurationRulesType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseObjectType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasProperty	Variable	AllowedFileFormats	FileFormatType[]	0:PropertyType	Optional
0:HasProperty	Variable	AllowedEngineeringUnits	0:EUInformation[]	0:PropertyType	Optional
0:HasProperty	Variable	MachineProcessingCoordinate System	CoordinateSystemEnumerati on	0:PropertyType	Mandatory

AllowedFileFormats contains all file formats allowed for this machine. A file format describes the syntax and semantic of a document. If there are different versions allowed all must be added.

AllowedEngineeringUnits contains an array of engineering units that can be handled by the OPC UA server for this machine. A machine that supports a method with the input argument EUInformation must also provide this array.

Note: It is recommended to use SI units or units derived from SI units The following units should be used:

- Millimeter (mm) for Length
- Kilogram (kg) for Weight

MachineProcessingCoordinateSystem specifies where the machine coordinate origin is located and in which direction the axes are pointing.

7.2 Production

7.2.1 ProductionType Definition

7.2.1.1 ObjectType Definition

The ProductionType provides a job management structure to manage the machine capacity utilization and is formally defined in Table 16.

Table 16 – ProductionType Definition

Attribute	Value				
BrowseName	ProductionType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseObjectType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Method	ChangePositionInList			Optional
0:HasComponent	Method	DeleteJob			Optional
0:HasComponent	Method	InsertJob			Optional
0:HasComponent	Object	ProductionPlan		ProductionPlanType	Mandatory
0:HasProperty	Variable	JobListIsRecommendation	0:Boolean	0:PropertyType	Mandatory
0:HasProperty	Variable	MaxCountOfJobs	0:UInt32	0:PropertyType	Optional
0:HasProperty	Variable	CurrentCountOfJobs	0:UInt32	0:PropertyType	Optional
0:HasProperty	Variable	SupportedMaterialTypes	0:NodeId[]	0:PropertyType	Optional

ProductionPlan defines a folder for the placeholders of jobs and keeps all information regarding the jobs. The job list contains all jobs that should be processed with this machine. The jobs are sorted in the ProductionPlan, but that sorting can be changed. If a job is ended, the machine decides which Job will be processed next.

JobListIsRecommendation is a flag which indicates that the sequence in the ProductionPlan is a recommendation. That means that the machine and/or the operator can modify the sequence. If the first job in the sequence is interrupted or idled/queued another job can be processed. The sequence in the JobList is fixed if the flag is false and if the first job cannot be processed, an interaction is necessary.

MaxCountOfJobs contains the maximum amount of jobs, which the server is able to handle within its memory capacity. *CurrentCountOfJobs* contains the current number of jobs, which are stored on the server at the moment.

SupportedMaterialTypes contains the NodeId of all MaterialTypes (subtypes of the BaseMaterialType) that can be used in this machine.

7.2.1.2 ChangePositionInListMethod

This method requests a change of the PositionInList to modify the processing order. The new position is specified by the NodeId of another job. If the flag ´before´ is true, the new position is before the given job. If the flag ´before´ is false, the new position will be after the given job. For example, the list contains 3 jobs (job_a, job_b, job_c), if the method is called for job_c with the NodeId of job_a and before is true, the new list is job_c, job_a, job_b. If the same method is called with the flag false, the new list is job_a, job_c, job_b.

The signature of this Method is specified below in Table 17 and Table 18 which specify the Arguments and AddressSpace representation, respectively.

Signature

```
ChangePositionInList (
   [in]    LimitedString64  Target,
   [in]    LimitedString64  Source,
   [in]    0:Boolean        Before);
```

Table 17 – ChangePositionInList Method Arguments

Argument	Description
Target	NodeID of the target job
Source	NodeID of the source job
Before	Boolean: If it is true the actual Job is moved before the target in list, if it is false the actual Job if moved behind.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.

Table 18 – ChangePositionInList Method AddressSpace definition

Attribute	**Value**				
BrowseName	ChangePositionInList				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
0:HasProperty	Variable	0:InputArguments	0:Argument[]	0:PropertyType	0:Mandatory

7.2.1.3 DeleteJob Method

Method to delete a specific job. The HLS sends a deletion request to the machine. As a result, the job is removed from the job list. This is only possible, if the job has been suspended before. Suspend acts as a parking place.

The signature of this Method is specified below in Table 19 and Table 20 which specify the Arguments and AddressSpace representation, respectively.

Signature

```
DeleteJob (
   [in] LimitedString64     Identifier
);
```

Table 19 – DeleteJob Method Arguments

Argument	**Description**
Identifier	Unique Identifier for the job

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.

Table 20 – DeleteJob Method AddressSpace definition

Attribute	**Value**				
BrowseName	DeleteJob				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
0:HasProperty	Variable	0:InputArguments	0:Argument[]	0:PropertyType	0:Mandatory

7.2.1.4 InsertJob Method

Method to create a new job. HLS creates and sends a job into the job list on the machine. The JobState is suspended (= default job state).

The signature of this Method is specified below in Table 21 and Table 22 which specify the Arguments and AddressSpace representation, respectively.

The AddressSpace definition can be omitted if there are no Properties other than InputArguments and OutputArguments.

Signature

```
InsertJob (
   [in]    LimitedString64  Identifier,
   [in]    LimitedString64  Name,
   [in]    0:NodeId[]       InputMaterial,
    [in]    0:NodeId[]       OutputMaterial
    [out]   0:NodeId         JobNodeId
);
```

Table 21 – InsertJob Method Arguments

Argument	Description
Identifier	Unique ID for the job
Name	Human readable name of the job
InputMaterial	Optional: Describes the materials to be processed within the job. Can include MaterialBaseType, RawGlassType, (Surface)GlassType
OutputMaterial	Optional: Describes the material-result of processing from the materials side.
JobNodeId	Optional: NodeId of the new created Job object (subtype of ProductionJobType)

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.

Table 22 – InsertJob Method AddressSpace definition

Attribute	**Value**				
BrowseName	InsertJob				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
0:HasProperty	Variable	0:InputArguments	0:Argument[]	0:PropertyType	0:Mandatory
0:HasProperty	Variable	0:OutputArguments	0:Argument[]	0:PropertyType	0:Mandatory

7.2.2 ProductionPlanType Definition

The ProductionPlanType provides the list of ProductionJobTypes and is formally defined in Table 23.

Table 23 – ProductionPlanType Definition

Attribute	**Value**				
BrowseName	ProductionPlanType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the *OrderedListType* defined in OPC 10000-13, i.e. inheriting the InstanceDeclarations of that Node.					
HasOrderedComponent	Object	<OrderedObject>		ProductionJobType	OptionalPlaceholder

0:<OrderedObject> is a placeholder for any number of ProductionJobType instances. To indicate the order of jobs on the machine tool, the NumberInList parameter of the ProductionJobType is used. This index shall be 0 for the first list element and increase by one for each subsequent list element. If jobs are deleted from the list or inserted into the list, the NumberInList has to be adjusted for all following ProductionJobType instances in the list, such that the NumberInList elements always form a sequential series of numbers. For the DisplayName of the <OrderedObject>, it is recommended to use the value of the Identifier Property of the respective ProductionJobType instance.

7.2.3 InstructionType Definition

The InstructionType provides all the information required by the glass machine to perform the job. In case of an assembly job, it contains all necessary steps for the mounting of the unit. For a glass cutting process the cutting plan is provided. The InstructionType is formally defined in Table 24.

Table 24 – InstructionType Definition

Attribute	Value				
BrowseName	InstructionType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseObjectType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	Plan		0:FileType	Mandatory
0:HasProperty	Variable	PlanFileFormat	FileFormatType	0:PropertyType	Mandatory

Plan contains the instruction for the job. In case of a programmable machine, it contains the program code.

PlanFileFormat defines the file format of the plan. This is used to ensure compatibility between the plan and the machine.

7.2.4 ProductionJobType Definition

The *ProductionJobType* provides all information for a single job and is formally defined in Table 25.

Table 25 – ProductionJobType Definition

Attribute	Value				
BrowseName	ProductionJobType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseObjectType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	InputMaterials		FolderType	Mandatory
0:HasComponent	Object	OutputMaterials		FolderType	Mandatory
0:HasComponent	Variable	EndTime	0:DateTime	0:BaseDataVariableType	Mandatory
0:HasComponent	Object	Instruction		InstructionType	Mandatory
0:HasProperty	Variable	Identifier	LimitedString64	0:PropertyType	Mandatory
0:HasComponent	Object	State		ProductionStateMachineType	Mandatory
0:HasComponent	Object	Lock		2:LockingServicesType	Optional
0:HasProperty	Variable	Name	LimitedString64	0:PropertyType	Optional
0:HasComponent	Method	ReleaseJob			Optional
0:HasProperty	Variable	JobGroup	LimitedString64	0:PropertyType	Optional
0:HasComponent	Variable	StartTime		0:BaseDataVariableType	Mandatory
0:HasComponent	Method	SuspendJob			Optional
0:HasComponent	Method	AbortJob			Optional
0:HasComponent	Method	QueueJob			Optional
0:HasInterface	ObjectType	0:IOrderedObjectType			
Applied from 0:IOrderedObjectType					
0:HasProperty	Variable	NumberInList	0:UInt16	0:PropertyType	Mandatory

InputMaterials describes a folder that contains objects of subtypes of the BaseMaterialType. These represents the materials to be processed within the job. The folder contains a MandatoryPlaceholder of the BaseMaterialType.

OutputMaterial describes a folder that contains objects of subtypes of the BaseMaterialType. These represents the materials after processing from the materials side. The folder contains a MandatoryPlaceholder of the BaseMaterialType.

EndTime defines the timestamp when processing was finished and the machine switched to the ended or aborted state.

Instruction contains all the information required by the glass machine to perform the job. In case of an assembly job, it contains all necessary steps for the mounting of the unit. For a glass cutting process, the cutting plan is provided.

Identifier defines a unique identifier for the job.

State describes the current processing state in reference to the ProductionStateMachine.

Lock contains the methods and properties to get the information about the locking status of the job and to modify the lock. For more information refer to Locking Service in OPC 10000-100.

Name defines a human readable name of the job.

NumberInList is used to enumerate *ProductionJobType* instances used as list elements. This index shall be 0 for the first list element and increase by one for each subsequent list element. If nodes are deleted from the list or inserted into the list, the *NumberInList* has to be adjusted for all following nodes in the list, such that the *NumberInList* elements always form a sequential series of numbers.

StartTime defines the timestamp of starting processing when the machine switched from released to the running state.

JobGroup is an attribute to combine multiple jobs in a configurable order.

The components of the *ProductionJobType* have additional subcomponents which are defined in Table 26.

Table 26 – *ProductionJobType* Type Additional Subcomponents

BrowsePath	References	NodeClass	BrowseName	DataType	TypeDefinition	Others
InputMaterials	0:HasComponent	Object	<InputMaterial>		BaseMaterialType	MandatoryPlaceholder
OutputMaterials	0:HasComponent	Object	<OutputMaterial>		BaseMaterialType	MandatoryPlaceholder

7.2.4.1 AbortJob Method

Method to abort a job. Let the job enter the state Abort from the JobStateMachine. If the job is in running the stop will be stopped. The stop behaviour depends on the machine and the current conditions of the machine.

The signature of this Method is specified below in Table 27 and Table 28 which specify the Arguments and AddressSpace representation, respectively.

Signature

```
AbortJob (
);
```

Table 27 – AbortJob Method Arguments

Argument	Description

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.

Table 28 – AbortJob Method AddressSpace definition

Attribute	Value				
BrowseName	ReleaseJob				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

7.2.4.2 ReleaseJobMethod

Method to release a job. Let the job enter the state released from the JobStateMachine. A released job can be processed by the machine. This Release Method can be called only if the job is locked from this client (see Lock Service). This method will unlock the job within the releasing process.

The signature of this Method is specified below in Table 29 and Table 30 which specify the Arguments and AddressSpace representation, respectively.

Signature

```
ReleaseJob (
);
```

Table 29 – ReleaseJob Method Arguments

Argument	Description

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.

Table 30 – ReleaseJob Method AddressSpace definition

Attribute	**Value**				
BrowseName	ReleaseJob				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

7.2.4.3 SuspendJobMethod

A method to suspend a job. Let the job enter the state suspended from the JobStateMachine. A suspended job will not be processed by the machine.

The signature of this Method is specified below in Table 31 and Table 32 which specify the Arguments and AddressSpace representation, respectively.

Signature

```
SuspendJob (
);
```

Table 31 – SuspendJob Method Arguments

Argument	Description

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.

Table 32 – SuspendJob Method AddressSpace definition

Attribute	**Value**				
BrowseName	SuspendJob				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

7.2.4.4 QueueJobMethod

Method to queue a job which has been inserted via the InsertJob Method and all required data has been passed to the OPC UA server for production. Let the job enter the state Queued from the JobStateMachine. A queued job is ready to receive the final production release.

The signature of this Method is specified below in Table 31 and Table 32 which specify the Arguments and AddressSpace representation, respectively.

Signature

```
QueueJob(
);
```

Table 33 – QueueJob Method Arguments

Argument	Description

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	See OPC 10000-4 for a general description.

Table 34 – QueueJob Method AddressSpace definition

Attribute	Value				
BrowseName	QueueJob				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

7.2.5 ProductionStateMachineType definition

The ProductionStateMachineType describes a fundamental statemachine for processing jobs from glass machines and is formally defined in Table 38. An implementation of this statemachine is mandatory. Furthermore, it can be extended by further substatemachines (e.g. for a detailed description of the Running state). The name of each transition consists of the names of the states it connects: [*FromState*]To[*ToState*]. Their *References* are specified in Table 37.

On the first level, the state machine consists of the states Initialized (there is a sub-state-machine with Idle, Queued, Released), Running, Interrupted, Aborted and Ended. The state transitions can be initiated by HLS or by the machine itself. Figure 15 shows the state machine and the state transitions graphically.

Table 35 – ProductionStateMachineType Definition

Attribute	Value				
BrowseName	ProductionStateMachineType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the FiniteStateMachineType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	Aborted		0:StateType	
0:HasComponent	Object	AbortedToInitializing		0:TransitionType	
0:HasComponent	Variable	0:CurrentState	0:LocalizedText	0:FiniteStateVariableType	0:Mandatory
0:HasComponent	Object	Ended		0:StateType	
0:HasComponent	Object	EndedToInitializing		0:TransitionType	
0:HasComponent	Object	Initializing		0:InitialStateType	
0:HasComponent	Object	InitializingToAborted		0:TransitionType	
0:HasComponent	Object	InitializingToRunning		0:TransitionType	
0:HasComponent	Object	Interrupted		0:StateType	
0:HasComponent	Object	InterruptedToAborted		0:TransitionType	
0:HasComponent	Object	InterruptedToRunning		0:TransitionType	
0:HasComponent	Object	Running		0:StateType	
0:HasComponent	Object	RunningToAborted		0:TransitionType	
0:HasComponent	Object	RunningToEnded		0:TransitionType	
0:HasComponent	Object	RunningToInterrupted		0:TransitionType	
0:HasComponent	Object	RunningToRunning		0:TransitionType	
0:HasComponent	Object	InitializingState		InitializingSubStateMachineType	0:Mandatory

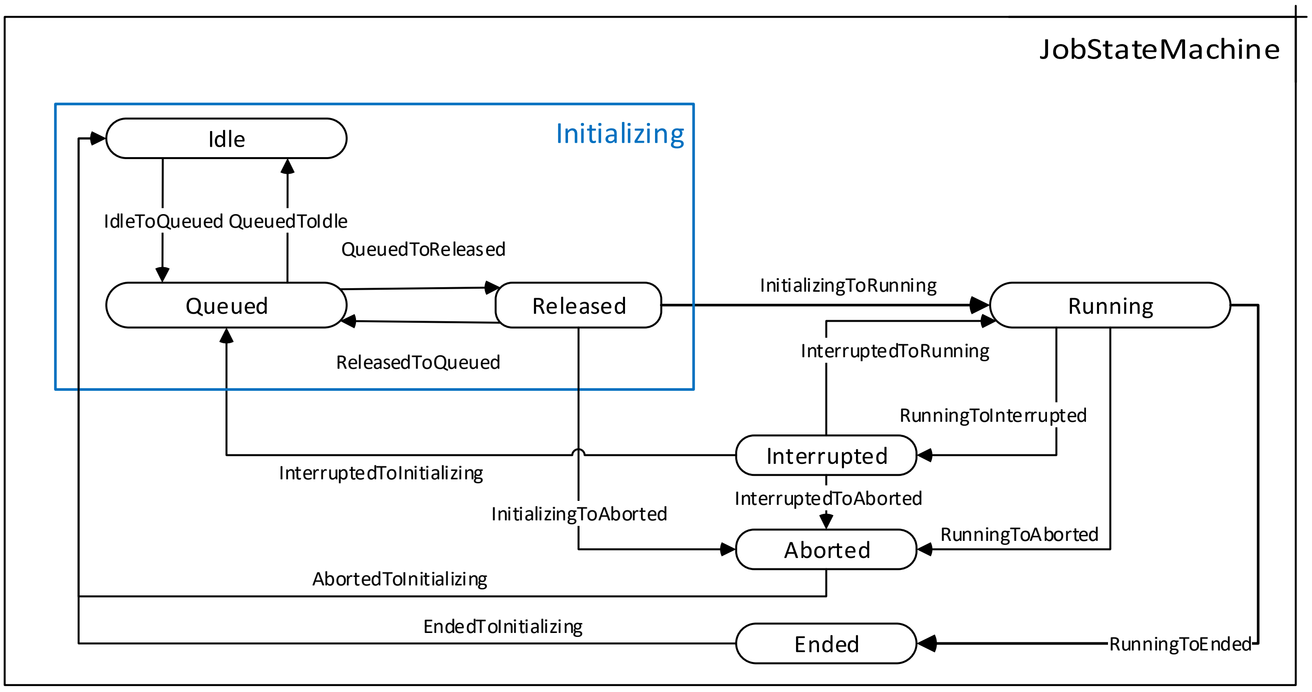


Figure 15 – JobStateMachine

The states shall have the numbers indicated in Table 36; the transitions shall have the numbers indicated in Table 36.

Initializing is the state in which the element in production is being prepared. During this state, the machine and the job doesn't have to be ready for production, although it has to be as soon as the transition InitializingToRunning is used. The production is not yet started. This state contains a sub-statemachine of the Type *InitializingSubStateMachineType.* Depending on the implementation, a glass machine starts to process the jobs which have been released for production either by itself or by an operator. The machine sets the status of the next released job in the queue to Running and processes it.

Running indicates that the operation of a job in production has been started or re-started and is currently running. There can be sub-statemachines in future versions of this standard or vendor-specific sub-statemachines.

Interrupted indicates that the execution of a job in production has been reversibly halted. This is usually due to an error or an intervention by the operating personnel. It is possible to restart operation of or on the element in production after it was in the interrupted state. From Interrupted you can switch back to the Running state by Continue or, if the processing is to be aborted with Abort to the State Aborted (both transitions are machine internal). Furthermore, an interrupted job can be brought back to the Queued state with the SuspendMethod called by an HLS.

The state *Aborted* indicates, that the operation of a job in production has been irreversibly stopped before finishing. In addition, an already scheduled job can be moved directly to the Aborted state with the Abort-Method from HLS if the job is not to be produced.

Ended is reached when the operation of a job in production has been finished. Jobs can only be removed in the states *Ended and Initializing* (only in sub-states *Idle* and *Queued*).

Table 36 – ProductionStateMachineType Attribute values for child Nodes

Source Path	Value Attribute	Description Attribute
State Numbers		
Initializing 0:StateNumber	0	
Running 0:StateNumber	1	
Ended 0:StateNumber	2	
Interrupted 0:StateNumber	3	
Aborted 0:StateNumber	4	
Transition Numbers		
InitializingToRunning 0:TransitionNumber	0	
RunningToEnded 0:TransitionNumber	1	
EndedToInitializing 0:TransitionNumber	2	
RunningToRunning 0:TransitionNumber	3	
RunningToInterrupted 0:TransitionNumber	4	
InterruptedToRunning 0:TransitionNumber	5	
RunningToAborted 0:TransitionNumber	6	
InterruptedToAborted 0:TransitionNumber	7	
AbortedToInitializing 0:TransitionNumber	8	
InitializingToAborted 0:TransitionNumber	9	

Fields may be empty which means this *Attribute* is not defined.

InitializingToRunning is triggered when the operation of a job in production starts.

RunningToEnded is triggered when the operation of a job in production finishes.

EndedToInitializing is triggered when re-initialization of the operation of a job in production starts.

RunningToRunning is triggered when another consecutive run of the operation of a job in production in direct succession starts.

RunningToInterrupted is triggered when the operation of a job in production is interrupted.

InterruptedToRunning is triggered when an interruption ends and the operation of a job in production continues running.

RunningToAborted is triggered when the operation of a job in production is aborted while in the Running state.

InterruptedToAborted is triggered when the operation of a job in production is aborted while in the Interrupted state. *AbortedToInitializing* is triggered if the operation of a job in production is being re-initialized after an abort.

InitializingToAborted is triggered when the operation of a job in production is aborted while in the *Initializing* state.

Table 37 – ProductionStateMachineType Additional References

Source Path	ReferenceType	Is Forward	Target Path
AbortedToInitializing	0:FromState	True	Aborted
	0:ToState	True	Initializing
EndedToInitializing	0:FromState	True	Ended
	0:ToState	True	Initializing
InitializingToAborted	0:FromState	True	Initializing
	0:ToState	True	Aborted
InitializingToRunning	0:FromState	True	Initializing
	0:ToState	True	Running
InterruptedToAborted	0:FromState	True	Interrupted
	0:ToState	True	Aborted
InterruptedToRunning	0:FromState	True	Interrupted
	0:ToState	True	Running
RunningToAborted	0:FromState	True	Running
	0:ToState	True	Aborted
RunningToEnded	0:FromState	True	Running
	0:ToState	True	Ended
RunningToInterrupted	0:FromState	True	Running
	0:ToState	True	Interrupted
RunningToRunning	0:FromState	True	Running
	0:ToState	True	Running

7.2.6 InitializingSubStateMachineType Definition

The InitializingSubStateMachineType provides more finely subdivided states (Idle, Queued and Released) to the Initialized state. Idle, are jobs which have been created but not yet scheduled. If the job is scheduled by an HLS in the queue of the glass machine, the state changes to Queued. To reach the Released state, which symbolizes the production release, an HLS must confirm the release by calling the ReleasedMethod. This SubStateMachine is formally defined in Table 38. The name of each transition consists of the names of the states it connects: [*FromState*]To[*ToState*]. Their *References* are specified in Table 40.

.

Table 38 – InitializingSubStateMachineType Definition

Attribute	Value				
BrowseName	InitializingSubStateMachineType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the FiniteStateMachineType defined in OPC UA 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	Idle		0:InitialStateType	None
0:HasComponent	Object	Queued		0:StateType	None
0:HasComponent	Object	Released		0:StateType	None
0:HasComponent	*Object*	*IdleToQueued*	*0:TransitionType*		
0:HasComponent	Object	QueuedToReleased	0:TransitionType		
0:HasComponent	Object	QueuedToIdle	0:TransitionType		
0:HasComponent	Object	ReleasedToQueued	0:TransitionType		

The states shall have the numbers indicated in Table 39; the transitions shall have the numbers indicated in Table 39.

Idle, are jobs that have been created but not yet scheduled.

If the job is scheduled by an HLS in the queue of the glass machine, the state changes to *Queued*. A job can only be modified in the states Idle and Queued.

To reach the *ReleasedState*, which symbolizes the production release, an HLS must confirm the release by calling the ReleasedMethod.

Table 39 – InitializingSubStateMachineType Attribute values for child Nodes

Source Path		Value Attribute	Description Attribute
State Numbers			
Idle	0:StateNumber	0	
Queued	0:StateNumber	1	
Released	0:StateNumber	2	
Transition Numbers			
IdleToQueued	0:TransitionNumber	0	
QueuedToReleased	0:TransitionNumber	1	
QueuedToIdle	0:TransitionNumber	2	
ReleasedToQueued	0:TransitionNumber	3	

Fields may be empty which means this *Attribute* is not defined.

IdleToQueued is triggered when the job is successfully added to the JobList.

QueuedToReleased is triggered when the job receives the final release for production.

QueuedToIdle is triggered when the job is need to changed and go to Idle.

ReleasedToQueued is triggered when a job is lose his release.

Table 40 – InitializingSubStateMachineType Additional References

Source Path	ReferenceType	Is Forward	Target Path
IdleToQueued	0:FromState	True	Idle
	0:ToState	True	Queued
QueuedToReleased	0:FromState	True	Queued
	0:ToState	True	Released
QueuedToIdle	0:FromState	True	Queued
	0:ToState	True	Idle
ReleasedToQueued	0:FromState	True	Released
	0:ToState	True	Queued

7.2.7 AssemblyJobType Definition

The AssemblyJobType provides the information for an assembly job, which assembles multiple input materials to one output material and is formally defined in Table 41.

Table 41 – AssemblyJobType Definition

Attribute	Value				
BrowseName	AssemblyJobType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the ProductionJobType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	<InputMaterial>		BaseMaterialType	MandatoryPlaceholder
0:HasComponent	Object	<OutputMaterial>		AssemblyType	MandatoryPlaceholder

<InputMaterial> describes the materials to be processed within the job. Can include MaterialBaseType or GlassType.

<OutputMaterial> describes the result after assembly from the materials side.

7.2.8 CuttingJobType Definition

The CuttingJobType provides the information for a cutting job, which cuts one sheet of glass into multiple smaller sheets and is formally defined in Table 42.

Table 42 – CuttingJobType Definition

Attribute	**Value**				
BrowseName	CuttingJobType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the ProductionJobType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	<InputMaterial>		BaseMaterialType	MandatoryPlaceholder
0:HasComponent	Object	<OutputMaterial>		GlassType	MandatoryPlaceholder

InputMaterial describes the materials to be processed within the job. Can include MaterialBaseType or GlassType.

OutputMaterial describes the result after cutting from the materials side.

7.2.9 ProcessingJobType Definition

The ProcessingJobType provides the information for a processing job and is formally defined in Table 43.

Table 43 – ProcessingJobType Definition

Attribute	**Value**				
BrowseName	ProcessingJobType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the ProductionJobType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	<InputMaterial>		BaseMaterialType	MandatoryPlaceholder
0:HasComponent	Object	<OutputMaterial>		BaseMaterialType	MandatoryPlaceholder

InputMaterial describes the materials to be processed within the job. Can include MaterialBaseType or GlassType.

OutputMaterial describes the result after processing from the materials side.

7.3 Material

7.3.1 BaseMaterialType Definition

The BaseMaterialType provides the basic material information and is formally defined in Table 44.

Table 44 – BaseMaterialType Definition

Attribute	**Value**				
BrowseName	BaseMaterialType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseObjectType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Variable	X	0:Double	0:AnalogUnitType	Optional
0:HasComponent	Variable	Y	0:Double	0:AnalogUnitType	Optional
0:HasComponent	Variable	Z	0:Double	0:AnalogUnitType	Optional
0:HasComponent	Variable	Weight	0:Double	0:AnalogUnitType	Optional
0:HasProperty	Variable	Description	0:LocalizedText	0:PropertyType	Optional
0:HasProperty	Variable	Location	LimitedString64	0:PropertyType	Mandatory
0:HasProperty	Variable	MaterialIdentifier	LimitedString64	0:PropertyType	Mandatory
0:HasProperty	Variable	Identifier	LimitedString64	0:PropertyType	Optional

Description is a human-readable description of the article.

Location defines source or destination location of the material, depending if it is an input or an output material.

MaterialIdentifier defines an Identifier to identify the Type of the item (item number).

Identifier defines a unique identifier for the specific component (e.g. the Serial number, Sheet-Id).

X defines the dimension of the material along the X axis of the machine coordinate system. For special shapes the surrounding rectangle is used.

Y defines the dimension of the material along the Y axis of the machine coordinate system. For special shapes the surrounding rectangle is used.

Z defines the dimension of the material along the Z axis of the machine coordinate system. For special shapes the surrounding rectangle is used.

Note: X,Y,Z are modelled separate, because subtype may only use a subset of X;Y,Z.

Weight defines the mass of the item. If possible the engineering unit should be kilograms.

7.3.2 FoilType Definition

The FoilType provides the information for foils used in glass technology and is formally defined in Table 45.

Table 45 – FoilType Definition

Attribute	**Value**				
BrowseName	FoilType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseMaterialType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Variable	Z	0:Double	0:AnalogUnitType	Mandatory

Z defines the thickness of the foil.

7.3.3 SpacerType Definition

The SpacerType provides the information for spacers used in glass technology and is formally defined in Table 46.

Table 46 – SpacerType Definition

Attribute	**Value**				
BrowseName	SpacerType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseMaterialType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Variable	Filling	LimitedString64	BaseDataVariableType	Optional
0:HasComponent	Variable	SealantDepth	0:Double	0:AnalogUnitType	Optional
0:HasProperty	Variable	SpacerMaterialClass	SpacerMaterialClass	0:PropertyType	Mandatory
0:HasProperty	Variable	SpacerMaterialSubClass	LimitedString64	0:PropertyType	Optional

Filling provides information about the filling of the spacer.

SealantDepth is the minimum dimension from the spacer to the outer edge of the silicone secondary seal (see section 3.3.5).

SpacerMaterialClass defines with an enumeration the material type of the spacer, e.g. metal, plastic etc.

SpacerMaterialSubClass provides a vendor-specific unique identification of the spacer material.

7.3.4 GasMixType Definition

The GasMixType provides information about the gas mix, which is used in glass technology assembly and is formally defined in Table 47.

Table 47 – GasMixType Definition

Attribute	Value				
BrowseName	GasMixType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseMaterialType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	Gas_1		BaseMaterialType	Optional
0:HasComponent	Object	Gas_2		BaseMaterialType	Optional
0:HasComponent	Variable	GasFilling	0:Number	0:AnalogUnitType	Optional
0:HasComponent	Variable	MixingRatio	0:Number	0:AnalogUnitType	Optional

Gas_1 uniquely describes the first gas, which is part of the gas filling. The manufacturer and the material designation should be included.

Gas_2 uniquely describes the second gas, which is part of the gas filling. The manufacturer and the material designation should be included.

GasFilling defines the filling level of the gas in percent, for a filling of 100% the values is 100. For this variable are also values of more than 100 allowed, to represent an overpressure in the final product.

MixingRatio describes the ratio of the gas mixture. Here the value 0 means that only the first gas (Gas_1) is contained. If the parameter is 100 only second gas (Gas_2) is in the mixture. With a value of 50 both gases are contained in the ratio 1:1.

The component *Variables* of the *GasMixType* have additional *Attributes* defined in Table 48.

Table 48 – GasMixType Attribute values for child Nodes

BrowsePath	Value Attribute	Description Attribute
MixingRatio 0:EngineeringUnits	NamespaceUri: http://www.opcfoundation.org/UA/units/un/cefact UnitId: 20529 DisplayName: % Description: percent	
GasFilling 0:EngineeringUnits	NamespaceUri: http://www.opcfoundation.org/UA/units/un/cefact UnitId: 20529 DisplayName: % Description: percent	

7.3.5 GlassTypeDefinition

The GlassType provides information about flat glass sheets and is formally defined in Table 49.

Table 49 – GlassType Definition

Attribute	**Value**				
BrowseName	GlassType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseMaterialType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Variable	Absorption	0:Number	0:AnalogUnitType	Optional
0:HasProperty	Variable	CoatingClass	CoatingClassEnumeration	0:PropertyType	Mandatory
0:HasComponent	Variable	CoatingEmessivity	0:Number	0:AnalogUnitType	Optional
0:HasProperty	Variable	CoatingSubClass	LimitedString64	0:PropertyType	Optional
0:HasComponent	Variable	ElectricalConductivity	0:Number	0:AnalogUnitType	Optional
0:HasProperty	Variable	Orientation	0:Number	0:PropertyType	Mandatory
0:HasComponent	Variable	Reflection	0:Number	0:AnalogUnitType	Optional
0:HasProperty	Variable	SignificantSide	SignificantSideEnumeration	0:PropertyType	Mandatory
0:HasProperty	Variable	StructureClass	LimitedString64	0:PropertyType	Mandatory
0:HasProperty	Variable	StructureAlignment	StructureAlignmentEnumeration	0:PropertyType	Mandatory
0:HasComponent	Variable	Transmission	0:Number	0:AnalogUnitType	Optional
The following nodes are override from BaseMaterialType and the Modelling Rules change to Mandatory					
0:HasComponent	Variable	X	0:Double	0:AnalogUnitType	Mandatory
0:HasComponent	Variable	Y	0:Double	0:AnalogUnitType	Mandatory

Absorption describes the absorption of the raw material.

CoatingClass defines which coatings have been applied to the SignificantSide.

CoatingEmessivity describes the emissivity of the coating.

CoatingSubClass provides a vendor-specific unique identification of the coating material.

ElectricalConductivity describes the conductivity of the raw material in Siemens.

Orientation describes the angle of the material coordinates in reference to the MachineProcessingCoordinateSystem.

Reflection describes the reflection of the raw material.

StructureClass defines a vendor-specific description of the glass structure (surface topology). An example for that is Chinchilla white.

SignificantSide specifies whether the significant (coating) side (see definition in Terms) should be at the top or bottom during machining (see also section 3.2.6 and 0).

StructureAlignment specifies how the surface structure is aligned. See StructureAlignmentEnumeration for more information e.g. about the orientation.

Transmission describes the transmission of the raw material.

7.3.6 PackagingType Definition

The PackagingType provides information about the packing material used in glass technology and is formally defined in Table 50.

Table 50 – PackagingType Definition

Attribute	**Value**				
BrowseName	PackagingType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseMaterialType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Variable	PerimeterProtection	LimitedString64	BaseDataVariableType	Mandatory
0:HasComponent	Variable	CornerProtection	LimitedString64	BaseDataVariableType	Mandatory
0:HasComponent	Variable	Spacer	LimitedString64	BaseDataVariableType	Mandatory

PerimeterProtection describes the protection of the wet glue for better handling of assembly units.

CornerProtection specifies the materials which will be used to protect the corners so that will not get damaged during transportation.

Spacer describes which separation materials are used to separate multiple units within the package.

7.3.7 SealingMaterialType Definition

The SealingMaterialType provides the information about the sealing materials used in glass technology assemblies and is formally defined Table 51.

Table 51 – SealingMaterialType Definition

Attribute	**Value**				
BrowseName	SealingMaterialType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseMaterialType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasComponent	Object	Hardener		BaseMaterialType	Optional
0:HasComponent	Variable	MixingRatio	0:Double	AnalogUnitType	Mandatory
0:HasComponent	Object	Resin		BaseMaterialType	Optional
0:HasComponent	Variable	AddOnMaterial	LimitedString64	BaseDataVariableType	Optional

Hardener uniquely describes the second sealing material, which is part of the sealing mixture. The manufacturer and the material designation should be included.

MixingRatio describes the ratio of the sealing material mixture. Here the value 0 means that only the first material (*Resin*) is contained. If the parameter is 100 only *Hardener* is contained.

Resin uniquely describes the first sealing material, which is part of the sealing mixture. The manufacturer and the material designation should be included.

AddOnMaterial describes a material that is added to the sealing e.g. to increases the stiffness.

The component *Variables* of the SealingMaterialType have additional *Attributes* defined in Table 52.

Table 52 – SealingMaterialType Attribute values for child Nodes

BrowsePath	**Value Attribute**	**Description Attribute**
MixingRatio 0:EngineeringUnits	NamespaceUri: http://www.opcfoundation.org/UA/units/un/cefact UnitId: 20529 DisplayName: % Description: percent	

7.3.8 AssemblyType Definition

The AssemblyType provides the information about the assembled glass product and is formally defined in Table 53.

Table 53 – AssemblyType Definition

Attribute	**Value**				
BrowseName	AssemblyType				
IsAbstract	false				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseMaterialType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8 OPC UA Event Types

8.1 GlassEventType ObjectType definition

The GlassEventType provides the additional information for an event sent by a glass machine and is formally defined in Table 54. All Event of this Companion Specification will inherit form the GlassEventType.

Table 54 – GlassEventType Definition

Attribute	**Value**				
BrowseName	GlassEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the BaseEventType defined in OPC 10000-5, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasProperty	Variable	JobdIdentifier	LimitedString64	0:PropertyType	Optional
0:HasProperty	Variable	Location	LimitedString64	0:PropertyType	Optional
0:HasProperty	Variable	MaterialIdentifier	LimitedString64	0:PropertyType	Optional
0:HasProperty	Variable	*Identifier*	LimitedString64	0:PropertyType	Optional

JobdIdentifier defines the job identification of the job related to this event.

Location defines the target/source location of the material which is provided.

MaterialIdentifier defines the material identification of the (missing) material which is provided.

Identifier defines a unique identifier for the specific component (e.g. the Serial number, Sheet-Id).

8.2 CommunicationErrorEventType Definition

The CommunicationErrorEventType is used to classify conditions for a communication error and is formally defined in Table 55.

Table 55 – CommunicationErrorEventType Definition

Attribute	**Value**				
BrowseName	CommunicationErrorEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8.3 GlassMaterialEventType Definition

The GlassMaterialEventType is used to send events related to a material (e.g. an input material is recived) and is formally defined in Table 56.

Table 56 – GlassMaterialEventType Definition

Attribute	**Value**				
BrowseName	GlassMaterialEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassEventType defined in this Companion Specification , i.e. inheriting the InstanceDeclarations of that Node.					

8.4 MaterialReceivedEventType EventType Definition

The MaterialReceivedEventType is used to classify conditions, where the input material is successfully received from the glass technology machine and is formally defined in Table 57.

Table 57 – MaterialReceivedEventType Definition

Attribute	**Value**				
BrowseName	MaterialReceivedEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassMaterialEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8.5 MaterialMissingEventType EventType Definition

The MaterialMissingEventType is used to classify conditions, where the input material is not successfully received by the glass technology machine and is formally defined in Table 58.

Table 58 – MaterialMissingEventType Definition

Attribute	**Value**				
BrowseName	MaterialMissingEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassMaterialEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8.6 MaterialExitEventType EventType Definition

The MaterialExitEventType is used to classify conditions, where the output material is successfully delivered by the glass technology machine and is formally defined in Table 59.

Table 59 – MaterialExitEventType Definition

Attribute	**Value**				
BrowseName	MaterialExitEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassMaterialEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8.7 IntermediateStepEvent EventType Definition

The IntermediateStepEvent is used to classify conditions, where an intermediate step in the production sequence is successfully processed by the glass technology machine and is formally defined in Table 60.

Table 60 – IntermediateStepEvent Definition

Attribute	**Value**				
BrowseName	IntermediateStepEvent				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasProperty	Variable	ProcessStep	LimitedString64	0:PropertyType	Optional
0:HasProperty	Variable	Status	LimitedString64	0:PropertyType	Optional

ProcessStep defines the name of the current process running on the machine while this event is sent.

Status defines a vendor specific status of the current process. This can be a status like “washing” “waiting on X” or running, for example.

8.8 InterruptedEventType EventType Definition

The InterruptedEventType is used to classify conditions, where the processing of a glass technology machine is interrupted and is formally defined in Table 61.

Table 61 – InterruptedEventType Definition

Attribute	**Value**				
BrowseName	InterruptedEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasProperty	Variable	ProcessName	LimitedString64	0:PropertyType	Optional

ProcessName is a hint which process was interrupted.

8.9 OutOfJobEventType EventType Definition

The OutOfJobEventType is used to classify conditions, where the ProductionJobList of a glass technology machine has no processable Job on top of the list and is formally defined in Table 62.

Table 62 – OutOfJobEventType Definition

Attribute	**Value**				
BrowseName	OutOfJobEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the InterruptedEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8.10 ProcessParameterOutOfRangeType EventType Definition

The ProcessParameterOutOfRangeType is used to classify conditions, where a process parameter of a glass technology machine is out of range and is formally defined in Table 63.

Table 63 – ProcessParameterOutOfRangeType Definition

Attribute	**Value**				
BrowseName	ProcessParameterOutOfRangeType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the InterruptedEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8.11 ToolMissingEventType EventType Definition

The ToolMissingEventType is used to classify conditions, where a tool required for the processing of a glass technology machine is missing and is formally defined in Table 64.

Table 64 – ToolMissingEventType Definition

Attribute	**Value**				
BrowseName	ToolMissingEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the InterruptedEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					

8.12 JobMovedEventType EventType Definition

The JobMovedEventType is used to classify conditions, where the position of a Job is moved inside a ProductionJobList of a glass technology machine and is formally defined in Table 65.

Table 65 – JobMovedEventType Definition

Attribute	**Value**				
BrowseName	JobMovedEventType				
IsAbstract	true				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the GlassEventType defined in this Companion Specification, i.e. inheriting the InstanceDeclarations of that Node.					
0:HasProperty	Variable	JobdIdentifier		0:PropertyType	Mandatory
0:HasProperty	Variable	NewPosition	0:Number	0:PropertyType	Optional

NewPosition contains the new position number in the JobList

9 OPC UA DataTypes

9.1 UserProfileType

This structure contains the information about a logged in user with his profile data. The structure is defined inTable 66.

Table 66 – UserProfileType Structure

Name	Type	Description
UserProfileType	Structure	Subtype of Struct defined in OPC UA 10000-5
Name	0:String	Human-readable name of the user profile.
LoginTime	0:DateTime	Date and Time in UTC this profile is logged in.
Language	0:LocaleId	The Language that is configured for the user
MeasurementFormat	0:String	Defines which system of measurement (collection of units of measurement) is used. Use “SI” for the International System of Units.
AccessLevel	0:String	Describes the access right the user Profile has. This is a human readable string and should be in English language, e.g. “Administrator”
OpcUaUser	0:Boolean	This flag is true if the user is logged in via OPC UA. If it is false the user is logged in another way e.g. as local user.

Its representation in the *AddressSpace* is defined in Table 67.

Table 67 – UserProfileType Definition

Attribute	**Value**				
BrowseName	UserProfileType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of Structure defined in OPC UA 10000-3					

9.2 FileFormatType

A file format describes the syntax and semantic of a document. This structure contains the information about a file format. The structure is defined in Table 68. If there are different version allowed all must be added.

Table 68 – FileFormatType Structure

Name	Type	Description
FileFormatType	Structure	Subtype of Struct defined in OPC UA 10000-5
Name	0:String	The Name of the FileFormat. The following strings are examples for a FileFormat Name: “Edicut”, ”Lisec.TRF”, ”Lenhardt”
FileExtension	0:String	is the identifier specified as a suffix to the name of a file. The FileExtention has a leading dot. So, the FileExtension should be look like “.nc”,”.json”,”.edi”
Version	0:String	Version of the FileFormat. Syntax is major.minor[.build] (example *2.5 or 5.9.2)*

Its representation in the *AddressSpace* is defined in Table 69.

Table 69 – FileFormatType Definition

Attribute	**Value**				
BrowseName	FileFormatType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of Structure defined in OPC UA 10000-3					

9.3 SpacerMaterialClass

This enumeration describes the main class of the SpacerMaterial. The values represent different spacer materials e.g. metal, plastic or additional framing materials like elastic spacer. The enumeration is defined in Table 70.

Table 70 – SpacerMaterialClass Items

Name	Value	Description
Other	0	All other materials that are not in other classes.
Metalic	1	The value is used if the spacer material is metalic.
TPS	2	The value is used if the spacer material is TPS.
Plastic	3	The value is used if the spacer material is plastic.
Elastic	4	The value is used if the spacer material is elastic.

Its representation in the *AddressSpace* is defined in Table 71.

Table 71 – SpacerMaterialClass Definition

Attribute	Value				
BrowseName	SpacerMaterialClass				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Enumeration type defined in OPC 10000-5					
0:HasProperty	Variable	0:EnumStrings	0:LocalizedText[]	0:PropertyType	

9.4 SignificantSideEnumeration

This enumeration describes whether the significant side of a sheet should be top or bottom. Additionally, there is the indifferent option, where the orientation does not matter. The enumeration is defined in Table 72.

Table 72 – SignificantSideEnumeration Items

Name	Value	Description
Indifferent	0	There is no significant Side or it do not matter if is top or bottom
Top	1	Defined that the significant side is on Top. That means the significant side is on the opposite side than the machine table or the transport system
Down	2	Defined that the significant side is on Bottom. That means the significant side is on the side of the machine table or the transport system

Its representation in the *AddressSpace* is defined in Table 73.

Table 73 – SignificantSideEnumeration Definition

Attribute	Value				
BrowseName	SignificantSideEnumeration				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Enumeration type defined in OPC 10000-5					
0:HasProperty	Variable	0:EnumStrings	0:LocalizedText[]	0:PropertyType	

9.5 StructureAlignmentEnumeration

This enumeration specifies how the structure of the surface is aligned within the direction of the material flow. In addition to longitudinal and transversal alignment, it is also possible to leave the option indifferent. The enumeration is defined in Table 74.

Table 74 – StructureAlignmentEnumeration Items

Name	Value	Description
Indifferent	0	The direction of the alignment can be ignored
Longitudinal	1	The direction of the alignment is along the material flow axis
Transverse	2	The direction of the alignment is across the body length axis

Its representation in the *AddressSpace* is defined in Table 75.

Table 75 – StructureAlignmentEnumeration Definition

Attribute	Value				
BrowseName	StructureAlignmentEnumeration				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Enumeration type defined in OPC 10000-5					
0:HasProperty	Variable	0:EnumStrings	0:LocalizedText[]	0:PropertyType	

9.6 CoatingClassEnumeration

This enumeration specifies the different coating classes. The options are hard and soft coating as well as coating with protection foil. In addition, a user-defined selection (UD) with further description in the coating subclass is possible. The enumeration is defined in Table 76.

Table 76 – CoatingClassEnumeration Items

Name	Value	Description
HardCoating	0	This describes a type of coating or covering a glass surface in a way that the resulting surface is rather resistant against damage, at least at a similar level as compared to standard Floatglass. In a composite, the surface may be exposed to the environment.
SoftCoating	1	This describes a type of coating covering a glass surface resulting in a surface that is more vulnerable by environmental conditions such as moist, dirt etc. Soft coated glass panes require more care in processing.
CoatedWithFoilProtection	2	This describes glass panes where the coating (typically a soft coating) is protected against environmental influence by a foil that might have to be removed when the glass is to be processed.
UserDefined	3	This describes all other cases with further description provided in the coatingsubclass.

Its representation in the *AddressSpace* is defined in Table 77.

Table 77 – CoatingClassEnumeration Definition

Attribute	Value				
BrowseName	CoatingClassEnumeration				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Enumeration type defined in OPC 10000-5					
0:HasProperty	Variable	0:EnumStrings	0:LocalizedText[]	0:PropertyType	

9.7 LimitedString64

This DataType specifies a String, which is limited to 64 chars. Its representation in the *AddressSpace* is defined in Table 78.

Table 78 – LimitedString64 Definition

Attribute	Value				
BrowseName	LimitedString64				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the String type defined in OPC 10000-5					

9.8 CoordinateSystemEnumeration

This enumeration specifies the different options in placing the machine processing coordinate systems. The eight different options are shown in Figure 16, as well as some examples in Figure 17 and Figure 18.

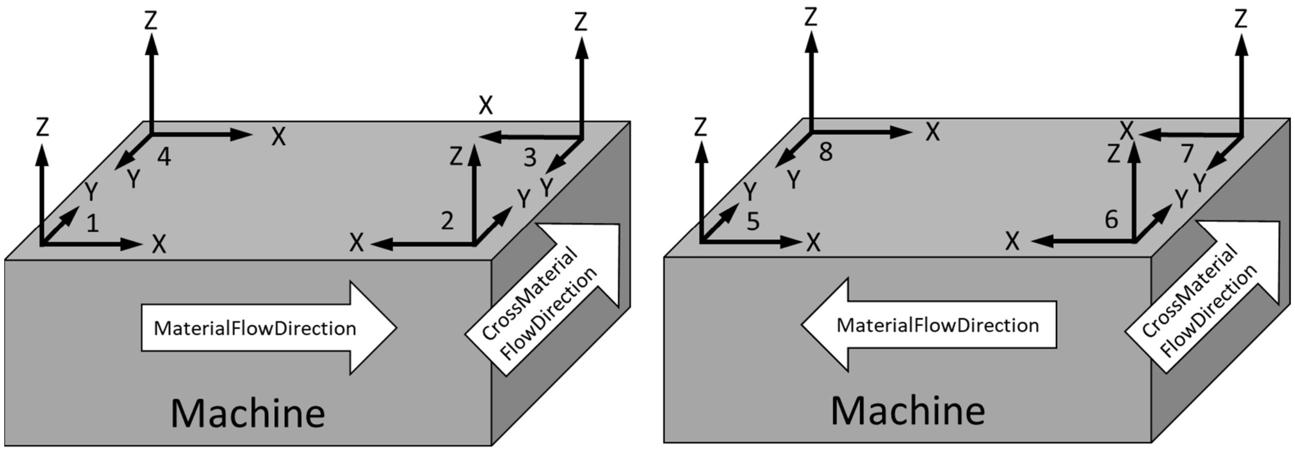


Figure 16 – Machine processing coordinate systems

The enumeration is defined in Table 79.

Table 79 – CoordinateSystemEnumeration Items

Name	Value	Description
Unknown	0	The CoordinateSystem is unknown.
System1	1	Lefthanded system, X-axis along material flow, Y-axis along cross material flow
System2	2	Righthanded system, X-axis against material flow, Y-axis along cross material flow
System3	3	Lefthanded system, X-axis against material flow, Y-axis against cross material flow
System4	4	Righthanded system, X-axis along material flow, Y-axis against cross material flow
System5	5	Lefthanded system, X-axis against material flow, Y-axis along cross material flow
System6	6	Righthanded system, X-axis along material flow, Y-axis along cross material flow
System7	7	Lefthanded system, X-axis along material flow, Y-axis against cross material flow
System8	8	Righthanded system, X-axis against material flow, Y-axis against cross material flow

Its representation in the *AddressSpace* is defined in Table 80.

Table 80 – CoordinateSystemEnumeration Definition

Attribute	Value				
BrowseName	CoordinateSystemEnumeration				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Other**
Subtype of the Enumeration type defined in OPC 10000-5					
0:HasProperty	Variable	0:EnumStrings	0:LocalizedText[]	0:PropertyType	

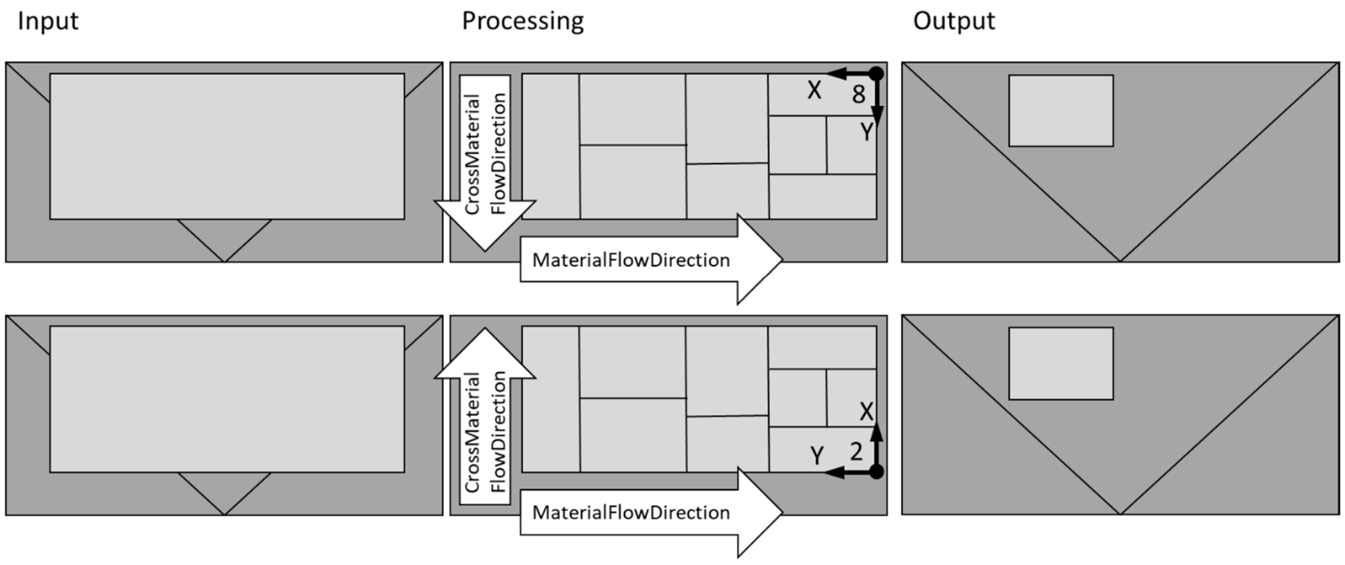


Figure 17 – Example of glass processing machines for flat lying processing (top view)

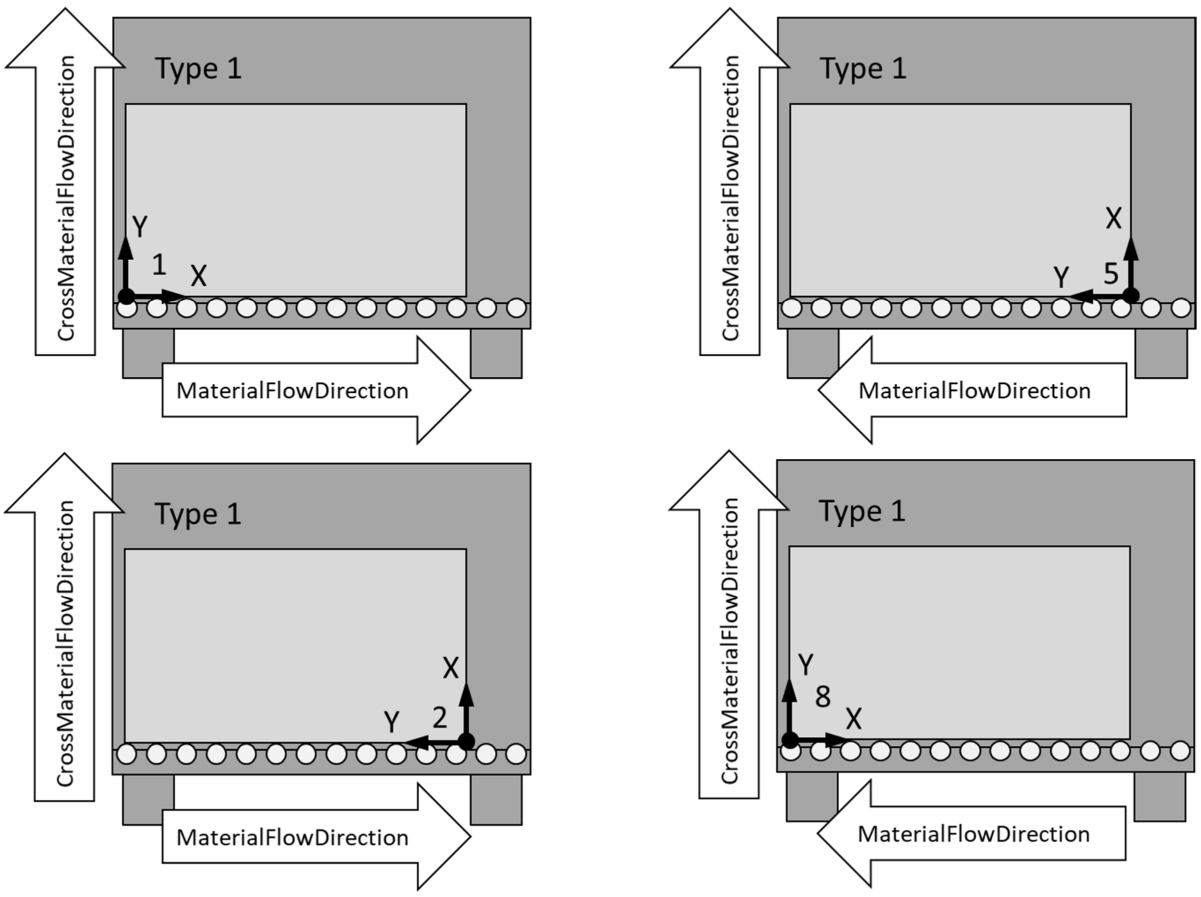


Figure 18 – Example of glass processing machines with standing processing (front view)

10 Profiles and ConformanceUnits

10.1 Conformance Units

This chapter defines the corresponding *Conformance Units* for the OPC UA Information Model for glass technology.

Table 81 – Conformance Units for glass technology

Category	Title	Description
Server	FlatGlass_GlassMachineType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) GlassMachineType. This node has to include all mandatory components of the GlassMachineType and may include the optional components. The instance(s) of the (subtypes of)[SomeType] is/are available in the AddressSpace Machines Folder in Model.
Server	FlatGlass_GlassMachineIdentificationType	The GlassMachineIdentificationType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) MachineIdentificationType. This node has to include all mandatory components of the MachineIdentificationType and may include the optional components.
Server	FlatGlass_ManualFolderType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) ManualFolderType. This node has to include all mandatory components of the ManualFolderType and may include the optional components.
Server	FlatGlass_ProductionType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) ProductionType. This node has to include all mandatory components of the ProductionType and may include the optional components.
Server	FlatGlass_ProductionJobType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) ProductionJobType. This node has to include all mandatory components of the ProductionJobType and may include the optional components.
Server	FlatGlass_ProductionJobListType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) ProductionJobListType. This node has to include all mandatory components of the ProductionJobListType and may include the optional components.
Server	FlatGlass_InstructionType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) InstructionType. This node has to include all mandatory components of the InstructionType and may include the optional components.
Server	FlatGlass_ProductionJobType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) ProductionJobType. This node has to include all mandatory components of the ProductionJobType and may include the optional components.
Server	FlatGlass_AssemblyJobType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) AssemblyJobType. This node has to include all mandatory components of the AssemblyJobType and may include the optional components.
Server	FlatGlass_CuttingJobType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) CuttingJobType. This node has to include all mandatory components of the CuttingJobType and may include the optional components.
Server	FlatGlass_ProcessingJobType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) ProcessingJobType. This node has to include all mandatory components of the ProcessingJobType and may include the optional components.

Category	**Title**	**Description**
Server	FlatGlass_BaseMaterialType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) BaseMaterialType. This node has to include all mandatory components of the BaseMaterialType and may include the optional components.
Server	FlatGlass_FoilType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) FoilType. This node has to include all mandatory components of the FoilType and may include the optional components.
Server	FlatGlass_SpacerType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) SpacerType. This node has to include all mandatory components of the SpacerType and may include the optional components.
Server	FlatGlass_GasMixType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) GasMixType. This node has to include all mandatory components of the GasMixType and may include the optional components.
Server	FlatGlass_GasType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) GasType. This node has to include all mandatory components of the GasType and may include the optional components.
Server	FlatGlass_PackagingType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) PackagingType. This node has to include all mandatory components of the PackagingType and may include the optional components.
Server	FlatGlass_SealingMaterialType	The GlassMachineType node is available in the AddressSpace. Supports nodes that conform to the (subtypes of) SealingMaterialType. This node has to include all mandatory components of the SealingMaterialType and may include the optional components.
Server	FlatGlass_ProductionStateMachineType	The ProductionStateMachineType node is available in the AddressSpace. The server supports nodes that conform to the (subtypes of) ProductionStateMachineType.The ProductionStateMachineType state machine is implemented correctly by the server. That means the succession of states adheres to the transitions defined in this specification and the hasCause and hasEffect References are implemented correctly.
Server	FlatGlass_ InitializingSubStateMachineType	The InitializingSubStateMachineType node is available in the AddressSpace. The server supports nodes that conform to the (subtypes of) InitializingSubStateMachineType. The instance(s) of the (subtypes of) InitializingSubStateMachineType is/are available in the AddressSpace as sub-statemachine of the ProductionStateMachineType in the model. The InitializingSubStateMachineType state machine is implemented correctly by the server. That means the succession of states adheres to the transitions defined in this specification and the hasCause and hasEffect References are implemented correctly.
Server	FlatGlass_GlassEventType	The [GlassEventType] node is available in the AddressSpace. Events of the (sub-types of) [GlassEventType] are generated by the server.
Server	FlatGlass_CommunicationErrorEventType	The [CommunicationErrorEventType] node is available in the AddressSpace. Events of the (sub-types of) [CommunicationErrorEventType] are generated by the server.
Server	FlatGlass_MaterialReceivedEventType	The [MaterialReceivedEventType] node is available in the AddressSpace. Events of the (sub-types of) [MaterialReceivedEventType] are generated by the server.
Server	FlatGlass_MaterialMissingEventType	The [MaterialMissingEventType] node is available in the AddressSpace. Events of the (sub-types of) [MaterialMissingEventType] are generated by the server.
Server	FlatGlass_MaterialExitEventType	The [MaterialExitEventType] node is available in the AddressSpace. Events of the (sub-types of) [MaterialExitEventType] are generated by the server.
Server	FlatGlass_IntermediateStepEvent	The [IntermediateStepEvent] node is available in the AddressSpace. Events of the (sub-types of) [IntermediateStepEvent] are generated by the server.

Category	Title	Description
Server	FlatGlass_InterruptedEventType	The [InterruptedEventType] node is available in the AddressSpace. Events of the (sub-types of) [InterruptedEventType] are generated by the server.
Server	FlatGlass_OutOfJobEventType	The [OutOfJobEventType] node is available in the AddressSpace. Events of the (sub-types of) [OutOfJobEventType] are generated by the server.
Server	FlatGlass_ProcessParameterOutOfRangeType	The [ProcessParameterOutOfRangeType] node is available in the AddressSpace. Events of the (sub-types of) [ProcessParameterOutOfRangeType] are generated by the server.
Server	FlatGlass_ToolMissingEventType	The [ToolMissingEventType] node is available in the AddressSpace. Events of the (sub-types of) [ToolMissingEventType] are generated by the server.
Server	FlatGlass_JobMovedEventType	The [JobMovedEventType] node is available in the AddressSpace. Events of the (sub-types of) [JobMovedEventType] are generated by the server.
Server	FlatGlass_ChangePositionInList_Method	Supports the handling of the method ChangePositionInList.
Server	FlatGlass_ReleaseJob_Method	Supports the handling of the method ReleaseJob.
Server	FlatGlass_SuspendJob_Method	Supports the handling of the method SuspendJob.
Server	FlatGlass_InsertJob_Method	Supports the handling of the method InsertJob.
Server	FlatGlass_DeleteJob_Method	Supports the handling of the method DeleteJob.
Server	FlatGlass_QueuedJob_Method	Supports the handling of the method QueueJob.

10.2 Profiles

10.2.1 Profile list

Table 82 lists all Profiles defined in this document and defines their URIs.

Table 82 – Profile URIs for glass technology

Profile	URI
Glass Server Base Profile	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Server Base Profile
Glass Identification Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Identification Facet
Glass Minimal Production Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Minimal Production Facet
Glass Changeable JobList Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Changeable JobList Facet
Glass Dynamic Production Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Dynamic Production
Glass Job Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Job Facet
Glass Processing Job Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Processing Job Facet
Glass Assembly Job Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Assembly Job Facet
Glass Cutting Job Facet Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Cutting Job Facet
Glass Event Server Facet	http://opcfoundation.org/UA-Profile/Glass/Flat/Server/Event Facet

10.2.2 Server Facets

10.2.2.1 Overview

The following sections specify the *Facets* available for *Servers* that implement the glass technology companion specification. Each section defines and describes a *Facet* or *Profile*.

An OPC UA Server that implements this Companion Specification needs to implement the Glass Base Server Profile (including the Facets Glass Identification Server Facet and Glass Minimal Production Facet). Optional extensions are the Glass Event Server Facet, the Glass Dynamic Production Server Facet and the Changeable JobList Facet. One of the three Job Facets needs to be implemented (Processing, Cutting, Assembly). The following figure shows the structure of the base server profile and facets. The mandatory Profiles are grey, optionally are green and the yellow Profiles are optional and describe the production process. Mostly, only one of the yellow Facets is needed.

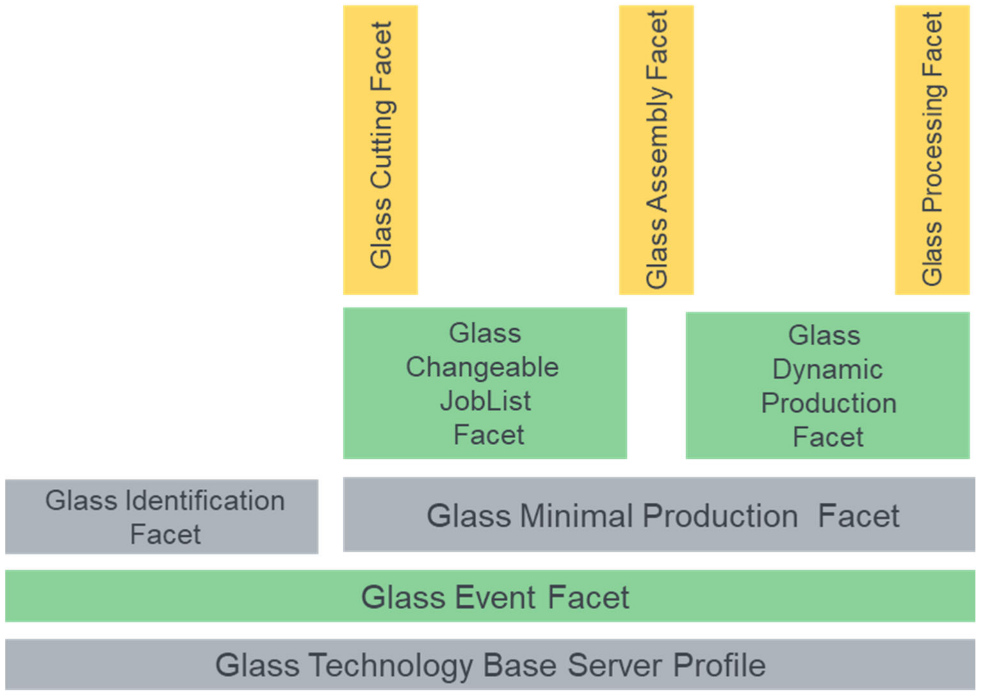

Glass Cutting Facet
Glass Assembly Facet
Glass Processing Facet
Glass Changeable JobList Facet
Glass Dynamic Production Facet
Glass Identification Facet
Glass Minimal Production Facet
Glass Event Facet
Glass Technology Base Server Profile

Figure 19 – Glass Technology Profiles and Facets

10.2.2.2 Glass Base Server Profile and Facets

Table 83 defines a *Profile* that describes the minimum requirements for the implementation of a Glass Technology OPC UA server.

Table 83 – Glass Base Server Profile

Group	Conformance Unit / Profile Title	Mandatory / Optional
Profile	0:Nano Embedded Device 2017 Server Profile http://opcfoundation.org/UA-Profile/Server/NanoEmbeddedDevice2017	M
Profile	2:BaseDevice_Server_Facet	M
Profile	0:Data Access Server Facet http://opcfoundation.org/UA-Profile/Server/DataAccess	O
Flat Glass	GlassMachineType	M
Flat Glass	Glass Identification Facet	M
Flat Glass	Glass Minimal Production Facet	M
Flat Glass	Glass Dynamic Production Facet	O
Flat Glass	Glass Changeable JobList Server Facet	O
Flat Glass	Glass Event Server Facet	O

10.2.2.3 Glass Identification Facet

Table 84 defines a *Facet* for the identification of glass technology machines, which requires the InstructionType and MachineIdentificationType as mandatory.

Table 84 – Glass Identification Server Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	FlatGlass_InstructionType	M
Flat Glass	FlatGlass_GlassMachineIdentificationType	M
Machinery	3:Machinery Machine Identification Server Facet	M

10.2.2.4 Glass Minimal Production Facet

Table 85 defines the Minimal Production Facet, that needs the ProductionStateMachineType and the InitializingSubStateMachineType to provide the actual production state.

Table 85 – Glass Minimal Production Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	FlatGlass_ProductionType	M
Flat Glass	FlatGlass_ProductionStateMachineType	M
Flat Glass	FlatGlass_InitializingSubStateMachineType	M

10.2.2.5 Glass Changeable JobList Facet

Table 86 defines a *Facet* called Changeable JobList Facet that allows to change the production sequence of the jobs.

Table 86 – Glass Changeable JobList Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	FlatGlass_ChangePositionInList	M

10.2.2.6 Glass Dynamic Production Facet

Table 87 defines the Dynamic Production Facet that supports the full model specification for job handling, including inserting and deleting, releasing and suspending, as well as modifying.

Table 87 – Glass Dynamic Production Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	FlatGlass_Minimal Production Facet	M
Flat Glass	FlatGlass_ReleaseJob_Method	M
Flat Glass	FlatGlass_SuspendJob_Method	M
Flat Glass	FlatGlass_InsertJob_Method	M
Flat Glass	FlatGlass_Delete Job_Method	M
Flat Glass	FlatGlass_Queue Job_Method	M
Flat Glass	Device_Lock_Service_Facet	M

10.2.2.7 Glass Assembly Job Facet

Table 88 defines a *Facet* that describes the mandatory Types for assembly jobs.

Table 88 – Glass Assembly Job Server Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	FlatGlass_AssemblyJobType	M
Flat Glass	FlatGlass_BaseMaterialType	M
Flat Glass	FlatGlass_GlassType	O
Flat Glass	FlatGlass_FoilType	O
Flat Glass	FlatGlass_SpacerType	O
Flat Glass	FlatGlass_GasMixType	O
Flat Glass	FlatGlass_GlassType	O
Flat Glass	FlatGlass_PackagingType	O
Flat Glass	FlatGlass_SealingMaterialType	O

10.2.2.8 Glass Cutting Job Facet

Table 89 defines a *Facet* that describes the mandatory Types for cutting jobs.

Table 89 – Glass Cutting Job Server Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	FlatGlass_CuttingJobType	M
Flat Glass	FlatGlass_BaseMaterialType	M
Flat Glass	FlatGlass_GlassType	M
Flat Glass	FlatGlass_FoilType	O
Flat Glass	FlatGlass_SpacerType	O

10.2.2.9 Glass Processing Job Facet

Table 90 defines a *Facet* that describes the mandatory Types for processing jobs.

Table 90 – Glass Processing Job Server Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	FlatGlass_ProcessingJobType	M
Flat Glass	FlatGlass_BaseMaterialType	M
Flat Glass	FlatGlass_GlassType	O
Flat Glass	FlatGlass_FoilType	O
Flat Glass	FlatGlass_SpacerType	O
Flat Glass	FlatGlass_GasMixType	O
Flat Glass	FlatGlass_GasType	O
Flat Glass	FlatGlass_PackagingType	O
Flat Glass	FlatGlass_SealingMaterialType	O

10.2.2.10 Glass Event Facet

Table 91 defines a *Facet* called Glass Event Facet that ca be used to send events when certain processing steps are completed or errors have occurred.

Table 91 – Glass Event Server Facet

Group	Conformance Unit / Profile Title	M / O
Flat Glass	GlassEventType	M
Flat Glass	CommunicationErrorEventType	O
Flat Glass	MaterialReceivedEventType	O
Flat Glass	MaterialMissingEventType	O
Flat Glass	MaterialExitEventType	O
Flat Glass	IntermediateStepEvent	O
Flat Glass	InterruptedEventType	O
Flat Glass	OutOfJobEventType	O
Flat Glass	ProcessParameterOutOfRangeType	O
Flat Glass	ToolMissingEventType	O
Flat Glass	JobMovedEventType	O
Flat Glass	GlassEventType	O

11 Namespaces

11.1 Namespace Metadata

Table 92 defines the namespace metadata for this document. The *Object* is used to provide version information for the namespace and an indication about static *Nodes*. Static *Nodes* are identical for all *Attributes* in all *Servers*, including the *Value Attribute*. Refer to OPC 10000-5 for more details.

The information is provided as *Object* of type *NamespaceMetadataType*. This *Object* is a component of the *Namespaces Object* that is part of the *Server Object*. The *NamespaceMetadataType ObjectType* and its *Properties* are defined in OPC 10000-5.

The version information is also provided as part of the ModelTableEntry in the UANodeSet XML file. The UANodeSet XML schema is defined in OPC 10000-6.

Table 92 – Namespace Metadata Object for this Document

Attribute	**Value**	
BrowseName	http://opcfoundation.org/UA/Glass/Flat/	
Property	**DataType**	**Value**
NamespaceUri	String	http://opcfoundation.org/UA/Glass/Flat/
NamespaceVersion	String	1.0.0
NamespacePublicationDate	DateTime	2022-01-01
IsNamespaceSubset	Boolean	False
StaticNodeIdTypes	IdType[]	0
StaticNumericNodeIdRange	NumericRange[]	
StaticStringNodeIdPattern	String	

11.2 Handling of OPC UA Namespaces

Namespaces are used by OPC UA to create unique identifiers across different naming authorities. The *Attributes* *NodeId* and *BrowseName* are identifiers. A *Node* in the UA *AddressSpace* is unambiguously identified using a *NodeId*. Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different *Nodes* may have the same *BrowseName*. They are used to build a browse path between two *Nodes* or to define a standard *Property*.

Servers may often choose to use the same namespace for the *NodeId* and the *BrowseName*. However, if they want to provide a standard *Property*, its *BrowseName* shall have the namespace of the standards body although the namespace of the *NodeId* reflects something else, for example the *EngineeringUnits Property*. All *NodeIds* of *Nodes* not defined in this document shall not use the standard namespaces.

Table 93 provides a list of mandatory and optional namespaces used in a glass OPC UA *Server*.

Table 93 – Namespaces used in a glass Server

NamespaceURI	**Description**	**Use**
http://opcfoundation.org/UA/	Namespace for *NodeIds* and *BrowseNames* defined in the OPC UA specification. This namespace shall have namespace index 0.	Mandatory
Local Server URI	Namespace for nodes defined in the local server. This may include types and instances used in an AutoID Device represented by the Server. This namespace shall have namespace index 1.	Mandatory
http://opcfoundation.org/UA/DI/	Namespace for *NodeIds* and *BrowseNames* defined in OPC 10000-100. The namespace index is *Server* specific.	Mandatory
http://opcfoundation.org/UA/Glass/Flat	Namespace for *NodeIds* and *BrowseNames* defined in this document. The namespace index is *Server* specific.	Mandatory
Vendor specific types	A *Server* may provide vendor-specific types like types derived from *ObjectTypes* defined in this document in a vendor-specific namespace.	Optional
Vendor specific instances	A *Server* provides vendor-specific instances of the standard types or vendor-specific instances of vendor-specific types in a vendor-specific namespace. It is recommended to separate vendor specific types and vendor specific instances into two or more namespaces.	Mandatory

Table 94 provides a list of namespaces and their index used for *BrowseNames* in this document. The default namespace of this document is not listed since all *BrowseNames* without prefix use this default namespace.

Table 94 – Namespaces used in this document

NamespaceURI	**Namespace Index**	**Example**
http://opcfoundation.org/UA/	0	0:EngineeringUnits
http://opcfoundation.org/UA/DI/	2	2:DeviceRevision
http://opcfoundation.org/UA/Machinery/	3	3:MachineryIdentificationType

Annex A (normative)

Glass Namespace and mappings

A.1 Namespace and identifiers for Glass Information Model

This appendix defines the numeric identifiers for all of the numeric *NodeIds* defined in this specification. The identifiers are specified in a CSV file with the following syntax:

```
<SymbolName>, <Identifier>, <NodeClass>
```

Where the *SymbolName* is either the *BrowseName* of a *Type Node* or the *BrowsePath* for an *Instance Node* that appears in the specification and the *Identifier* is the numeric value for the *NodeId*.

The *BrowsePath* for an *Instance Node* is constructed by appending the *BrowseName* of the instance *Node* to the *BrowseName* for the containing instance or type. An underscore character is used to separate each *BrowseName* in the path. Let's take for example, the GlassMachineIdentificationType *ObjectType Node* which has the *AllowerdFileFormat Property*. The **Name** for the *DeviceLocation InstanceDeclaration* within the *AutoIdDeviceType* declaration is: GlassMachineIdentificationType _ *AllowerdFileFormat*.

The *NamespaceUri* for all *NodeIds* defined here is http://opcfoundation.org/UA/Glass/Flat

The CSV released with this version of the specification can be found here:

– http://www.opcfoundation.org/UA/schemas/Glass/Flat/1.0/NodeIds.csv

NOTE The latest CSV that is compatible with this version of the specification can be found here:

– http://www.opcfoundation.org/UA/schemas/Glass/NodeIds.csv

A computer processible version of the complete Information Model defined in this specification is also provided. It follows the XML Information Model schema syntax defined in OPC 10000-6.

The Information Model Schema for this version of the document (including any revisions, amendments or errata) can be found here:

– http://www.opcfoundation.org/UA/schemas/Glass/Flat/1.0/Opc.Ua.Glass.NodeSet2.xml

NOTE The latest Information Model schema that is compatible with this version of the specification can be found here:

– http://www.opcfoundation.org/UA/schemas/Glass/Flat/Opc.Ua.Glass.NodeSet2.xml