

|  |  |  |
| --- | --- | --- |
|  |  | OPC UA Companion Specification |
| OPC 40501-m | |
| OPC UA for machine tools and manufacturing systems  Part <n>: <Part Name>  Draft 0.05.01  2019-11-14 | |

|  |  |  |  |
| --- | --- | --- | --- |
| Specification Type: | Industry Standard Specification | Comments: |  |
|  |  |  |  |
| Document Number | OPC 40501-m |  |  |
| Title: | :<Part Name> | Date: |  |
|  |  |  |  |
| Version: |  | Software: | MS-Word |
|  |  | Source: | OPC 40501 - UA Companion Specification for machine tools v0.05.01.docx |
|  |  |  |  |
| Author: | VDW e.V. | Status: |  |
|  |  |  |  |

**CONTENTS**

Page

**Figures** v

**Tables** vi

1 Scope 1

2 Normative references 1

3 Terms, abbreviated terms and conventions 2

3.1 Overview 2

3.2 OPC UA for machine tools and manufacturing systems terms 3

3.3 Abbreviated terms 7

3.4 Conventions used in this document 7

3.4.1 Conventions for Node descriptions 7

3.4.2 NodeIds and BrowseNames 9

3.4.2.1 NodeIds 9

3.4.2.2 BrowseNames 9

3.4.3 Common Attributes 9

3.4.3.1 General 9

3.4.3.2 Objects 10

3.4.3.3 Variables 10

3.4.3.4 VariableTypes 10

3.4.3.5 Methods 11

4 General information to <title> and OPC UA 11

4.1 Introduction to <title> 11

4.2 Introduction to OPC Unified Architecture 11

4.2.1 What is OPC UA? 11

4.2.2 Basics of OPC UA 11

4.2.3 Information modelling in OPC UA 12

4.2.3.1 Concepts 12

4.2.3.2 Namespaces 16

4.2.3.3 Companion Specifications 16

5 Use cases 17

5.1 Identify machines of different manufacturers 17

5.2 Overview if production is running 17

5.3 Overview of parts in a job 17

5.4 Overview of runtimes for a job 17

5.5 Overview of machine tool state 17

5.6 Overview of upcoming manual activities 18

5.7 Overview of errors and warnings 18

5.8 Providing data for KPI calculations 18

5.9 Providing an Overview of Tool Data 18

6 umati Information Model overview 20

6.1 Information model overview 20

7 OPC UA ObjectTypes 21

7.1 AlertType Definition 21

7.2 BaseIdentificationType Definition 21

7.3 MachineIdentificationType Definition 21

7.4 SoftwareIdentificationType Definition 22

7.5 BaseMonitoringType Definition 23

7.6 ElementMonitoringType Definition 23

7.7 ChannelMonitoringType Definition 24

7.8 AggregatedChannelMonitoringType Definition 24

7.9 WorkingUnitMonitoringType Definition 25

7.10 SpindleMonitoringType Definition 25

7.11 MachineOperationMonitoringType Definition 26

7.12 BaseProductionType Definition 26

7.13 ProductionJobType Definition 27

7.14 ProductionPartType Definition 28

7.15 ProductionProgramType Definition 29

7.16 BaseToolType Definition 29

7.17 MultiToolType Definition 30

7.18 ToolType Definition 30

7.19 EquipmentType Definition 31

7.20 MachineToolsFolderType 32

7.21 IdentificationType Definition 32

7.22 LampType Definition 32

7.23 MachineToolType Definition 33

7.24 MonitoredElementsListType Definition 34

7.25 MonitoringType Definition 34

7.26 NotificationType Definition 35

7.27 ProductionJobListType Definition 35

7.28 ProductionType Definition 36

7.29 PrognosisListType Definition 37

7.30 PrognosisType Definition 37

7.31 MaintenancePrognosisType Definition 38

7.32 ManualActivityPrognosisType Definition 38

7.33 PartUnloadPrognosisType Definition 39

7.34 ProcessChangeoverPrognosisType Definition 39

7.35 ProductionJobEndPrognosisType Definition 39

7.36 RawPartLoadPrognosisType Definition 40

7.37 ToolChangePrognosisType Definition 40

7.38 UtilityChangePrognosisType Definition 40

7.39 StacklightType Definition 41

7.40 ProductionJobStateMachineType 41

7.41 ProductionPartStateMachineType 44

7.42 ProductionProgramStateMachineType 47

7.43 ToolListType Definition 50

8 *…*OPC UA EventTypes 52

8.1 AlertConditionType 52

8.2 InterruptionConditionType 52

8.3 InterruptionExternalConditionType 52

8.4 InterruptionByOperatorConditionType 53

8.5 InterruptionSafetyConditionType 53

8.6 InterruptionProcessConditionType 53

8.7 InterruptionClampingConditionType 54

8.8 InterruptionManualProcessStepConditionType 54

8.9 InterruptionMeasurementConditionType 55

8.10 InterruptionProcessIrregularityConditionType 55

8.11 InterruptionToolChangeConditionType 55

8.12 NotificationEventType 56

8.13 ProductionJobTransitionEventType 56

8.14 ProductionPartTransitionEventType 56

8.15 ProductionProgramTransitionEventType 57

9 OPC UA VariableTypes 58

9.1 OverrideItemType 58

9.2 ToolLifeType 58

10 OPC UA DataTypes 60

10.1 ChannelState 60

10.2 ControlMode 60

10.3 LampColor 60

10.4 LampStatus 61

10.5 NCProcessing 61

10.6 PartQuality 62

10.7 ProcessIrregularity 62

10.8 SafetyMode 63

10.9 WearIndication 63

11 Instances 65

11.1 MachineTools entry point 65

12 Profiles and ConformanceUnits 66

12.1 Conformance Units 66

12.2 Profiles 66

12.2.1 Profile list 66

12.2.2 Server Facets 67

12.2.2.1 Overview 67

12.2.2.2 <short name> <Prf1name> Server Profile 67

12.2.2.3 <short name><Prf2name> Server Facet 67

12.2.3 Client Facets 68

12.2.3.1 Overview 68

12.2.3.2 <short name> < Prf3name> Client Facet 68

13 Namespaces 68

13.1 Namespace Metadata 68

13.2 Handling of OPC UA Namespaces 69

Annex A (normative): <Title> Namespace and mappings 71

A.1 Namespace and identifiers for <Title> Information Model 71

**Figures**

Figure 1 – The Scope of OPC UA within an Enterprise 12

Figure 2 – A Basic Object in an OPC UA Address Space 13

Figure 3 – The Relationship between Type Definitions and Instances 14

Figure 4 – Examples of References between Objects 15

Figure 5 – The OPC UA Information Model Notation 15

Figure 6: The States and Transitions of the ProductionJobStateMachineType 42

Figure 7 : The States and Transitions of the ProductionPartStateMachineType 45

Figure 8 : The States and Transitions of the ProductionProgramStateMachineType 48

**Tables**

[Table 1 – Examples of DataTypes 8](#_Toc25240395)

[Table 2 – Type Definition Table 8](#_Toc25240396)

[Table 3 – Examples of Other Characteristics 9](#_Toc25240397)

[Table 4 – Common Node Attributes 10](#_Toc25240398)

[Table 5 – Common Object Attributes 10](#_Toc25240399)

[Table 6 – Common Variable Attributes 10](#_Toc25240400)

[Table 7 – Common VariableType Attributes 11](#_Toc25240401)

[Table 8 – Common Method Attributes 11](#_Toc25240402)

[Table 9 – AlertType Definition 21](#_Toc25240403)

[Table 10 – BaseIdentificationType Definition 21](#_Toc25240404)

[Table 11 – MachineIdentificationType Definition 22](#_Toc25240405)

[Table 12 – SoftwareIdentificationType Definition 23](#_Toc25240406)

[Table 13 – BaseMonitoringType Definition 23](#_Toc25240407)

[Table 14 - ElementMonitoringType Definition 24](#_Toc25240408)

[Table 15 - ChannelMonitoringType Definition 24](#_Toc25240409)

[Table 16 - AggregatedChannelMonitoringType Definition 25](#_Toc25240410)

[Table 17 - WorkingUnitMonitoringType Definition 25](#_Toc25240411)

[Table 18 - SpindleMonitoringType Definition 25](#_Toc25240412)

[Table 19 - MachineOperationMonitoringType Definition 26](#_Toc25240413)

[Table 20 - BaseProductionType Definition 26](#_Toc25240414)

[Table 21 - ProductionJobType Definition 27](#_Toc25240415)

[Table 22 – PartsInProduction Definition 28](#_Toc25240416)

[Table 23 – ProductionPrograms Definition 28](#_Toc25240417)

[Table 24 - ProductionPartType Definition 29](#_Toc25240418)

[Table 25 - ProductionProgramType Definition 29](#_Toc25240419)

[Table 26 - BaseToolType Definition 30](#_Toc25240420)

[Table 27 - MultiToolType Definition 30](#_Toc25240421)

[Table 28 - ToolType Definition 31](#_Toc25240422)

[Table 29 - EquipmentType Definition 31](#_Toc25240423)

[Table 30 - MachineToolsFolderType Definition 32](#_Toc25240424)

[Table 31 - IdentificationType Definition 32](#_Toc25240425)

[Table 32 - LampType Definition 33](#_Toc25240426)

[Table 33 - MachineToolType Definition 33](#_Toc25240427)

[Table 34 - MonitoredElementsListType Definition 34](#_Toc25240428)

[Table 35 - MonitoringType Definition 34](#_Toc25240429)

[Table 36 - NotificationType Definition 35](#_Toc25240430)

[Table 37 – Messages Definition 35](#_Toc25240431)

[Table 38 - ProductionJobListType Definition 36](#_Toc25240432)

[Table 39 - ProductionType Definition 36](#_Toc25240433)

[Table 40 – Statistics Definition 37](#_Toc25240434)

[Table 41 - PrognosisListType Definition 37](#_Toc25240435)

[Table 42 - PrognosisType Definition 38](#_Toc25240436)

[Table 43 - MaintenancePrognosisType Definition 38](#_Toc25240437)

[Table 44 - ManualActivityPrognosisType Definition 39](#_Toc25240438)

[Table 45 - PartUnloadPrognosisType Definition 39](#_Toc25240439)

[Table 46 - ProcessChangeoverPrognosisType Definition 39](#_Toc25240440)

[Table 47 - ProductionJobEndPrognosisType Definition 40](#_Toc25240441)

[Table 48 - RawPartLoadPrognosisType Definition 40](#_Toc25240442)

[Table 49 - ToolChangePrognosisType Definition 40](#_Toc25240443)

[Table 50 - UtilityChangePrognosisType Definition 41](#_Toc25240444)

[Table 51 - StacklightType Definition 41](#_Toc25240445)

[Table 52 - ProductionJobStateMachineType Definition 42](#_Toc25240446)

[Table 53: State Numbers for the ProductionJobStateMachineType 43](#_Toc25240447)

[Table 54 - ProductionJobStateMachineType Transitions 44](#_Toc25240448)

[Table 55 - ProductionPartStateMachineType Definition 45](#_Toc25240449)

[Table 56: State Numbers for the ProductionPartStateMachineType 46](#_Toc25240450)

[Table 57 - ProductionPartStateMachineType Transitions 47](#_Toc25240451)

[Table 58 - ProductionProgramStateMachineType Definition 48](#_Toc25240452)

[Table 59: State Numbers for the ProductionJobStateMachineType 49](#_Toc25240453)

[Table 60 - ProductionProgramStateMachineType Transitions 50](#_Toc25240454)

[Table 61 - ToolListType Definition 51](#_Toc25240455)

[Table 62 –AlertConditionType Definition 52](#_Toc25240456)

[Table 63 – InterruptionConditionType Definition 52](#_Toc25240457)

[Table 64 – InterruptionExternalConditionType Definition 52](#_Toc25240458)

[Table 65 – InterruptionByOperatorConditionType Definition 53](#_Toc25240459)

[Table 66 – InterruptionSafetyConditionType Definition 53](#_Toc25240460)

[Table 67 – InterruptionProcessConditionType Definition 54](#_Toc25240461)

[Table 68 – InterruptionClampingConditionType Definition 54](#_Toc25240462)

[Table 69 – InterruptionManualProcessStepConditionType Definition 54](#_Toc25240463)

[Table 70 – InterruptionMeasurementConditionType Definition 55](#_Toc25240464)

[Table 71 – InterruptionProcessIrregularityConditionType Definition 55](#_Toc25240465)

[Table 72 – InterruptionToolChangeConditionType Definition 55](#_Toc25240466)

[Table 73 –NotificationEventType Definition 56](#_Toc25240467)

[Table 74 – ProductionJobTransitionEventType Definition 56](#_Toc25240468)

[Table 75 – ProductionPartTransitionEventType Definition 57](#_Toc25240469)

[Table 76 – ProductionProgramTransitionEventType Definition 57](#_Toc25240470)

[Table 77 – OverrideItemType Definition 58](#_Toc25240471)

[Table 78 – ToolLifeType Definition 58](#_Toc25240472)

[Table 79 – ChannelState Definition 60](#_Toc25240473)

[Table 80 – ChannelState Definition 60](#_Toc25240474)

[Table 81 – ControlMode Definition 60](#_Toc25240475)

[Table 82 – ControlMode Definition 60](#_Toc25240476)

[Table 83 – LampColor Definition 61](#_Toc25240477)

[Table 84 – LampColor Definition 61](#_Toc25240478)

[Table 85 – LampStatus Definition 61](#_Toc25240479)

[Table 86 – LampStatus Definition 61](#_Toc25240480)

[Table 87 – NCProcessing Definition 62](#_Toc25240481)

[Table 88 – NCProcessing Definition 62](#_Toc25240482)

[Table 89 – PartQuality Definition 62](#_Toc25240483)

[Table 90 – PartQuality Definition 62](#_Toc25240484)

[Table 91 – ProcessIrregularity Definition 63](#_Toc25240485)

[Table 92 – ProcessIrregularity Definition 63](#_Toc25240486)

[Table 93 – SafetyMode Definition 63](#_Toc25240487)

[Table 94 – SafetyMode Definition 63](#_Toc25240488)

[Table 95 – WearIndication Definition 64](#_Toc25240489)

[Table 96 – WearIndication Definition 64](#_Toc25240490)

[Table 97 – MachineTools Definition 65](#_Toc25240491)

[Table 98 – Conformance Units for <Title> 66](#_Toc25240492)

[Table 99 – Profile URIs for <Title> 66](#_Toc25240493)

[Table 100 - <short name> <Prf1name> Server Profile 67](#_Toc25240494)

[Table 101 - <short name> <Prf2name> Server Facet 67](#_Toc25240495)

[Table 102 - <short name> < Prf3name> Client Facet 68](#_Toc25240496)

[Table 103 – NamespaceMetadata Object for this Document 69](#_Toc25240497)

[Table 104 – Namespaces used in a umati Server 69](#_Toc25240498)

[Table 105 – Namespaces used in this document 70](#_Toc25240499)

OPC FOUNDATION, German Machine tool builders’ Association (VDW)

\_\_\_\_\_\_\_\_\_\_\_\_

This specification is a derivative work of the public domain document OPC UA Companion Specification Template, by OPC Foundation.

AGREEMENT OF USE

COPYRIGHT RESTRICTIONS

* This document is provided "as is" by the OPC Foundation and the VDW.
* Right of use for this specification is restricted to this specification and does not grant rights of use for referred documents.
* Right of use for this specification will be granted without cost.
* This document may be distributed through computer systems, printed or copied as long as the content remains unchanged and the document is not modified.
* OPC Foundation and VDW do not guarantee usability for any purpose and shall not be made liable for any case using the content of this document.
* The user of the document agrees to indemnify OPC Foundation and VDW and their officers, directors and agents harmless from all demands, claims, actions, losses, damages (including damages from personal injuries), costs and expenses (including attorneys' fees) which are in any way related to activities associated with its use of content from this specification.
* The document shall not be used in conjunction with company advertising, shall not be sold or licensed to any party.
* The intellectual property and copyright is solely owned by the OPC Foundation and the VDW.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OPC or VDW specifications may require use of an invention covered by patent rights. OPC Foundation or VDW shall not be responsible for identifying patents for which a license may be required by any OPC or VDW specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OPC or VDW specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

WARRANTY AND LIABILITY DISCLAIMERS

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OPC FOUDATION NOR VDW MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OPC FOUNDATION NOR VDW BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you.

RESTRICTED RIGHTS LEGEND

This Specification is provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor / manufacturer are the OPC Foundation, 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ, 85260-1830

COMPLIANCE

The combination of VDW and OPC Foundation shall at all times be the sole entities that may authorize developers, suppliers and sellers of hardware and software to use certification marks, trademarks or other special designations to indicate compliance with these materials as specified within this document. Products developed using this specification may claim compliance or conformance with this specification if and only if the software satisfactorily meets the certification requirements set by VDW or the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

Trademarks

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of Germany.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

# Scope

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

German Machine Tool Builders’ Association (VDW)

The VDW, based in Frankfurt/Main, represents the interests of the German machine tool industry. In this context VDW is interested in increasing the innovation and competitive capacity of machine tool builders and manufacturers of machine tool controllers by introducing a unified machine tool interface. This universal interface is understood as essential prerequisite towards digital manufacturing.

# Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments and errata) applies.

For references to the OPC UA Specification it is recommended to define the minimum required version. Example: “1.04.03 is the minimum required version for the following OPC Unified Architecture parts.”

The build number of the version (in this case “03”) refers to an ERRATA document with the corresponding version (see <https://opcfoundation.org/developer-tools/specifications-unified-architecture/errata-and-amendments/> for the published ERRATA documents).

<Insert only references that apply to this document. Following are examples only>

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 10000-8, *OPC Unified Architecture - Part 8: Data Access*

<http://www.opcfoundation.org/UA/Part8/>

OPC 10000-9, *OPC Unified Architecture - Part 9: Alarms and Conditions*

<http://www.opcfoundation.org/UA/Part9/>

OPC 10000-10, *OPC Unified Architecture - Part 10: Programs*

<http://www.opcfoundation.org/UA/Part10/>

OPC 10000-11, *OPC Unified Architecture - Part 11: Historical Access*

<http://www.opcfoundation.org/UA/Part11/>

OPC 10000-12, *OPC Unified Architecture - Part 12: Discovery and Global Services*

<http://www.opcfoundation.org/UA/Part12/>

OPC 10000-13, *OPC Unified Architecture - Part 13: Aggregates*

<http://www.opcfoundation.org/UA/Part13/>

OPC 10000-14, *OPC Unified Architecture - Part 14: PubSub*

[http://www.opcfoundation.org/UA/Part14/](http://www.opcfoundation.org/UA/Part14/%20)

OPC 10001-1, *OPC Unified Architecture V1.04 - Amendment 1: AnalogItem Types*

<http://www.opcfoundation.org/UA/Amendment1/>

OPC 10001-3, *OPC Unified Architecture V1.04 - Amendment 3: Method Metadata*

<http://www.opcfoundation.org/UA/Amendment3/>

OPC 10001-5, *OPC Unified Architecture V1.04 - Amendment 5: Dictionary Reference*

<http://www.opcfoundation.org/UA/Amendment5/>

OPC 10001-7, *OPC Unified Architecture V1.04 - Amendment 7: Interfaces ad AddIns*

<http://www.opcfoundation.org/UA/Amendment7/>

OPC 10001-11, *OPC Unified Architecture V1.04 -* [*Amendment 11: Spatial Types*](https://reference.opcfoundation.org/Core/docs/Amendment11/)

<http://www.opcfoundation.org/UA/Amendment11/>

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

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

# Terms, abbreviated terms and conventions

## Overview

It is assumed that basic concepts of OPC UA information modelling and <other specifications> are understood in this document. This document will use these concepts to describe the <title> 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 document are *italicized* in the document.

## OPC UA for machine tools and manufacturing systems terms

Alert

A defined message indicating noteworthy information for the operator and for historic data.   
An alert can have three subcategories: Error - indicating a state that blocks operation of the process and needs human intervention.  
Warning - indicating a state that requires attention, it can prevent operation in indicated way, however it is generally not blocking operation of the process.  
Message - display of information the machine tool builder deemed necessary to display, neither blocking or reducing operational capability.

Channel

Runtime component of the NC which executes an NC-program.   
This execution may happen in Block Sequence mode (execution of the next NC command starts as soon as the previous has completed), or Single Block Mode (NC channels stops and waits for a NC Start signal to resume executing the NC program). A channel contains an assigned set of axes which can be moved in a synchronized and interpolated manner. Auxiliary axes may also be assigned to a channel which will usually be commanded in an uninterpolated manner. A productive NC channel runs actual NC programs which relate to the workpiece. Auxiliary NC channels may exist in a machine to command tool magazine or loader motion, which is not the main focus for 2019 use cases. Depending on the machine there may be several productive NC channels.

Controller:

Based on DIN 4000: Device which control a process by computing control data.

Identification:

The Identification branch of the UMATI information model holds static data which shall uniquely identify a machine among a pool of the machine operating entity.   
It should typically remain constant over several weeks or even over the entire product life of the machine. Contents shall be of relevance to the machine operator or OEM.

Job

Aggregated production data for running a sequence to produce several parts after one preparation mounting.   
Examples for such a mounting are putting four raw parts on a pallet for a machining center, setting up the fitting diameter bars in a turning center bar feeder or loading a sheet metal blank from which hundreds of parts can be cut/punched. This sequence shall represent several parts which will usually (but not always) be several identical products. A job may be executed several times.

Lamp

Lamp is an element of the stacklight (sometimes referred to as signal light) which is found on virtually all production machinery.   
Color codes of the semantics may often be aligned to DIN EN60204-1, but definitions specific to a machine-operating company or an OEM commonly occur.

Machine

The Machine represents the entire machine interface of the information model. It is the entry point to the umati OPC UA interface. It gives a basic structure to the interface.

Machine Tool

Machine tool as defined by DIN, in 2019 covering chipping machine tools, metal sheet punching, and bending machines and laser processing machines.

Maintenance Prognosis

A prognosis of the control system indicating at which time in the future a specific maintenance action may become necessary.   
Examples may be oil changes, filter mat replacements or regular checks. Reliability will rely on the specific case and cannot be guaranteed to be precise.

Manual Tool Change

Manual action of inserting a tool into the machine as opposed to an Automatic Tool Change.   
There are two common reasons this is done or necessary: 1) tool life of one group of tools has expired and machining cannot continue until a new tool with sufficient tool life for the next operation is inserted (causing a tool change) 2) a tool for a give job is not available (or defined as "hand tool" and) must be provisioned.

Manual Activity Prognosis

A prognosis of the control system indicating at which time in the future a manual intervention may become necessary.   
Examples may be manual tool changes for deep boring tools which do not fit in the tool magazine. Reliability will rely on the specific case and cannot be guaranteed to be precise.

Multitool

Unit of different tools, usually used in order to have several tools available in-process without requiring explicit tool-changes.   
Typical applications are in turning, when one indexed position of the tool revolver holds several outer-diameter cutting inserts and boring tools, such that a tool change process can quickly complete by merely readjusting the NC setpoint position tool compensation.

Part

A Part is the workpiece of the machine which is treated in the purpose of the machine.   
This may be for the purpose of machining, measuring or others, depending on the machine type.

Part Change Prognosis

Prediction made by the machine or its controller at which point in the future a part change is necessary/going to happen.

Parts in production

A list of the parts, which are currently present in the machine's work area, typically for manufacturing purposes. This does not include parts in the supply chain.

Production

Production is the structuring branch to hold information focusing on current production and planned production of the machine. It is a representation of a collection of jobs.

Production Plan

A list of all job elements a specific machine knows about. I.e. all jobs which were somehow transferred to the machine.   
Note that the single Machine may get the Job for an operation sequence when machining parts are addressed to this specific machine only, depending on the production workflow on-site.

Production Job Prognosis Type

Estimated timespan until the end of the current Job.

Prognoses

Structuring node to collect estimations of the machine control system about future times (or timespans) when certain events may occur.

Program

A program is a list of operations that the controllers performs in sequence. It's usually a machine-readable file which is needed for the controller to fulfill the job.

Replacement Tool

A tool with equivalent (identical) process capabilities (size and functionality) as an existing tool. Replacement tools are automatically used if the designated tool is locked due to wear.

Software Component Version

SoftwareComponentVersion holds information about the specific software in operation in the machine. Almost all modern machine tools operate on several software system components, this shall enable presentation of all relevant software components (NC Kernel, HMI base system, etc.)

Spindle State Mode

The spindlestatemode is a collection of information about the rotary process axis.   
Depending on the actual context of the machine this may be a tool-holding milling spindle or a workpiece-holding turning spindle for example.

Stacklight

The stacklight is a visual machine state indicator. It consists of one or more lamps stacked on top of one another, each having another color.   
The combination of on/off/blinking lights corresponds to a machine state. The ordering of the colors is counted from bottom to top and from left to right.  The stacklight object is the virtual representation of the physical stacklight of the machine.

State Mode List

An aggregation element for states of (sub)components of the machine tool.

Tool

Tools are exchangeable components used in a machine tool to execute the production process and may for example be drills, ball milling heads, cutting inserts, pinching tools and so forth.   
May be a non-contact tool, for example a processing laser.

Tool List

A ToolList is a list of tools, where a tool may be a single tool or a multitool.   
Multitools carry several tools on one tool magazine socket or one revolver index position and will be mounted into the machine as one prepared unit.

Utility

All for running the machine necessary media (pressurized air, coolant, lubrication, ...) and consumables (filters, (space in) chip carts, .... ).   
Tools as consumables are excluded as tools are in their own class of complexity and therefore defined separately.

Utility Change Prognosis

The UtilityChangePrognosis is an estimate of the time the current utility supply is sufficient that the production process can run without intervention.

Wear

State of decay/ usage of a tool. Wear can be measured in usage e.g. number of times the tool has been changed into the spindle, minutes of run time or deviation of a defined geometry.

## Abbreviated terms

The following abbreviations are examples. The list shall only contain abbreviations used in the document.

AC Alarm and Condition

DCS Distributed Control Systems

umati Universal Machine Tool Interface

MES Manufacturing Execution System

ERP Enterprise Resource Planning

## Conventions used in this document

Following are basic conventions that shall be followed for all formal definitions used.

### Conventions for Node descriptions

*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** | **Data­Type** | **Value­Rank** | **Array­Dimensions** | **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 3.4.2.2).

*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 Other 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 “--“ is used. | | | | |
|  |  | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| *ReferenceType* name | *NodeClass* of the *TargetNode*. | *BrowseName* of the target *Node*. If the *Reference* is to be instantiated by the server, then the value of the target Node’s BrowseName is “--“. | *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 3.4.3.1. 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.

### NodeIds and BrowseNames

#### 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*.

#### 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 in Annex A.

If the *BrowseName* is not defined by this document, a namespace index prefix like ‘0:EngineeringUnits’ or ‘2:DeviceRevision’ is added to the *BrowseName*. This is typically necessary if a *Property* of another specification is overwritten or used in the OPC UA types defined in this document. Table 105 provides a list of namespaces and their indexes as used in this document.

### Common Attributes

#### 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 document or if it is server-specific.

For all *Nodes* specified in this document, the *Attributes* named in Table 4 shall be set as specified in the table.

Table 4 – 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* are 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 3.4.2.1. |
| 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 document. |
| 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-specific and depends on the *RolePermissions* *Attribute* (if provided) and the current *Session*. |
| AccessRestrictions | Optionally server-specific access restrictions can be provided. |

#### Objects

For all *Objects* specified in this document, the *Attributes* named in Table 5 shall be set as specified in the Table 5. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 5 – Common Object Attributes

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

#### Variables

For all *Variables* specified in this document, the *Attributes* named in Table 6 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 6 – 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 document, 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. |

#### VariableTypes

For all *VariableTypes* specified in this document, the *Attributes* named in Table 7 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC 10000-3.

Table 7 – 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*. |

#### Methods

For all *Methods* specified in this document, 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 Method Attributes

|  |  |
| --- | --- |
| **Attributes** | **Value** |
| Executable | All *Methods* defined in this document 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. |

# General information to <title> and OPC UA

## Introduction to <title>

Insert an introduction (about one page) of the companion organization and the model that it represents.

## Introduction to OPC Unified Architecture

This is an OPC UA introduction that may be used as is, shortened or enhanced as appropriate.

### 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 <title>, 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.

### 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 1.



Figure 1 – 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.

### Information modelling in OPC UA

#### 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 2.



Figure 2 – 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 3 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 3 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.



Figure 3 – 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 4 depicts several *References,* connecting different *Objects*.



Figure 4 – 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 5. 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 5 – 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 expected 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)

#### 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. Namespaces in OPC UA have a globally unique string called a NamespaceUri and a locally unique integer called a *NamespaceIndex*. The *NamespaceIndex* is only unique within the context of a *Session* between an OPC UA *Client* and an OPC UA *Server*. The *Services* defined for OPC UA use the *NamespaceIndex* to specify the Namespace for qualified values.

There are two types of values in OPC UA that are qualified with Namespaces: NodeIds and *QualifiedNames*. NodeIds are globally unique identifiers for *Nodes*. This means the same *Node* with the same NodeId can appear in many *Servers*. This, in turn, means *Clients* can have built in knowledge of some *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.

#### 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.

# Use cases

|  |
| --- |
| **Disclaimer concerning the use case formulations:**  The formulations have in large parts been taken from the previous draft and still need to be matched to the current model. Not all ObjectTypes mentioned here match the model described in Section 7. |

## Identify machines of different manufacturers

The machines of different manufacturers shall be identifiable in a standardized manner. To realize this, a number of basic and static information like manufacturer name and model number are offered on the umati interface. This information can be found on the interface in an instance of the *IdentificationType*.

## Overview if production is running

Using information provided by the umati interface, an overview if the machine is in production or not should be possible. Additionally, if the machine tool is in an erroneous state, it needs to be evident over the interface.

If the machine tool is not in production, the reason for this state should also be identifiable.

The information of the machine and controller state can be found in the information model in the *Monitoring* Component of the *MachineToolType*.

Other nodes that provide important information for an overview if the production is running are the override values of CNC Channels and Spindles in the *ChannelMonitoringType* and the *SpindleMonitoringType*. The *MachineOperationMonitoringType* also provides the parameter LeastOneAxisMoving, which indicates if an axis has an active movement command.

Another indication of the machine status is the machine stacklight. Its representation in the umati information model can be found in the *StacklightType*.

The errors and warnings on the machine shall be available on the umati interface with the OPC UA mechanism described in OPC UA Part 9 – Alarms and Conditions.

## Overview of parts in a job

Using the umati interface, an overview of the target and actual manufactured parts is possible. Additionally, it is possible to see which parts belong to which internal or customer order. If there is an irregularity in the process which might affect the part quality, the parts representation on the interface is marked accordingly.

The relevant information can be found in the information model in the part counters of the *ProductionProgramType*, the CustomerOrderId of *PartType* and *ProductionJobType* and the quality information of the *PartType*.

## Overview of runtimes for a job

In order to calculate cycle times and prognoses for production, the umati interface provides the time data of start, end, interruption and abortion of machining processes and programs on the machine tool.

The events can be found in the information model as *InterruptionConditionType* with all its subtypes (that specify the reason for the interruption further), *EndEventType*, *StartEventType* and*AbortEventType*.

To receive the events for a specific program, job or controller, the OPC UA client can subscribe to the associated StateMachine.

## Overview of machine tool state

With the interface, information on the machine tool state like tool changes or part changes is available. The states of CNC channels and controllers in the machine tool are available as well.

In the information model, there are the *InterruptionToolChangeEventType* and the *InterruptionClampingEventType* to signal a tool change and a part clamping event. As these are subtypes of the *ConditionType* defined in OPC UA Part 9, they have an enabled and a disabled state. These states allow to mark the beginning and end of the interruption.

The information model also holds the states of spindles and CNC channels in the *ChannelMonitoringType* and the*SpindleMonitoringType*.

A controller state is represented with the *ControllerStateModeType*.

## Overview of upcoming manual activities

For a machine operator who works on multiple machines in his shift, an indication which of the machines has the soonest need of a manual intervention is helpful (e.g. tool change, part change, preparation for the next job…).

To achieve this, the umati interface offers the possibility to give prognoses for different events. These prognoses can of course only be provided if the machine can estimate the time of the respective future event.

The available types of prognoses are: *PrognosisType, MaintenancePrognosisType, ManualActivityPrognosisType, PartUnloadPrognosisType, ProcessChangeoverPrognosisType, ProductionJobEndPrognosisType, RawPartLoadPrognosisType ToolChangePrognosisType and UtilityChangePrognosisType*.

All of these prognoses have a parameter which indicates if the required action is automated or not.

On the umati interface, there is a list of available prognoses, which is of *PrognosisListType*. It contains all known prognoses with their times to happen. It also contains a parameter with the soonest time.

## Overview of errors and warnings

The machine is expected to offer all current errors and warnings over the umati interface.

These errors and warnings shall be mapped to OPC UA event types accordingly. For errors, umati offers the *AlertConditionType*. For messages with lower urgency, there is the *NotificationEventType*.

## Providing data for KPI calculations

To facilitate the calculation of different KPIs like for example OEE, the umati interface offers different machine times. These times allow to calculate the durations of different machine modes.

All of these relevant times are transferred via the event mechanism in OPC UA.

This happens with the *ProductionStateMachineType*, which is a part of every subtype of the *BaseProductionType*. This means, every *Job*, *Program*, and *Part* has its own state machine with start, end, abortion and interruption states. Each state change sends an appropriate event as a notification. This event can be received by the OPC UA client, and the timestamp can be used to calculate the time durations needed for KPI compilation.

## Providing an Overview of Tool Data

On the machine interface, data concerning the tools in the machine is available.

In the umati interface, tools are modelled with the *MultiToolType* and *ToolType* and aggregated in a list with the *ToolListType*.

In umati, the tool data are constrained to very basic information. Especially all geometric information about the tool is omitted on the interface. This is mainly due to the multitude of different norms for different tools.

On the interface, there are the identifiers of the tools in the machine. With these, it can be verified if a machine is prepared to execute a certain machining task.

There is also some information about the wear condition of the tool. The interface will create a warning if a certain wear level has been reached and it will create an alert if another, higher wear level is reached.

If there are multiple tools of the same type equipped in the machine, the one that will primarily be used in the machining process is marked as active. Using this information, the tool distribution among different machines can be planned remotely and changed without disturbing the current machine operation.

# umati Information Model overview

## Information model overview

An overview of the model elements and how they relate to each other.

Following shall be sections that specify the companion information model. Such models may vary and no fixed structure can be given. An option could be to have separate chapters for ObjectTypes, VariableTypes, DataTypes, a.s.o.

# OPC UA ObjectTypes

|  |
| --- |
| **Disclaimer concerning *BaseIdentificationType*, *MachineIdentificationType* and *SoftwareIdentificationType*:**  The identification information for use in OPC UA Companion Specifications is currently being discussed in the Harmonization Group in VDMA. We intend to use the resulting modeling suggestion in umati. This will most probably result in a change of *BaseIdentificationType*, *MachineIdentificationType* and *SoftwareIdentificationType*. |

## BaseIdentificationType Definition

The *BaseIdentificationType* serves as a supertype to the *MachineIdentificationType* and the *SoftwareIdentificationType*. It is an abstract type.

It is formally defined in Table 10.

Table 10 – BaseIdentificationType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:BaseIdentificationType | | | | |
| IsAbstract | True | | | | |
| **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:HasSubtype | ObjectType | 1:MachineIdentificationType | Defined in 7.3 | | |
| 0:HasSubtype | ObjectType | 1:SoftwareIdentificationType | Defined in 7.4 | | |
| 0:HasComponent | Variable | 1:Manufacturer | 0:String | 0:BaseDataVariableType | O |

*Manufacturer* is used to …

## MachineIdentificationType Definition

The *MachineIdentificationType* is …

The *MachineIdentificationType* is a subtype of the *BaseIdentificationType* defined in 7.2 and inherits all InstanceDeclarations of the *BaseIdentificationType*. The *Manufacturer* component is overridden to be mandatory.

The *MachineIdentificationType* is formally defined in Table 11.

Table 11 – MachineIdentificationType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:MachineIdentificationType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseIdentificationType* defined in 7.2 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:BuildYear | 0:UInt32 | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:CatalogueName | 0:String | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:CustomName | 0:String | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:LocationMachine | 0:String | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | 1:LocationPlant | 0:String | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | 1:Manufacturer | 0:String | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:SerialNumber | 0:String | 0:BaseDataVariableType | M |

*BuildYear* is …

*CatalogueName* refers to …

*CustomName* is …

*LocationMachine* refers to …

*LocationPlant* is …

*Manufacturer* is described in 7.2 for the *BaseIdentificationType*.

*SerialNumber* is …

## SoftwareIdentificationType Definition

The *SoftwareIdentificationType* is a subtype of the *BaseIdentificationType* defined in 7.2 and inherits all InstanceDeclarations of the *BaseIdentificationType*.

The *SoftwareIdentificationType* holds information about the specific software in operation in the machine. Almost all modern machine tools operate on several software system components, this shall enable presentation of all relevant software components (NC Kernel, HMI base system, etc.)

The *SoftwareIdentificationType* is formally defined in Table 12.

Table 12 – SoftwareIdentificationType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:SoftwareIdentificationType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseIdentificationType* defined in 7.2 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Object | 1:<SoftwareComponent> |  | 1:SoftwareIdentificationType | OptionalPlaceholder |
| 0:HasComponent | Variable | 1:ComponentVersion | 0:String | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:Identifier | 0:String | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:Manufacturer | 0:String | 0:BaseDataVariableType | O |

The OptionalPlaceholder <*SoftwareComponent*> allows to model a *SoftwareComponent* consisting of multiple independent *SoftwareComponent* elements.

*ComponentVersion* describes …

The *Identifier* Property is…

*Manufacturer* is described in 7.2 for the *BaseIdentificationType*.

## BaseMonitoringType Definition

The *BaseMonitoringType* is used as a supertype to the *ElementMonitoringType* and the *MachineOperationMonitoringType*. It is an abstract type.

The *BaseMonitoringType* is defined in Table 13.

Table 13 – BaseMonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:BaseMonitoringType | | | | |
| IsAbstract | True | | | | |
| **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:HasSubtype | ObjectType | 1:ElementMonitoringType | Defined in 7.6 | | |
| 0:HasSubtype | ObjectType | 1:MachineOperationMonitoringType | Defined in 7.11 | | |

There are no other *References* except for *HasSubtype* *References* specified for the *BaseMonitoringType*.

## ElementMonitoringType Definition

The *ElementMonitoringType* is intended to be a supertype for all monitoring information that is special to a specific element within the machine tool. An element doesn’t have to be a physical component. Examples for such elements are CNC channels or spindles.

The ElementMonitoringType is formally defined in Table 14.

Table 14 - ElementMonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ElementMonitoringType | | | | |
| IsAbstract | True | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseMonitoringType* defined in 7.5 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasSubtype | ObjectType | 1:ChannelMonitoringType | Defined in 7.7 | | |
| 0:HasSubtype | ObjectType | 1:WorkingUnitMonitoringType | Defined in 7.9 | | |
| 0:HasProperty | Variable | 1:Name | 0:String | 0:PropertyType | M |

The *Name* property refers to …

## ChannelMonitoringType Definition

The *ChannelMonitoringType* provides …

It is formally defined in Table 15.

Table 15 - ChannelMonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ChannelMonitoringType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *ElementMonitoringType* defined in 7.6 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasSubtype | ObjectType | 1:AggregatedChannelMonitoringType | Defined in 7.8 | | |
| 0:HasComponent | Variable | 1:ChannelState | 1:ChannelState | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:ControlMode | 1:ControlMode | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | 1:FeedOverride | 0:Double | 1:OverrideItemType | M |
| 0:HasComponent | Variable | 1:NcProcessing | 1:NCProcessing | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | 1:RapidOverride | 0:Double | 1:OverrideItemType | O |

*ChannelState* is …

*ControlMode* refers to …

*FeedOverride* is …

*NcProcessing* is …

*RapidOverride* refers to …

## AggregatedChannelMonitoringType Definition

The *AggregatedChannelMonitoringType* is a subtype of the *ChannelMonitoringType* and inherits all its InstanceDeclarations. Using this type instead of a *ChannelMonitoringType* provides …

The *AggregatedChannelMonitoringType* is formally defined in Table 16.

Table 16 - AggregatedChannelMonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:AggregatedChannelMonitoringType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *ChannelMonitoringType* defined in 7.7 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |

The *AggregatedChannelMonitoringType* contains no further *References* than the ones inherited.

## WorkingUnitMonitoringType Definition

The *WorkingUnitMonitoringType* is a supertype for monitoring information of machine tool elements that are a direct and active part of the manufacturing process. It is an abstract type, only its subtypes shall be instantiated. At the moment, the only subtype defined is the *SpindleMonitoringType*.

The *WorkingUnitMonitoringType* is formally defined in Table 17.

Table 17 - WorkingUnitMonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:WorkingUnitMonitoringType | | | | |
| IsAbstract | True | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *ElementMonitoringType* defined in 7.6 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasSubtype | ObjectType | 1:SpindleMonitoringType | Defined in 7.10 | | |

The *WorkingUnitMonitoringType* has no other *References* than *HasSubtype* *References*.

## SpindleMonitoringType Definition

The *SpindleMonitoringType* is a collection of information about the rotary process axis.

Depending on the actual context of the machine this may be a tool-holding milling spindle or a workpiece-holding turning spindle for example.

The *SpindleMonitoringType* is formally defined in Table 18.

Table 18 - SpindleMonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:SpindleMonitoringType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *WorkingUnitMonitoringType* defined in 7.9 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:SpindleIsRotating | 0:Boolean | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:SpindleOverride | 0:Double | 1:OverrideItemType | O |

*SpindleIsRotating* indicates …

*SpindleOverride* is …

## MachineOperationMonitoringType Definition

The *MachineOperationMonitoringType* provides …

It is formally defined in Table 19.

Table 19 - MachineOperationMonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:MachineOperationMonitoringType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseMonitoringType* defined in 7.5 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:HoursOfOperation | 0:UInt32 | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | 1:LeastOneAxisMoving | 0:Boolean | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | 1:SafetyMode | 1:SafetyMode | 0:BaseDataVariableType | M |

*HoursOfOperation* provides …

*LeastOneAxisMoving* indicates …

*SafetyMode* is …

## BaseProductionType Definition

The *BaseProductionType* serves as a supertype for the *ProductionJobType*, *ProductionPartType* and *ProductionProgramType*, which provide information about the production job on the machine tool. As an abstract type, the *BaseProductionType* itself cannot be instantiated, only its subtypes can.

The *BaseProductionType* is formally defined in Table 20.

Table 20 - BaseProductionType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:BaseProductionType | | | | |
| IsAbstract | True | | | | |
| **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:HasSubtype | ObjectType | 1:ProductionJobType | Defined in 7.13 | | |
| 0:HasSubtype | ObjectType | 1:ProductionPartType | Defined in 7.14 | | |
| 0:HasSubtype | ObjectType | 1:ProductionProgramType | Defined in 7.15 | | |
| 0:HasProperty | Variable | 1:Identifier | 0:String | 0:PropertyType | M |
| 0:HasProperty | Variable | 1:IndexInList | 0:UInt32 | 0:PropertyType | O |

The *Identifier* is …

*IndexInList* is used to enumerate list elements in the subtypes.

## ProductionJobType Definition

The *ProductionJobType* provides aggregated production data for running a sequence to produce several parts after one preparation mounting.

Examples for such a mounting are putting four raw parts on a pallet for a machining center, setting up the fitting diameter bars in a turning center bar feeder or loading a sheet metal blank from which hundreds of parts can be cut/punched. This sequence shall represent several parts which will usually (but not always) be several identical products. A job may be executed several times.

The *ProductionJobType* is formally defined in Table 21.

Table 21 - ProductionJobType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionJobType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseProductionType* defined in 7.12 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasProperty | Variable | 1:CustomerOrderId | 0:String | 0:PropertyType | O |
| 0:HasProperty | Variable | 1:IsSerialProduction | 0:Boolean | 0:PropertyType | M |
| 0:HasProperty | Variable | 1:OrderId | 0:String | 0:PropertyType | O |
| 0:HasComponent | Variable | 1:PartsCompleted | 0:UInt32 | 0:BaseDataVariableType | O |
| 0:HasComponent | Object | 1:PartsInProduction |  | 0:BaseObjectType | O |
|  | Variable | 1:PartsInProduction.<Part> |  | 1:ProductionPartType | MandatoryPlaceholder |
| 0:HasComponent | Variable | 1:PartsPlanned | 0:UInt32 | 0:BaseDataVariableType | O |
| 0:HasComponent | Object | 1:ProductionPrograms |  | 0:BaseObjectType | M |
|  | Variable | 1:ProductionPrograms.<Program> |  | 1:ProductionProgramType | MandatoryPlaceholder |
| 0:HasProperty | Variable | 1:RunsCompleted | 0:UInt32 | 0:PropertyType | M |
| 0:HasProperty | Variable | 1:RunsPlanned | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Object | 1:State |  | 1:ProductionJobStateMachineType | M |

The *CustomerOrderId* is used to …

*IsSerialProduction* indicates …

The *OrderId* is …

*PartsCompleted* counts the finished parts in the job.

*PartsInProduction* contains a list of *ProductionPartType* nodes related to the job. It is a list of the parts, which are currently present in the machine's work area, typically for manufacturing purposes. This does not include parts in the supply chain.

*PartsInProduction* is formally defined in Table 23.

Table 22 – PartsInProduction object Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | *1:PartsInProduction* | | | | |
|  |  | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| ComponentOf the ProductionJobType defined in 7.13 | | | | | |
| 0:HasTypeDefinition | ObjectType | *0:BaseObjectType* | Defined in OPC 10000-5 | | |
| 0:HasOrderedComponent | Object | 1:<Part> |  | ProductionPartType | MandatoryPlaceholder |

This list is made up of at least one <*Part*> instance of *ProductionPartType*. The ordering of the parts is displayed using the *HasOrderedComponent* *Reference* and the *NumberInList* component of the Part instance inherited from the *BaseProductionType*. The underlying ordering is the production sequence of the parts, the part produced the earliest shall have the smallest number and appear first along the *OrderedComponents*.

*PartsPlanned* indicates …

*ProductionPrograms* contains a list of *ProductionProgramType* nodes related to the job. It is formally defined in Table 23. …

Table 23 – ProductionPrograms object Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | *1:ProductionPrograms* | | | | |
|  |  | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| ComponentOf the ProductionJobType defined in 7.13 | | | | | |
| 0:HasTypeDefinition | ObjectType | *0:BaseObjectType* | Defined in OPC 10000-5 | | |
| 0:HasOrderedComponent | Object | 1:<Program> |  | ProductionProgramType | MandatoryPlaceholder |

This list is made up of at least one <*Program*> instance of *ProductionProgramType*. The ordering of the programs is displayed using the *HasOrderedComponent* *Reference* and the *NumberInList* component of the Program instance inherited from the *BaseProductionType*. The underlying ordering is the call sequence of the programs. The program called first shall have the smallest number and appear first along the *OrderedComponents*. If any program is called several times in a single job, only the first call is relevant to the ordering.

*RunsCompleted* is a counter that …

*RunsPlanned* indicates …

State is an instance representation of the *ProductionJobStateMachineType*. It indicates …

## ProductionPartType Definition

The *ProductionPartType* represents a part. A Part is the workpiece of the machine which is treated in the purpose of the machine.

This may be for the purpose of machining, measuring or others, depending on the machine type.

The *ProductionPartType* is formally defined in Table 24.

Table 24 - ProductionPartType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionPartType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseProductionType* defined in 7.12 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasProperty | Variable | 1:CustomerOrderId | 0:String | 0:PropertyType | O |
| 0:HasComponent | Variable | 1:PartQuality | 1:PartQuality | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:ProcessIrregularity | 1:ProcessIrregularity | 0:BaseDataVariableType | M |
| 0:HasComponent | Object | 1:State |  | 1:ProductionPartStateMachineType | O |

The *CustomerOrderId* is …

*PartQuality* indicates …

*ProcessIrregularity* is used to …

*State* is an instance representation of the *ProductionPartStateMachineType*. It indicates …

## ProductionProgramType Definition

The *ProductionProgramType* is the representation of a program. A program is a list of operations that the controllers performs in sequence. It's usually a machine-readable file which is needed for the controller to fulfill the job.

The *ProductionProgramType* is formally defined in Table 25.

Table 25 - ProductionProgramType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionProgramType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseProductionType* defined in 7.12 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Object | 1:State |  | 1:ProductionProgramStateMachineType | O |

*State* is an instance representation of the *ProductionProgramStateMachineType*. It shows …

## BaseToolType Definition

The *BaseToolType* serves as a supertype to the *MultiToolType* and the *ToolType*. It is an abstract type, meaning it is not instantiated, only the subtypes are.

The *BaseToolType* is formally defined in Table 26.

Table 26 - BaseToolType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:BaseToolType | | | | |
| IsAbstract | True | | | | |
| **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:HasSubtype | ObjectType | 1:MultiToolType | Defined in 7.17 | | |
| 0:HasSubtype | ObjectType | 1:ToolType | Defined in 7.18 | | |

Other than *HasSubtype* *References*, the *BaseToolType* does not have any *References*.

## MultiToolType Definition

The *MultiToolType* represents a unit of different tools, usually used in order to have several tools available in-process without requiring explicit tool-changes.

Typical applications are in turning, when one indexed position of the tool revolver holds several outer-diameter cutting inserts and boring tools, such that a tool change process can quickly complete by merely readjusting the NC setpoint position tool compensation.

The *MultiToolType* is formally defined in Table 27.

Table 27 - MultiToolType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:MultiToolType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseToolType* defined in 7.16 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Object | 1:<Tool> | 1:ToolType |  | OptionalPlaceholder |
| 0:HasComponent | Variable | 1:UniqueId | 0:String | 0:BaseDataVariableType | M |

<*Tool*> is a placeholder for Instances of *ToolType*. Using this placeholder, the individual *ToolType* instances making up the *MultiTool* can be represented in the information model. …

*UniqueId* is …

## ToolType Definition

The *ToolType* is the representation of a tool. Tools are exchangeable components used in a machine tool to execute the production process and may for example be drills, ball milling heads, cutting inserts, pinching tools and so forth.

May be a non-contact tool, for example a processing laser.

The *ToolType* is formally defined in Table 28.

Table 28 - ToolType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ToolType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseToolType* defined in 7.16 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:Active | 0:Boolean | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:Duplonumber | 0:UInt32 | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:LastUsage | 0:UtcTime | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | 1:Locked | 0:Boolean | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:ToolLife | 0:Number | 1:ToolLifeType | O |
| 0:HasComponent | Variable | 1:TypeId | 0:String | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:UniqueId | 0:String | 0:BaseDataVariableType | M |

*Active* indicates …

The *Duplonumber* is used to …

*LastUsage* refers to …

*Locked* indicates …

*ToolLife* shows …

*TypeId* is …

*UniqueId* is …

## EquipmentType Definition

The *EquipmentType* is used to structure information given in the *MachineToolType*. It contains the entry point to the list of tools available to the machine tool.

The *EquipmentType* is formally defined in Table 29.

Table 29 - EquipmentType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:EquipmentType | | | | |
| 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 | 1:Tools |  | 1:ToolListType | M |

*Tools* is the entry point to the list of *BaseToolType* subtype instances in the machine tool.

## MachineToolsFolderType

The *MachineToolsFolderType* is used to define the entry point to the umati namespace. It is instantiated once in every umati server. For Information about the instance declaration, see chapter 11.1.

The *MachineToolsFolderType* is formally defined in Table 30.

Table 30 - MachineToolsFolderType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:MachineToolsFolderType | | | | |
| 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. | | | | | |

The *MachineToolsFolderType* defines no *References*.

## IdentificationType Definition

The *IdentificationType* of the umati information model holds static data which shall uniquely identify a machine among a pool of the machine operating entity.

It should typically remain constant over several weeks or even over the entire product life of the machine. Contents shall be of relevance to the machine operator or OEM.

The *IdentificationType* is formally defined in Table 31.

Table 31 - IdentificationType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:IdentificationType | | | | |
| 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 | 1:Machine |  | 1:MachineIdentificationType | M |
| 0:HasComponent | Object | 1:Software |  | 1:SoftwareIdentificationType | M |

*Machine* is an instance of the *MachineIdentificationType*. This instance contains the machine tool’s identification information. …

*Software* is an instance of the *SoftwareIdentificationType*. This instance contains the machine tool’s software identification information. …

## LampType Definition

The *LampType* defines the representation of a Lamp. A Lamp is an element of the stacklight (sometimes referred to as signal light) which is found on virtually all production machinery.

Color codes of the semantics may often be aligned to DIN EN60204-1, but definitions specific to a machine-operating company or an OEM commonly occur.

The *LampType* is formally defined in Table 32.

Table 32 - LampType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:LampType | | | | |
| 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 | 1:Color | 1:LampColor | 0:PropertyType | M |
| 0:HasProperty | Variable | 1:PositionFromBottom | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Variable | 1:Status | 1:LampStatus | 0:BaseDataVariableType | M |

*Color* refers to …[the color the lamp has when on. “Normal” stacklight color should be obvious. Special implementations with color-changing elements should be modelled with the same “traditional” stacklight in mind, in the closest way fitting, e.g. the color the lamp currently has/last had (if off).]

*PositionFromBottom* indicates the lamps position in a stacklight. The numbering is intended to start from the lowest lamp in the stacklight counting upwards. …

*Status* refers to …[lamp status enum: on/off/blinking]

## MachineToolType Definition

The *MachineToolType* represents the entire machine interface of the information model. It is the entry point to the umati OPC UA interface. It gives a basic structure to the interface. An instance of this type aggregates all information related to one machine tool.

At least one such instance must be present in a umati server. All instances of *MachineToolType* have to be referenced from the *MachineTools* node (see section 11.1).

The *MachineToolType* is formally defined in Table 33.

Table 33 - MachineToolType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:MachineToolType | | | | |
| 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 | 1:Equipment |  | 1:EquipmentType | M |
| 0:HasComponent | Object | 1:Identification |  | 1:IdentificationType | M |
| 0:HasComponent | Object | 1:Monitoring |  | 1:MonitoringType | M |
| 0:HasComponent | Object | 1:Notification |  | 1:NotificationType | M |
| 0:HasComponent | Object | 1:Production |  | 1:ProductionType | M |

*Equipment* groups all the data related to items that are not an inherent part of the machine tool, but can be installed temporarily. At the moment, the *Equipment* Component contains information about the tools the machine tool is equipped with.

*Identification* groups all information related to identifying the machine tool.

*Monitoring* groups data relevant to monitor the machine tool or a component of the machine tool.

*Notification* sends the messages of the machine tool and contains the prognoses.

*Production* groups all information about production jobs.

## MonitoringType Definition

The *MonitoringType* is used to structure information given in the *MachineToolType*. It contains the monitoring information of the machine tool ...

The *MonitoringType* is formally defined in Table 35.

Table 35 - MonitoringType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:MonitoringType | | | | |
| 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 | 1:<MonitoredElement> |  | 1:ElementMonitoringType | OptionalPlaceholder |
| 0:HasComponent | Object | 1:MachineTool |  | 1:MachineOperationMonitoringType | M |
| 0:HasComponent | Object | 1:Stacklight |  | 1:StacklightType | O |

<*MonitoredElement*> is an optional Placeholder for *ElementMonitoringType* instances. This allows for any number of such instances as a component of the *MonitoredElementsListType*. …

*MachineTool* is …

*Stacklight* contains the information about a stacklight’s composition and status.

## NotificationType Definition

The *NotificationType* is used to structure information given in the *MachineToolType*. It groups the messages and alerts of the machine tool and contains the prognoses for the machining operation.

The *NotificationType* is formally defined in Table 36.

Table 36 - NotificationType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:NotificationType | | | | |
| 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 | 1:Messages |  | 0:BaseObjectType | M |
| 0:HasComponent | Object | 1:Prognoses |  | 1:PrognosisListType | M |

*Messages* is used to send out errors, warnings and messages. It is formally defined in Table 37.

Table 37 – Messages object Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | *1:Messages* | | | | |
|  |  | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| ComponentOf the NotificationType defined in 7.26. | | | | | |
| 0:HasTypeDefinition | ObjectType | *0:BaseObjectType* | Defined in OPC 10000-5 | | |
| 0:GeneratesEvent | ObjectType | 1:AlertConditionType |  |  |  |
| 0:GeneratesEvent | ObjectType | 1:NotificationEventType |  |  |  |

To differentiate between errors, warnings and messages on the interface, the following convention shall be used:

Errors have a *Severity* between 661 and 1000 and are using an *AlertConditionType*.

Warnings have a *Severity* between 331 and 660 and are using an *AlertConditionType*.

Messages have a *Severity* lower or equal to 330 and are using a *NotificationEventType*.

*Prognoses* contains a list of the current prognoses for machine tool operation.

## ProductionJobListType Definition

The *ProductionJobListType* is a list of all job elements a specific machine knows about. I.e. all jobs which were somehow transferred to the machine.

Note that the single Machine may get the Job for an operation sequence when machining parts are addressed to this specific machine only, depending on the production workflow on-site.

The *ProductionJobListType* is formally defined in Table 38.

Table 38 - ProductionJobListType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionJobListType | | | | |
| 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:HasOrderedComponent | Object | 1:<ProductionJob> |  | 1:ProductionJobType | OptionalPlaceholder |
| 0:HasProperty | Variable | 1:NodeVersion | String | 0:PropertyType | O |
| 0:GeneratesEvent | ObjectType | 0:BaseModelChangeEventType |  |  |  |

<*ProductionJob*> is a placeholder for any number of *ProductionJobType* instances. To indicate the order of jobs on the machine too, the *IndexInList* parameter of the *ProductionJobType* is used. This index shall be 1 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 *IndexInList* has to be adjusted for all following *ProductionJobType* instances in the list, such that the *IndexInList* elements always form a sequential series of numbers. If the order of production jobs is determined in any other way on a specific machine tool, the representation in the *ProductionJobListType* shall be done in such a way that the interpretation as a sequential list is not misleading.

The *NodeVersion* is intended to be used in conjunction with the *BaseModelChangeEventType* in the way defined in OPC 10000-3.

The *BaseModelChangeEventType* is intended to be used as defined in OPC 10000-3. In the context of the *ProdcutionJobListType* it indicates that a node has been added to or deleted from the *ProductionJobListType*, correlating with the information that a job has been added or deleted to the machine tool’s production plan.

## ProductionType Definition

The *ProductionType* is used to structure information given in the *MachineToolType*. It groups the information about the production plan and the production statistics.

It is formally defined in Table 39.

Table 39 - ProductionType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1: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 | Object | 1:ProductionPlan |  | 1:ProductionJobListType | M |
| 0:HasComponent | Object | 1:Statistics |  | 0:BaseObjectType | O |

*ProductionPlan* is …

*Statistics* is …

It is formally defined in Table 40.

Table 40 – Statistics Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | *1:Statistics* | | | | |
|  |  | | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| ComponentOf the ProductionType defined in 7.28. | | | | | |
| 0:HasTypeDefinition | ObjectType | *0:BaseObjectType* | Defined in OPC 10000-5 | | |
| 0:HasComponent | Variable | 1:PartCounter | UInt32 | BaseDataVariableType | M |

*PartCounter* is …

## PrognosisListType Definition

The *PrognosisListType* is a structuring node to collect estimations of the machine control system about future times (or timespans) when certain events may occur.

It is formally defined in Table 41.

Table 41 - PrognosisListType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:PrognosisListType | | | | |
| 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 | 1:<Prognosis> |  | 1:PrognosisType | OptionalPlaceholder |
| 0:HasComponent | Object | 1:EarliestUserInteraction |  | 1:PrognosisType | O |
| 0:HasProperty | Variable | 1:NodeVersion | String | 0:PropertyType | O |
| 0:GeneratesEvent | ObjectType | 0:BaseModelChangeEventType |  |  |  |

<*Prognosis*> is an optional placeholder for *PrognosisType* nodes. Thus, the *PrognosisListType* can have any number of prognoses as components. If the number of prognoses in this list changes during the runtime of the OPC UA server, the *NodeVersion* and *BaseModelChangeEventType* can be used to indicate those changes. The usage of *NodeVersion* and *BaseModelChangeEventType* is defined in OPC 10000-3.

As content for the NodeVersion property, a timestamp of the moment the node structure was changed using the *DateTime* value defined in OPC 10000-6 converted to a string shall be used.

*EarliestUserInteraction* is used to …

## PrognosisType Definition

The *PrognosisType* is the supertype to more specific prognosis types. …

It is formally defined in Table 42.

Table 42 - PrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:PrognosisType | | | | |
| 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:HasSubtype | ObjectType | 1:MaintenancePrognosisType | Defined in 7.31 | | |
| 0:HasSubtype | ObjectType | 1:ManualActivityPrognosisType | Defined in 7.32 | | |
| 0:HasSubtype | ObjectType | 1:PartUnloadPrognosisType | Defined in 7.33 | | |
| 0:HasSubtype | ObjectType | 1:ProcessChangeoverPrognosisType | Defined in 7.34 | | |
| 0:HasSubtype | ObjectType | 1:ProductionJobEndPrognosisType | Defined in 7.35 | | |
| 0:HasSubtype | ObjectType | 1:RawPartLoadPrognosisType | Defined in 7.36 | | |
| 0:HasSubtype | ObjectType | 1:ToolChangePrognosisType | Defined in 7.37 | | |
| 0:HasSubtype | ObjectType | 1:UtilityChangePrognosisType | Defined in 7.38 | | |
| 0:HasProperty | Variable | 1:PredictedTime | 0:UtcTime | 0:PropertyType | M |

*PredictedTime* is used to …

## MaintenancePrognosisType Definition

The *MaintenancePrognosisType* is a prognosis of the control system indicating at which time in the future a specific maintenance action may become necessary.

Examples may be oil changes, filter mat replacements or regular checks. Reliability will rely on the specific case and cannot be guaranteed to be precise.

The *MaintenancePrognosisType* is formally defined in Table 44.

Table 43 - MaintenancePrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:MaintenancePrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:Activity | 0:String | 0:BaseDataVariableType | M |

*Activity* is …

## ManualActivityPrognosisType Definition

The *ManualActivityPrognosisType* is a prognosis of the control system indicating at which time in the future a manual intervention may become necessary.

Examples may be manual tool changes for deep boring tools which do not fit in the tool magazine. Reliability will rely on the specific case and cannot be guaranteed to be precise.

The *ManualActivityPrognosisType* is formally defined in Table 44.

Table 44 - ManualActivityPrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ManualActivityPrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:Activity | 0:String | 0:BaseDataVariableType | M |

*Activity* is …

## PartUnloadPrognosisType Definition

The *PartUnloadPrognosisType* is …

It is formally defined in Table 45.

Table 45 - PartUnloadPrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:PartUnloadPrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:Location | 0:String | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | 1:PartIdentifier | 0:String | 0:BaseDataVariableType | O |

*Location* is …

*PartIdentifier* shall be a copy of the *Identifier* Property of the *ProductionPartType* instance the prognosis relates to.

## ProcessChangeoverPrognosisType Definition

The *ProcessChangeoverPrognosisType* is …

It is formally defined in Table 46.

Table 46 - ProcessChangeoverPrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProcessChangeoverPrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:Activity | 0:String | 0:BaseDataVariableType | M |

*Activity* is …

## ProductionJobEndPrognosisType Definition

The *ProductionJobEndPrognosisType* is the estimated timespan until the end of the current Job.

It is formally defined in Table 47.

Table 47 - ProductionJobEndPrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionJobEndPrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:SourceIdentifier | 0:String | 0:BaseDataVariableType | M |

The *SourceIdentifier* Variable shall contain a copy of the content of the *Identifier* property belonging to the *ProductionJobType* the prognosis refers to.

## RawPartLoadPrognosisType Definition

The *RawPartLoadPrognosisType* is …

It is formally defined in Table 49.

Table 48 - RawPartLoadPrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:RawPartLoadPrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:Location | 0:String | 0:BaseDataVariableType | M |

*Location* is …

## ToolChangePrognosisType Definition

The *ToolChangePrognosisType* is …

It is formally defined in Table 49.

Table 49 - ToolChangePrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ToolChangePrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Variable | 1:Destination | 0:String | 0:BaseDataVariableType | M |

*Destination* refers to …

## UtilityChangePrognosisType Definition

The *UtilityChangePrognosisType* is …

It is formally defined in Table 50.

Table 50 - UtilityChangePrognosisType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:UtilityChangePrognosisType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *PrognosisType* defined in 7.30 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |

The *UtilityChangePrognosisType* does not define any other *References*.

## StacklightType Definition

The stacklight is a visual machine state indicator. It consists of one or more lamps stacked on top of one another, each having another color.

The combination of on/off/blinking lights corresponds to a machine state. The ordering of the colors is counted from bottom to top and from left to right. A node of *StacklightType* is the virtual representation of the physical stacklight of the machine.

The *StacklightType* is formally defined in Table 51.

Table 51 - StacklightType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:StacklightType | | | | |
| 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:HasOrderedComponent | Object | 1:<Lamp> |  | 1:LampType | MandatoryPlaceholder |

<*Lamp*> is a mandatory placeholder for a *LampType* instance. An instance of the *StacklightType* shall have as many *LampType* instances as *Components* as there are individual lamps in the stacklight. …

The ordering of the lamps is both shown with the *HasOrderedComponent* *Reference* and the *PositionFromBottom* Property of the *LampType*. Both ordering indications shall match exactly. …

## ProductionJobStateMachineType

The *ProductionJobStateMachineType* shows …



Figure 6: The States and Transitions of the ProductionJobStateMachineType

It is formally defined in Table 52.

Table 52 - ProductionJobStateMachineType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionJobStateMachineType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *FiniteStateMachineType* defined in OPC 10000-5 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Object | 1:Aborted |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:AbortedToInitialized |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Variable | 1:CurrentState | 0:LocalizedText | 0:FiniteStateVariableType | M |
|  | Variable | 1:CurrentState.Number | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Object | 1:Ended |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:EndedToInitialized |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:Initialized |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:InitializedToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:InitializedToRunning |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:Interrupted |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:InterruptedToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:InterruptedToRunning |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Variable | 1:LastTransition | 0:LocalizedText | 0:FiniteTransitionVariableType | M |
|  | Variable | 1:LastTransistion.Number | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Object | 1:Running |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:RunningToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:RunningToEnded |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:RunningToInterrupted |  | 0:TransitionType | No ModellingRule |
| 0:GeneratesEvent | ObjectType | 1:InterruptionConditionType |  |  |  |

The state *Aborted* is …

*Ended* is reached when …

*Initialized* is …

*Interrupted* …

*Running* …

Both the *CurrentState* and the *LastTransition* *Components* have their optional *Property* *Number* overridden to be mandatory. The states shall have the numbers indicated in Table 53. The *Number* *Property* of *CurrentState* and *LastTransition* shall use those same numbers for the respective state.

Table 53: State Numbers for the ProductionJobStateMachineType

|  |  |
| --- | --- |
| **State** | **Number** |
| Initialized | 0 |
| Running | 1 |
| Ended | 2 |
| Interrupted | 3 |
| Aborted | 4 |

Table 54 - ProductionJobStateMachineType Transitions

|  |  |  |  |
| --- | --- | --- | --- |
| **BrowseName** | **References** | **BrowseName** | **TypeDefinition** |
|  | | | |
| **Transitions** | | | |
| 1:AbortedToInitialized | 0:FromState | Aborted | StateType |
|  | 0:ToState | 1:Initialized | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:EndedToInitialized | 0:FromState | 1:Ended | StateType |
|  | 0:ToState | 1:Initialized | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InitializedToAborted | 0:FromState | 1:Initialized | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InitializedToRunning | 0:FromState | 1:Initialized | StateType |
|  | 0:ToState | 1:Running | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InterruptedToAborted | 0:FromState | 1:Interrupted | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InterruptedToRunning | 0:FromState | 1:Interrupted | StateType |
|  | 0:ToState | 1:Running | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:RunningToAborted | 0:FromState | 1:Running | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:RunningToEnded | 0:FromState | 1:Running | StateType |
|  | 0:ToState | 1:Ended | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:RunningToInterrupted | 0:FromState | 1:Running | StateType |
|  | 0:ToState | 1:Interrupted | StateType |
|  | 0:HasEffect | 1:ProductionJobTransitionEventType | *Event* |
|  | 0:HasCause |  |  |

## ProductionPartStateMachineType

The *ProductionPartStateMachineType* is …



Figure 7 : The States and Transitions of the ProductionPartStateMachineType

The *ProductionPartStateMachineType* is formally defined in Table 55.

Table 55 - ProductionPartStateMachineType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionPartStateMachineType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *FiniteStateMachineType* defined in OPC 10000-5 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Object | 1:Aborted |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:AbortedToScheduled |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Variable | 1:CurrentState | 0:LocalizedText | 0:FiniteStateVariableType | M |
|  | Variable | 1:CurrentState.Number | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Object | 1:Finished |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:FinishedToScheduled |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:Scheduled |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:ScheduledToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:ScheduledToProcessing |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:Interrupted |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:InterruptedToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:InterruptedToProcessing |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Variable | 1:LastTransition | 0:LocalizedText | 0:FiniteTransitionVariableType | M |
|  | Variable | 1:LastTransistion.Number | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Object | 1:Processing |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:ProcessingToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:ProcessingToFinished |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:ProcessingToInterrupted |  | 0:TransitionType | No ModellingRule |

Aborted is …

Finished is …

Scheduled is …

Interrupted is …

Processing is …

Both the *CurrentState* and the *LastTransition* *Components* have their optional *Property* *Number* overridden to be mandatory. The states shall have the numbers indicated in Table 53. The *Number* *Property* of *CurrentState* and *LastTransition* shall use those same numbers for the respective state.

Table 56: State Numbers for the ProductionPartStateMachineType

|  |  |
| --- | --- |
| **State** | **Number** |
| Scheduled | 0 |
| Processing | 1 |
| Finished | 2 |
| Interrupted | 3 |
| Aborted | 4 |

Table 57 - ProductionPartStateMachineType Transitions

|  |  |  |  |
| --- | --- | --- | --- |
| **BrowseName** | **References** | **BrowseName** | **TypeDefinition** |
|  | | | |
| **Transitions** | | | |
| 1:AbortedToScheduled | 0:FromState | 1:Aborted | StateType |
|  | 0:ToState | 1:Scheduled | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:FinishedToScheduled | 0:FromState | 1:Finished | StateType |
|  | 0:ToState | 1:Scheduled | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InterruptedToAborted | 0:FromState | 1:Interrupted | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InterruptedToProcessing | 0:FromState | 1:Interrupted | StateType |
|  | 0:ToState | 1:Processing | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:ProcessingToAborted | 0:FromState | 1:Processing | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:ProcessingToFinished | 0:FromState | 1:Processing | StateType |
|  | 0:ToState | 1:Finished | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:ProcessingToInterrupted | 0:FromState | 1:Processing | StateType |
|  | 0:ToState | 1:Interrupted | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:ScheduledToAborted | 0:FromState | 1:Scheduled | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:ScheduledToProcessing | 0:FromState | 1:Scheduled | StateType |
|  | 0:ToState | 1:Processing | StateType |
|  | 0:HasEffect | 1:ProductionPartTransitionEventType | *Event* |
|  | 0:HasCause |  |  |

## ProductionProgramStateMachineType

The *ProductionProgramStateMachineType* is …



Figure 8 : The States and Transitions of the ProductionProgramStateMachineType

The *ProductionProgramStateMachineType* is formally defined in Table 60.

Table 58 - ProductionProgramStateMachineType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ProductionProgramStateMachineType | | | | |
| IsAbstract | False | | | | |
| **References** | **Node Class** | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *FiniteStateMachineType* defined in OPC 10000-5 i.e. inheriting the InstanceDeclarations of that Node. | | | | | |
| 0:HasComponent | Object | 1:Aborted |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:AbortedToInitialized |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Variable | 1:CurrentState | 0:LocalizedText | 0:FiniteStateVariableType | M |
|  | Variable | 1:CurrentState.Number | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Object | 1:Ended |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:EndedToInitialized |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:Initialized |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:InitializedToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:InitializedToRunning |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:Interrupted |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:InterruptedToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:InterruptedToRunning |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Variable | 1:LastTransition | 0:LocalizedText | 0:FiniteTransitionVariableType | M |
|  | Variable | 1:LastTransistion.Number | 0:UInt32 | 0:PropertyType | M |
| 0:HasComponent | Object | 1:Running |  | 0:StateType | No ModellingRule |
| 0:HasComponent | Object | 1:RunningToAborted |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:RunningToEnded |  | 0:TransitionType | No ModellingRule |
| 0:HasComponent | Object | 1:RunningToInterrupted |  | 0:TransitionType | No ModellingRule |

The state *Aborted* is …

*Ended* is reached when …

*Initialized* is …

*Interrupted* …

*Running* …

Both the *CurrentState* and the *LastTransition* *Components* have their optional *Property* *Number* overridden to be mandatory. The states shall have the numbers indicated in Table 53. The *Number* *Property* of *CurrentState* and *LastTransition* shall use those same numbers for the respective state.

Table 59: State Numbers for the ProductionJobStateMachineType

|  |  |
| --- | --- |
| **State** | **Number** |
| Initialized | 0 |
| Running | 1 |
| Ended | 2 |
| Interrupted | 3 |
| Aborted | 4 |

Table 60 - ProductionProgramStateMachineType Transitions

|  |  |  |  |
| --- | --- | --- | --- |
| **BrowseName** | **References** | **BrowseName** | **TypeDefinition** |
|  | | | |
| **Transitions** | | | |
| 1:AbortedToInitialized | 0:FromState | Aborted | StateType |
|  | 0:ToState | 1:Initialized | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:EndedToInitialized | 0:FromState | 1:Ended | StateType |
|  | 0:ToState | 1:Initialized | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InitializedToAborted | 0:FromState | 1:Initialized | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InitializedToRunning | 0:FromState | 1:Initialized | StateType |
|  | 0:ToState | 1:Running | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InterruptedToAborted | 0:FromState | 1:Interrupted | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:InterruptedToRunning | 0:FromState | 1:Interrupted | StateType |
|  | 0:ToState | 1:Running | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:RunningToAborted | 0:FromState | 1:Running | StateType |
|  | 0:ToState | 1:Aborted | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:RunningToEnded | 0:FromState | 1:Running | StateType |
|  | 0:ToState | 1:Ended | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |
| 1:RunningToInterrupted | 0:FromState | 1:Running | StateType |
|  | 0:ToState | 1:Interrupted | StateType |
|  | 0:HasEffect | 1:ProductionProgramTransitionEventType | *Event* |
|  | 0:HasCause |  |  |

## ToolListType Definition

The *ToolListType* is a list of tools, where a tool may be a single tool or a multitool.

Multitools carry several tools on one tool magazine socket or one revolver index position and will be mounted into the machine as one prepared unit.

The *ToolListType* is formally defined in Table 61.

Table 61 - ToolListType Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | | |
| BrowseName | 1:ToolListType | | | | |
| 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 | 1:<Tool> |  | 1:BaseToolType | OptionalPlaceholder |
| 0:HasProperty | Variable | 1:NodeVersion | String | 0:PropertyType | Optional |
| 0:GeneratesEvent | ObjectType | 1:BaseModelChangeEventType |  |  |  |

<*Tool*> is an *OptionalPlaceholder* for nodes of *BaseToolType*. The tool list can thus contain any number of tools, including none.

The contents of the *ToolListType* instance can change during the *Server* runtime (e.g. if tools are inserted into the machine tool or removed from it). A change in the list can be indicated using the *NodeVersion* Property and the *BaseModelChangeEventType* in the way defined in OPC 10000-3.

As content for the NodeVersion property, a timestamp of the moment the node structure was changed using the *DateTime* value defined in OPC 10000-6 converted to a string shall be used.

# *…*OPC UA EventTypes

## AlertConditionType

The AlertConditionType is used to transport errors and warnings. …

It is formally defined in Table 62.

Table 62 –AlertConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:AlertConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *ConditionType* defined in OPC 10000-9 which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasProperty | Variable | | 1:Identifier | 0:String | 0:PropertyType | M |

The *Identifier* is used for the manufacturer defined error code. Often this is a numeric code whose meaning can be found in the manufacturer’s documentation for the machine tool.

## InterruptionConditionType

The *InterruptionConditionType* is a supertype for the *InterruptionExternalConditionType* and the *InterruptionProcessConditionType*. It is an abstract type.

The *InterruptionConditionType* is formally defined in Table 64.

Table 63 – InterruptionConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionConditionType | | | | |
| IsAbstract | | True | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *ConditionType* defined in OPC 10000-9, which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionExternalConditionType | Defined in 8.3 | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionProcessConditionType | Defined in 8.6 | | |

Other than *HasSubtype*, the *InterruptionConditionType* has no *References*.

## InterruptionExternalConditionType

The *InterruptionExternalConditionType* is a supertype to *InterruptionConditionTypes* that are external to the machining process. …?

It is formally defined in Table 64.

Table 64 – InterruptionExternalConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionExternalConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionConditionType* defined in 8.2, which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionByOperatorConditionType | Defined in 8.4 | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionSafetyConditionType | Defined in 8.5 | | |

Other than *HasSubtype*, the *InterruptionExternalConditionType* has no *References*.

## InterruptionByOperatorConditionType

The *InterruptionByOperatorConditionType* indicates …

It is formally defined in Table 65.

Table 65 – InterruptionByOperatorConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionByOperatorConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionExternalConditionType* defined in 8.3, which means it inherits the InstanceDeclarations of that Node. | | | | | | |

The *InterruptionByOperatorConditionType* has no *References*.

## InterruptionSafetyConditionType

The *InterruptionSafetyConditionType* is used to …

It is formally defined in Table 67.

Table 66 – InterruptionSafetyConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionSafetyConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionExternalConditionType* defined in 8.3, which means it inherits the InstanceDeclarations of that Node. | | | | | | |

The *InterruptionSafetyConditionType* has no *References*.

## InterruptionProcessConditionType

The *InterruptionProcessConditionType* is a supertype to all *ConditionTypes* that indicate a cause for the interruption that is internal to the machining process. …?

The *InterruptionProcessConditionType* is formally defined in Table 67.

Table 67 – InterruptionProcessConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionProcessConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionConditionType* defined in 8.2, which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionClampingConditionType | Defined in 8.7 | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionManualProcessStepConditionType | Defined in 8.8 | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionMeasurementConditionType | Defined in 8.9 | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionProcessIrregularityConditionType | Defined in 8.10 | | |
| 0:HasSubtype | ObjectType | | 1:InterruptionToolChangeConditionType | Defined in 8.11 | | |
| 0:HasProperty | Variable | | 1:IsAutomated | 0:Boolean | 0:PropertyType | M |

*IsAutomated* indicates …

## InterruptionClampingConditionType

The *InterruptionClampingConditionType* is …

It is formally defined in Table 68.

Table 68 – InterruptionClampingConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionClampingConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionProcessConditionType* defined in 8.6, which means it inherits the InstanceDeclarations of that Node. | | | | | | |

The *InterruptionClampingConditionType* has no *References*.

## InterruptionManualProcessStepConditionType

The *InterruptionManualProcessStepConditionType* is …

It is formally defined in Table 69.

Table 69 – InterruptionManualProcessStepConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionManualProcessStepConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionProcessConditionType* defined in 8.6, which means it inherits the InstanceDeclarations of that Node. | | | | | | |

The *InterruptionManualProcessStepConditionType* has no *References*.

## InterruptionMeasurementConditionType

The *InterruptionMeasurementConditionType* is …

It is formally defined in Table 70.

Table 70 – InterruptionMeasurementConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionMeasurementConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionProcessConditionType* defined in 8.6, which means it inherits the InstanceDeclarations of that Node. | | | | | | |

The *InterruptionMeasurementConditionType* has no *References*.

## InterruptionProcessIrregularityConditionType

The *InterruptionProcessIrregularityConditionType* is …

It is formally defined in Table 71.

Table 71 – InterruptionProcessIrregularityConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionProcessIrregularityConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionProcessConditionType* defined in 8.6, which means it inherits the InstanceDeclarations of that Node. | | | | | | |

The *InterruptionProcessIrregularityConditionType* has no *References*.

## InterruptionToolChangeConditionType

The *InterruptionToolChangeConditionType* is …

It is formally defined in Table 72.

Table 72 – InterruptionToolChangeConditionType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:InterruptionToolChangeConditionType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *InterruptionProcessConditionType* defined in 8.6, which means it inherits the InstanceDeclarations of that Node. | | | | | | |

The *InterruptionToolChangeConditionType* has no *References*.

## NotificationEventType

The *NotificationEventType* is used to send simple messages from the machine tool. …

It is formally defined in Table 73.

Table 73 –NotificationEventType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:NotificationEventType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseEventType* defined in OPC 10000-5 which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasProperty | Variable | | 1:Identifier | 0:String | 0:PropertyType | M |

*Identifier* is …

## ProductionJobTransitionEventType

The *ProductionJobTransitionEventType* is sent when a transition of the *ProductionJobStateMachineType* is triggered. It purposely contains all the properties and components of the *ProductionJobType* in order to transport its state at the moment of the transition. Using this mechanism, fast paced jobs can still be monitored precisely. …?

The *ProductionJobTransitionEventType* is formally defined in Table 74.

Table 74 – ProductionJobTransitionEventType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:ProductionJobTransitionEventType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *TransitionEventType* defined in OPC 10000-5 which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasProperty | Variable | | 1:CustomerOrderId | 0:String | 0:PropertyType | O |
| 0:HasProperty | Variable | | 1:IsSerialProduction | 0:Boolean | 0:PropertyType | M |
| 0:HasProperty | Variable | | 1:JobIdentifier | 0:String | 0:PropertyType | M |
| 0:HasProperty | Variable | | 1:OrderId | 0:String | 0:PropertyType | O |
| 0:HasComponent | Variable | | 1:PartsCompleted | 0:UInt32 | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | | 1:PartsPlanned | 0:UInt32 | 0:BaseDataVariableType | M |
| 0:HasProperty | Variable | | 1:RunsCompleted | 0:UInt32 | 0:PropertyType | M |
| 0:HasProperty | Variable | | 1:RunsPlanned | 0:UInt32 | 0:PropertyType | M |

All *Properties* and *Components* in Table 74 are described in 7.13 for the *ProductionJobType*. Their values in the *ProductionJobTransitionEventType* shall be copies of those values in the moment of the transition.

## ProductionPartTransitionEventType

The *ProductionPartTransitionEventType* is sent when a transition of the *ProductionPartStateMachineType* is triggered. It purposely contains all the properties and components of the *ProductionPartType* in order to transport its state at the moment of the transition. Using this mechanism, fast paced production of parts can still be monitored precisely. …?

The *ProductionPartTransitionEventType* is formally defined in Table 75.

Table 75 – ProductionPartTransitionEventType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:ProductionPartTransitionEventType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *TransitionEventType* defined in OPC 10000-5 which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasProperty | Variable | | 1:CustomerOrderID | 0:String | 0:PropertyType | O |
| 0:HasProperty | Variable | | 1:PartIdentifier | 0:String | 0:PropertyType | M |
| 0:HasComponent | Variable | | 1:PartQuality | 1:PartQuality | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | | 1:ProcessIrregularity | 1:ProcessIrregularity | 0:BaseDataVariableType | M |

All *Properties* and *Components* in Table 75 are described in 7.14 for the *ProductionPartType*. Their values in the *ProductionPartTransitionEventType* shall be copies of those values in the moment of the transition.

## ProductionProgramTransitionEventType

The *ProductionProgramTransitionEventType* is sent when a transition of the *ProductionProgramStateMachineType* is triggered. …

The *ProductionProgramTransitionEventType* is formally defined in Table 76.

Table 76 – ProductionProgramTransitionEventType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | 1:ProductionProgramTransitionEventType | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *TransitionEventType* defined in OPC 10000-5 which means it inherits the InstanceDeclarations of that Node. | | | | | | |
| 0:HasProperty | Variable | | 1:ProductionProgramIdentifier | 0:String | 0:PropertyType | M |

*ProductionProgramIdentifier* is …

# OPC UA VariableTypes

## OverrideItemType

The *OverrideItemType* defines …

It is formally defined in Table 77.

Table 77 – OverrideItemType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | OverrideItemType | | | | |
| IsAbstract | | False | | | | |
| ValueRank | | −1 | | | | |
| DataType | | Double | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *AnalogItemType* defined in OPC 10000-8 | | | | | | |
| 0:HasProperty | Variable | | 1:EngineeringUnits | 0:EUInformation | 0:PropertyType | M |

*EngineeringUnits* is inherited from the *AnalogItemType* and overridden to have the *ModellingRule* *Mandatory*.

## ToolLifeType

The *ToolLifeType* …

It is formally defined in Table 78.

Table 78 – ToolLifeType Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | ToolLifeType | | | | |
| IsAbstract | | False | | | | |
| ValueRank | | −1 | | | | |
| DataType | | Number | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the *BaseDataVariableType* defined in OPC 10000-5 | | | | | | |
| 0:HasComponent | Variable | | 1:EngineeringUnits | 0:EUInformation | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | | 1:StartValue | 0:Number | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | | 1:WearLimit | 0:Number | 0:BaseDataVariableType | O |
| 0:HasComponent | Variable | | 1:WearMeasurement | 1:WearIndication | 0:BaseDataVariableType | M |
| 0:HasComponent | Variable | | 1:WearWarning | 0:Number | 0:BaseDataVariableType | O |

*EngineeringUnits* is …

*StartValue* is …

*WearLimit* is …

*WearMeasurement* is …

*WearWarning* is …

# OPC UA DataTypes

## ChannelState

This enumeration …. The enumeration is defined in Table 79.

Table 79 – ChannelState Definition

|  |  |
| --- | --- |
| Name | Description |
| Active\_0 | <EnumValue1Description> |
| Interrupted\_1 | <EnumValue2Description> |
| Reset\_2 | <EnumValue4Description> |

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

Table 80 – ChannelState Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | ChannelState | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## ControlMode

This enumeration …. The enumeration is defined in Table 81.

Table 81 – ControlMode Definition

|  |  |
| --- | --- |
| Name | Description |
| Undefined\_0 | <EnumValue1Description> |
| Automatic\_1 | <EnumValue2Description> |
| AutoSingleStep\_2 | <EnumValue4Description> |
| MdaMdi\_3 |  |
| JogManual\_4 |  |
| JogIncrement\_5 |  |
| Edit\_6 |  |

Its representation in the *AddressSpace* is defined in Table 82.

Table 82 – ControlMode Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | ControlMode | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## LampColor

This enumeration …. The enumeration is defined in Table 83.

Table 83 – LampColor Definition

|  |  |
| --- | --- |
| Name | Description |
| Green\_0 | This value indicates a green lamp color. |
| Red\_1 | This value indicates a red lamp color. |
| Yellow\_2 | This value indicates a yellow or orange lamp color. |
| Blue\_3 | This value indicates a blue lamp color. |
| White\_4 | This value indicates a white lamp color. |

Its representation in the *AddressSpace* is defined in Table 84.

Table 84 – LampColor Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | LampColor | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## LampStatus

This enumeration …. The enumeration is defined in Table 85.

Table 85 – LampStatus Definition

|  |  |
| --- | --- |
| Name | Description |
| Off\_0 | This value indicates a lamp switched on and emitting light. |
| On\_1 | This value indicates a lamp switched off. |
| Blinking\_2 | This value indicates intentional blinking of the lamp (only used if the lamp uses blinking to transport some information that differs from the “Off” and “On” states). |

Its representation in the *AddressSpace* is defined in Table 86.

Table 86 – LampStatus Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | LampStatus | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## NCProcessing

This enumeration …. The enumeration is defined in Table 87.

Table 87 – NCProcessing Definition

|  |  |
| --- | --- |
| Name | Description |
| Undefined\_0 | <EnumValue1Description> |
| SingleStep\_1 | <EnumValue2Description> |
| BlockSequence\_2 | <EnumValue4Description> |

Its representation in the *AddressSpace* is defined in Table 88.

Table 88 – NCProcessing Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | NCProcessing | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## PartQuality

This enumeration …. The enumeration is defined in Table 89.

Table 89 – PartQuality Definition

|  |  |
| --- | --- |
| Name | Description |
| CapabilityUnavailable\_0 | <EnumValue1Description> |
| Good\_1 | <EnumValue2Description> |
| Bad\_2 | <EnumValue4Description> |
| NotYetMeasured\_3 |  |

Its representation in the *AddressSpace* is defined in Table 90.

Table 90 – PartQuality Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | PartQuality | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## ProcessIrregularity

This enumeration …. The enumeration is defined in Table 91.

Table 91 – ProcessIrregularity Definition

|  |  |
| --- | --- |
| Name | Description |
| CapabilityUnavailable\_0 | <EnumValue1Description> |
| Detected\_1 | <EnumValue2Description> |
| NotDetected\_2 | <EnumValue4Description> |
| NotYetDetermined\_3 |  |

Its representation in the *AddressSpace* is defined in Table 92.

Table 92 – ProcessIrregularity Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | ProcessIrregularity | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## SafetyMode

This enumeration …. The enumeration is defined in Table 93.

Table 93 – SafetyMode Definition

|  |  |
| --- | --- |
| Name | Description |
| Other\_0 | <EnumValue1Description> |
| Automatic\_1 | <EnumValue2Description> |
| Setup\_2 | <EnumValue4Description> |
| Process\_observation\_3 |  |
| Process\_observation\_no\_confirmation\_4 |  |

Its representation in the *AddressSpace* is defined in Table 94.

Table 94 – SafetyMode Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | SafetyMode | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

## WearIndication

Wear is the state of decay/ usage of a tool. Wear can be measured in usage e.g. number of times the tool has been changed into the spindle, minutes of run time or deviation of a defined geometry.

This enumeration …. The enumeration is defined in Table 95.

Table 95 – WearIndication Definition

|  |  |
| --- | --- |
| Name | Description |
| Time\_0 | <EnumValue1Description> |
| NumberOfParts\_1 | <EnumValue2Description> |
| NumberOfUsages\_2 | <EnumValue4Description> |
| Distance\_3 |  |
| Length\_4 |  |
| Diameter\_5 |  |

Its representation in the *AddressSpace* is defined in **Fehler! Verweisquelle konnte nicht gefunden werden.**.

Table 96 – WearIndication Definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | | | |
| BrowseName | | WearIndication | | | | |
| IsAbstract | | False | | | | |
| **References** | **NodeClass** | | **BrowseName** | **DataType** | **TypeDefinition** | **Other** |
| Subtype of the Enumeration type defined in OPC 10000-3 | | | | | | |
| 0:HasProperty | Variable | | 0:EnumStrings | 0:LocalizedText [] | 0:PropertyType |  |

# Instances

## MachineTools object

An entry point to locate machines modeled with the umati model is provided. It shall directly reference all instances of *MachineToolType* with a hierarchical reference. Each umati server is expected to have exactly one *MachineTools* entry point. There can be one or more *MachineToolType* instances.

Examples:

* The UA Server represents one Machine Tool. In this case, the *MachineTools* Folder contains one Instance of *MachineToolType*.
* The UA Server represents several Machine Tools. The *MachineTools* Folder contains all *MachineToolType* Instances the Server provides access to.

The *MachineTools* Node is formally defined in Table 97.

Table 97 – MachineTools object Definition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Value** | | | |
| BrowseName | *1:MachineTools* | | | |
|  |  | | | |
| **References** | **NodeClass** | **BrowseName** | **DataType** | **TypeDefinition** |
| OrganizedBy by the Objects Folder defined in OPC 10000-5 | | | | |
| 0:HasTypeDefinition | ObjectType | *1:MachineToolsFolderType* | Defined in 7.20 | |
|  |  |  |  |  |

# Profiles and ConformanceUnits

*Profiles* and *ConformanceUnits* break functionality into testable groups. All companion specification shall include at least one *Profile*/*Facet*. If there are any groupings of functionality that not all *Servers*/*Client* would implement then multiple *Profile*/*Facet* are encouraged. A *ConformanceUnit* should describe a testable unit. A single *ConformanceUnit* is tested as a unit so all items covered by it must be support or the *ConformanceUnit* will fail. *ConformanceUnits* can be included in multiple *Profiles*, thus they are declared in their own table.

The name of the *Profile* should end with *Facet* or *Profile*. A *Facet* is a grouping of functionality that must also be paired with other *Facets* to create a running *Server* or *Client*. A *Profile* is all inclusive, in that is the *Profile* is implemented no additional functionality would be required to have a running application.

**<short name>**

A <short name> is required for each companion specification to assure uniqueness of string identifiers. It precedes the names of profiles and conformance units and is included in URIs and URLs defined in a companion specification.

A <short name> is all caps if an acronym, otherwise camel case.

Exception if the short name is a trademark. Use trademark casing.

## Conformance Units

This chapter defines the corresponding *Conformance Units* for the OPC UA Information Model for <title>.

Table 98 – Conformance Units for <Title>

| **Category** | **Title** | **Description** |
| --- | --- | --- |
| Server | <short name> <Function1> | Supports the base functionality defined in <Title> Information Model. This includes….. |
| Server | <short name> <Function2> | Supports the …... |
| Server | <short name> <Function3> | Supports the …... |
| Client | <short name> Client <Function1> | The client can make use of the …... |

Typically, *Client* *ConformanceUnits* describe the use of a function, but they do not need to match 1 to 1 with *Server* *ConformanceUnits*. They might also reference to other categories defined in Part 7 (Pub, Sub, GDS…). For larger companion specifications, there might be separate tables for *Client* *ConformanceUnits*, *Server* *ConformanceUnits*, etc.

## Profiles

* + 1. Profile list

Table 99 lists all Profiles defined in this document and defines their URIs.

Table 99 – Profile URIs for <Title>

| **Profile** | **URI** |
| --- | --- |
| <short name> <Prf1name> Server Profile | [http://opcfoundation.org/UA-Profile/<short](http://opcfoundation.org/UA-Profile/%3cshort) name>/Server/<Prf1name> |
| <short name> <Prf2name> Server Facet | [http://opcfoundation.org/UA-Profile/<short](http://opcfoundation.org/UA-Profile/%3cshort) name>/Server/<Prf2name> |
| <short name> <Prf3name> Client Facet | [http://opcfoundation.org/UA-Profile/<short](http://opcfoundation.org/UA-Profile/%3cshort) name>/Client/<Prf3name> |

## Server Facets

### Overview

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

A specification can define multiple *Facets* if not all features are to be implemented by all *Servers* and *Clients*. The name of the *Facet* shall give a hint of the subset. An overall description shall be provided that explains the subset and it potential use.

#### <short name> <Prf1name> Server Profile

Table 100 defines a *Profile* that describes the …….

Table 100 - <short name> <Prf1name> Server Profile

| **Group** | **Conformance Unit / Profile Title** | **M / O** |
| --- | --- | --- |
| Profile | Core 2017 Server Facet http://opcfoundation.org/UA-Profile/Server/Core2017Facet |  |
| Profile | UA-TCP UA-SC UA Binary http://opcfoundation.org/UA-Profile/Transport/uatcp-uasc-uabinary |  |
| Profile | Data Access Server Facet http://opcfoundation.org/UA-Profile/Server/DataAccess |  |
| Profile | <short name> <Prf2name> Server Facet |  |
| Subscription Services | Subscription Durable | M |
| <short name> | <short name> <Function1> | M |

This table lists a *Profile*, in that it includes other base *Profiles* that would be needed to make a working *Server*. It also includes other *Facets* defined in this companion specification and *ConformanceUnits* defined in this companion standard.

The column with title "M / O" defines whether support of included *ConformanceUnits* is optional or mandatory. Optional means that an application has the option to not support the *ConformanceUnit*. However, if supported, the application shall pass all tests associated with the *ConformanceUnit*.

The "Group" for all Conformance Units defined in this document shall be the <short name>. If Conformance Units of OPC 10000-7 are referenced, the corresponding Groups shall be used. See the example with group "Subscription Services".

#### <short name><Prf2name> Server Facet

Table 101 defines a *Facet* that describes the …….

Table 101 - <short name> <Prf2name> Server Facet

| **Group** | **Conformance Unit / Profile Title** | **M / O** |
| --- | --- | --- |
| <short name> | <short name> <Function1> | M |
| <short name> | <short name> <Function3> | O |

This table lists a *Facet*, in that it must be include with other *Facets* to create a running application. It defines the *ConformanceUnits* and other facets that are required

## Client Facets

### Overview

The following tables specify the *Facets* available for *Clients* that implement the <title> companion specification.

A specification can define multiple facets if not all features are to be implemented by all *Servers* and *Clients*. The name of the facet shall give a hint of the subset. An overall description shall be provided that explains the subset and it potential use.

### <short name> < Prf3name> Client Facet

Table 102 defines a *Facet* that describes the base characteristics for all OPC UA *Clients* that make use of this companion specification. Additional *Profiles* will define support for various information models that are part of this document.

Table 102 - <short name> < Prf3name> Client Facet

| **Group** | **Conformance Unit / Profile Title** | **M / O** |
| --- | --- | --- |
| Profile | AddressSpace Lookup Client Facet http://opcfoundation.org/UA-Profile/Client/AddressSpaceLookup |  |
| Profile | DataAccess Client Facet http://opcfoundation.org/UA-Profile/Client/DataAccess |  |
| Profile | DataChange Subscriber Client Facet http://opcfoundation.org/UA-Profile/Client/DataChangeSubscriber |  |
| Session Services | Session Client Detect Shutdown | M |
| <short name> | <short name> Client <Function1> | M |

This table lists a *Facet*, in that it must be include with other *Facets* to create a running application. It defines the *ConformanceUnits* and other facets that are required as an example it include other base Facets and a Base system *ConformanceUnit*

## Namespaces

## Namespace Metadata

Namespace Metadata are required for any companion standard that specifies an information model (e.g. *Objects* and *ObjectTypes*). The metadata provide standardized information about the elements of this namespace. This information is particularly important for aggregating *Servers*.

Table 103 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*. See 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 103 – NamespaceMetadata Object for this Document

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | | **Value** | | |
| BrowseName | | 1:<http://opcfoundation.org/UA/umati> | | |
| **References** | **BrowseName** | | **DataType** | **Value** |
| HasProperty | NamespaceUri | | String | <http://opcfoundation.org/UA/>umati |
| HasProperty | NamespaceVersion | | String | 0.05 |
| HasProperty | NamespacePublicationDate | | DateTime | 2019-10-01 |
| HasProperty | IsNamespaceSubset | | Boolean | False |
| HasProperty | StaticNodeIdTypes | | IdType [] | Null |
| HasProperty | StaticNumericNodeIdRange | | NumericRange [] | {0:65535} |
| HasProperty | StaticStringNodeIdPattern | | String | Null |

## 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 104 provides a list of mandatory and optional namespaces used in a umati OPC UA *Server*.

Table 104 – Namespaces used in a umati 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/umati/ | 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 105 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 105 – Namespaces used in this document

| **NamespaceURI** | **Namespace Index** | **Example** |
| --- | --- | --- |
| http://opcfoundation.org/UA/ | 0 | 0:EngineeringUnits |
| http://opcfoundation.org/UA/umati | 1 | 1:MachineToolType |

1. (normative): <Title> Namespace and mappings
   1. Namespace and identifiers for <Title> Information Model

This appendix defines the numeric identifiers for all of the numeric *NodeIds* defined in this document. 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 *<type>* *ObjectType* *Node* which has the *<property> Property*. The **Name** for the *<property>* *InstanceDeclaration* within the *<type>* declaration is: *AutoIdDeviceType\_DeviceLocation*.

A NamespaceURI follows the convention: [http://opcfoundation.org/UA/<short name>/](http://opcfoundation.org/UA/POWERLINK/).

<short name> is described in 12.

.

Note that NamespaceURIs are NOT live URLs. Text in the specification should not suggest that they are.

The *NamespaceUri* for all *NodeIds* defined here is [http://opcfoundation.org/UA/<short name>/](http://opcfoundation.org/UA/%3cshort%20name%3e/)

**File Locations**

The location of any version dependent files follow this convention:

[http://opcfoundation.org/UA/schemas/<short name>/<version>/<file name>](http://opcfoundation.org/UA/schemas/%3cshort%20name%3e/%3cversion%3e/%3cfile%20name%3e)

The <short name> is the same as specified in the NamespaceURI;

The <version> is a number with the form #.# or #.##;

The location of the version independent files are the same but with the <version> omitted.

e.g. [http://opcfoundation.org/UA/schemas/<short name>/<file name>](http://opcfoundation.org/UA/schemas/%3cshort%20name%3e/%3cfile%20name%3e)

**File Names**

**NodeIds**: Opc.Ua.<short name>.NodeIds.csv or <short name>.NodeIds.csv

**NodeSet**: Opc.Ua.<short name>.NodeSet.xml or <short name>.NodeSet.xml;

Any other files should have a prefix that provides context when the file is downloaded in a browser.

All published files must be added to GitHub <https://github.com/OPCFoundation/UA-Nodeset>

This can be done by creating a mantis issue in the “NodeSets, XSDs and Generated Code” project:

<https://opcfoundation-onlineapplications.org/mantis/main_page.php>

The files should be attached to the mantis issue.

If the NodeSet was generated with the Opc.Ua.ModelCompiler the design file should be attached as well.

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

[http://www.opcfoundation.org/UA/schemas/<short name>/1.0/NodeIds.csv](http://www.opcfoundation.org/UA/schemas/%3cshort%20name%3e/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/<short name>/NodeIds.csv](http://www.opcfoundation.org/UA/schemas/%3cshort%20name%3e/NodeIds.csv)

A NodeIds.csv file is not mandated but recommended.

It contains a flat list of NodeIds with unique names and can be used instead of a full NodeSet if only such NodeId constants for a programming environment are needed.

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

The Information Model Schema released with this version of the document can be found here:

[http://www.opcfoundation.org/UA/schemas/<short name>/1.0/Opc.Ua.<short name>.NodeSet2.xml](http://www.opcfoundation.org/UA/schemas/%3cshort%20name%3e/1.0/Opc.Ua.%3cshort%20name%3e.NodeSet2.xml)

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

[http://www.opcfoundation.org/UA/schemas/short name>/Opc.Ua.<short name>.NodeSet2.xml](http://www.opcfoundation.org/UA/schemas/short%20name%3e/Opc.Ua.%3cshort%20name%3e.NodeSet2.xml)

\_\_\_\_\_\_\_\_\_\_\_