HEIDENHAIN

Information Model

OPC UA NC Server

Core - Version 1.05

English (en)
10/2023

Application-Oriented Monitoring and Control

A variety of industrial applications can be realized with the information, functions, and events provided by the HEIDENHAIN OPC UA NC Server. Below you will find a few typical applications that can be implemented based on the Core Information Model described in this document.

Production Data Acquisition

Software applications for process data acquisition (PDA) provide a real-time view of the production status and productivity of your machines.

Is the current job still running, or was there an interruption? How long did it take to run the job?

For questions like these, your PDA application requires information from your machine's CNC control. The HEIDENHAIN OPC UA NC Server reliably delivers the required information, thus providing a foundation for efficient process data acquisition. The analysis and presentation of your production data are key factors in attaining process transparency and optimization.

Machine Messages

Stay informed by knowing when to change out a tool at the end of its service life or when to refill critical fluid levels to avoid program interruptions.

PDA applications use machine messages to notify you of important events within your manufacturing environment. These messages are recorded by the HEIDENHAIN OPC UA NC Server and forwarded to the OPC UA application, allowing you to respond quickly to machine downtime or avoid it completely.

Automation

Efficiently automated machine tools minimize costs and ensure high availability during production. But constant pricing pressure and growing workpiece variety represent major challenges for automation solutions.

Should the CAM system automatically transfer the program to the machine? Should the tool presetter automatically send the tool geometry to the machine?

The HEIDENHAIN OPC UA NC Server provides useful functions for every application: easy transfer of NC programs, control of the current program, transmission of tool data and automatic synchronization with a database.

Tool Data Management

Machine tools use specially designed tools to remove material from workpieces. For different production technologies different types of tools are used. Depending on the tool type, the machine tool requires a different set of tool data. Since the typical milling machine can nowadays often also turn and grind, the variety of tool data has increased greatly.

To enable you to manage the tool data of your CNC machines remotely, HEIDENHAIN has integrated a digital description of tool data directly in the information model.

Which tool data is relevant for a thread milling cutter? What data describe the wear of a tool? Can the new 3D tool model be used for collision monitoring and material removal simulation?

You can read out answers to these questions directly from the HEIDENHAIN OPC UA NC Server.

Machine-Specific Functionality

Whether you record your production data, provide maintenance personnel with current machine messages, or automate your machine, the HEIDENHAIN OPC UA NC Server gives you proven information models to make your job easier.

The fast and flexible way to more information: The machine manufacturer can also extend the HEIDENHAIN OPC UA NC Server, giving you access to additional sensors, machine subsystems, or values from PLC programs. This allows you to feed your applications with more than raw data, including units of measure, limit values, and other information from your machine through OPC UA.

Content

Terms of Use

Below are the terms of use of this document (information model). The product documentation for the *HEIDENHAIN OPC UA NC Server* is found in the Setup, Testing and Running NC Programs User's Manual or Setup and Program Run User's Manual for the specific control model.

- These terms of use apply solely to the abovementioned information model and do not grant any rights of use for any other documents referenced herein.
- The content of this document has been checked for accuracy with respect to the corresponding software. Nevertheless, deviations cannot be excluded. Therefore, HEIDENHAIN does not guarantee the complete accuracy, correctness, or completeness of this content.
- The user of this document, generally the application engineer, is solely responsible for the correct and proper functioning of any industrial software application created by him or her. Likewise, the application engineer is solely responsible for testing any such industrial software application in conjunction with the corresponding HEIDENHAIN control software and for verifying that any such industrial software application is free of errors.
- The copyright of this document is held solely by DR. JOHANNES HEIDENHAIN GmbH.
- This document may be distributed via computer systems, printed, or copied only if not modified.

WARRANTY AND LIABILITY DISCLAIMERS

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. HEIDENHAIN 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 HEIDENHAIN 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.

Any application examples provided herein are solely intended to provide an overview of typical functionalities. These application examples are neither binding nor complete, nor do they represent specific user solutions or requirements.

The user is solely responsible and liable for the quality and performance of software developed using this document.

About this Document

Where to Start?

Depending on your interest in the HEIDENHAIN OPC UA NC Server and the Core Information Model, the questions and statements listed here may quickly lead you to the relevant parts of the document.

- Which HEIDENHAIN CNC controls have an OPC UA NC Server?
  See 1.3 "Associated HEIDENHAIN CNC Controls", Page 15.
- I want to get an overview about the interface functionality and components of the Core Information Model.
  Start with 3 "Core Information Model Overview", Page 29.
- OPC UA information modeling concepts are new to me. How do I read this specification?
  2.2 "Information Modeling in OPC UA" briefly describes the concepts, 2.4 "Conventions used in this document" explains how to read the following type definitions. Not all details are needed for an initial understanding of the Core Information Model. Use these chapters as a reference when reading the following chapters.
- I have a machine with OPC UA NC Server in front of me. What do I need to do to get the server up and running the first time?
  The control's OPC UA Connection Assistant application supports you in checking the existing conditions and setting up a connection to the server. The application can be found in the HEROS operating system settings menu.
  See also "User’s Manual: Setup, Testing and Running NC Programs, or Setup and Program Run", Page 13, for information about the existing conditions. The respective manual for your specific control model and software version describes it in more detail.

Document History

Table 1: Document Versions

Version	Date	Reason	Comments
1.00	December 11, 2019	Release	First version of the Core Information Model
1.01	November 27, 2020	Release	Core Information Model extension: File System Access
1.02	November 29, 2021	Release	Core Information Model extension: Manufacturer Extensions
1.03	October 27, 2022	Release	Core Information Model extension: Tool Data Management
1.04	March 22, 2023	Release	Minor Core Information Model extension, Annex update
1.05	October 12, 2023	Release	Core Information Model extension: Service files and 3D model data for tools, Annex update

Have you found any errors or would you like to suggest a new feature?

We are continuously striving to improve our documentation and the server for you. Please help us by sending your suggestions to the following e-mail address:

OPCUA-NC-docu@heidenhain.de

What's New?

Extensions of the Core Information Model from Version 1.04 to Version 1.05

- 3D Model Data for Tools and Tool Carrier Kinematics

  Information Model Extensions

  - 4.16 "ToolDataManagementType" has a new optional component *Validation*, which has a component *3DModels*. This component provides information and functionality for tool data items which refer to 3D model files.
  - 4.19 "ToolDataItemType" has two new optional variables *FileLocation* and *FileLocationManufacturer*. They specify storage locations of associated files.

  More Information

  - 3D Model Files for Tools in 3.10 "Tool Data Management" provides an overview of the new elements and functionality for 3D model data for tools. For example, how to check whether 3D models of the tool or tool carrier can be used for collision monitoring and material removal simulation on the control.
  - "3D Model Files for Tools: Validation Issues List", Page 190 contains a list of possible issues found during validation of 3D model files.
- Machine Diagnostics with ServiceFiles

  Information Model Extensions

  - The new 4.23 "ServiceFileInterfaceType" provides a method to create service files of the machine.
  - 4.1 "MachineType" has a new optional component *Diagnostics* with a component *ServiceFiles* of 4.23 "ServiceFileInterfaceType".
- "Annex A: Tool Data Reference" has been updated according to the corresponding NC software version.

The new elements are supported by all products starting with the corresponding NC software versions listed under Overview of Core Information Model Versions in 1.3 "Associated HEIDENHAIN CNC Controls". The availability and use of 3D model files of tools and tool carriers for collision monitoring during program run and material removal simulation depend on the software option Dynamic Collision Monitoring (DCM) or DCM version 2.

Extensions of the Core Information Model from Version 1.03 to Version 1.04

- 4.16 "ToolDataManagementType" has a new optional method *GetAllToolNumbers*. It provides a list of all tool numbers that are present in the machine's tool memory.
- "Annex A: Tool Data Reference" has been updated according to the corresponding NC software version.

The new elements are supported by all products starting with the corresponding NC software versions listed under Overview of Core Information Model Versions in 1.3 "Associated HEIDENHAIN CNC Controls".

Extensions of the Core Information Model from Version 1.02 to Version 1.03

- Tool Data Management

  Information Model Extensions

  - 4.1 "MachineType" has a new optional component *ToolDataManagement* of type 4.16 "ToolDataManagement-Type". It describes the machine's available tool types, including their data items, and provides methods to access the tool data.
  - The new 4.16 "ToolDataManagementType" includes instances of the newly added 4.17 "ToolDataRepresentationType" and 4.22 "LocalToolDataSetType".
  - To describe the semantics of the machine's tool data with the 4.17 "ToolDataRepresentationType", the *ObjectTypes* 4.18 "ToolDataCategoryType" , 4.19 "ToolDataItemType", 4.20 "ToolTypeCategoryType" and 4.21 "ToolTypeDescriptionType" have been added.

  - The *Enumerations* 7.6 "ToolDataSynchronizationStatusType" and 7.5 "ToolRecordModificationType" as well as the *Structures* 7.10 "ToolRecordIdentifierDataType", 7.11 "MagazinePocketIdentifierDataType" and 7.9 "ToolRecordModificationDataType" have been added.
  - The *EventTypes* 5.7 "ToolDataSetModificationEventType", 5.8 "BaseToolEventType" and 5.9 "ToolLockedEventType" have been added.
  - The 4.12 "ChannelType" has been extended by the optional component *CurrentTool*.

  More Information

  - 3.10 "Tool Data Management" provides an overview of the new tool data management functionality and related types.
  - "Annex A: Tool Data Reference" contains a list of tool data items and tool types.
  - Terms in 2.3 "Abbreviations and Terms" defines tool and tool data related terms used in this document.

  The new Tool Data Management is supported by all products starting with the corresponding NC software versions listed under Overview of Core Information Model Versions in 1.3 "Associated HEIDENHAIN CNC Controls".

Extensions and Changes of the Core Information Model from Version 1.01 to Version 1.02

- Extensions of the Machine Manufacturer
  - 4.1 "MachineType" has a new optional component: *ManufacturerExtensions*. Machine-specific objects with variables are *Organized* by this object. *OptionalPlaceholder* elements represent the *ObjectType* and *VariableTypes* that can occur. Concrete machine-specific objects and variables are defined by the machine manufacturer.
  - Along with the 3.2 "Machine Instance" overview and the 4.1 "MachineType" type, chapter 10 "Extensions of the Machine Manufacturer" provides information about this functionality.

  The new Extensions of the Machine Manufacturer are supported by all products starting with the corresponding NC software versions listed under Overview of Core Information Model Versions in 1.3 "Associated HEIDENHAIN CNC Controls".
- Description and behavior hints of 4.12 "ChannelType" override variables, 4.15 "NCProgramStateMachineType" *Start* method, 4.9 "OperatorMachineInfoType" and 4.10 "ManufacturerInfoType" *Image* properties have been improved.
- Unused *VariableTypes CutterLocationType* and *ProgramPositionType* have been removed.

Extensions of the Core Information Model from Version 1.00 to Version 1.01

- File System Access

  Information Model Extensions
  - 4.1 "MachineType" now has an optional component *FileSystem* providing access to the *TNC* and *PLC* partitions of the machine's file system to transfer files to and from the machine following OPC UA Part 5 Annex C (using *FileType* and *FileDirectoryType*).
  - The machining-channel-related types 4.12 "ChannelType", 4.14 "ProgramType" and 4.15 "NCProgramStateMachineType" now have additional optional methods like *SelectProgramByNodeId* or a property like *FileNodeId* to simplify the usage of their functionality in relation to the new *FileSystem*.

  More Information
  - 3.9 "File System Access" and 9.1 "Introduction": Machine's file system model overview and access-right handling
  - 9.3 "FileType" and 9.2 "FileDirectoryType": Overview, hints and annotations regarding the OPC UA standard types *FileType* and *FileDirectoryType* to be used in a *FileSystem*.
  - 9.4 "Warnings and Important Hints": Important and useful information regarding file system functionality usage for OPC UA client developers

  The new File System Access is supported by all products starting with the corresponding NC software versions listed under Overview of Core Information Model Versions in 1.3 "Associated HEIDENHAIN CNC Controls".

1

Introduction

1.1 Scope

This documentation describes the information models made available by the HEIDENHAIN OPC UA NC Server and is intended for software developers for the development of modern industrial software.

The structure and concepts of this OPC UA information model specification are primarily derived from the public domain document OPC UA Companion Specification Template by the OPC Foundation. (Client application developers should have a basic understanding of OPC UA in order to understand and use this specification.)

Due to, for example, the platform independency of the open OPC UA standard, HEIDENHAIN does not provide an OPC UA Client SDK. For general questions on how to implement a client application with a specific OPC UA Client SDK of your choice, please refer to the manufacturer of the SDK.

This documentation describes the contents of the standard scope. Modification by the machine manufacturer may influence the behavior of the control and the OPC UA NC Server. Any additions or modifications must be documented by the machine manufacturer.

Core Information Model Specification

The HEIDENHAIN OPC UA NC Server Core Information Model is described in this document. The Core Information Model defines an interface to HEIDENHAIN numeric control systems provided by the HEIDENHAIN OPC UA NC Server.

The Core Information Model contains the basic contents and functions, such as the identification of the control, the machine, their characteristics, and their specific version information.

This specification is intended to be extended in the future, so please ensure that you use the most recent version of this document.

For technical questions regarding the Core Information Model, feel free to contact the HEIDENHAIN App-Programming Helpline.

HEIDENHAIN OPC UA NC Server

The HEIDENHAIN OPC UA NC Server provides an interface between HEIDENHAIN NC systems and external applications for machine-related information, monitoring, and control functions. The server is not a stand-alone product or application: it is always part of numeric control software products from HEIDENHAIN.

References to other information about the software option HEIDENHAIN OPC UA NC Server or the HEIDENHAIN controls can be found in 1.2.2 "HEIDENHAIN".

The HEIDENHAIN CNC control products for which the HEIDENHAIN OPC UA NC Server options are available, can be found, for example, on the HEIDENHAIN website and in 1.3 "Associated HEIDENHAIN CNC Controls". This chapter also contains information about the version of the Core Information Model supported by an OPC UA NC Server as part of the related NC Software version.

1.2 Reference Documents

1.2.1 OPC Foundation

All referred documents are available for registered users on the website of the OPC Foundation as part of the specification of OPC Unified Architecture.

The OPC Foundation additionally provides an OPC UA Online Reference of OPC UA specifications and information models.

OPC UA Specification (starting with version 1.04)

OPC UA is a secure, reliable and platform-independent communication standard that is used for information exchange in industrial applications. It is maintained by the OPC Foundation and published as the international standard IEC 62541. More than a data transport layer, OPC UA aims to enable interoperability through the use of information modeling and web services.

After version 1.04 each part of the specification is published individually. For updated and new parts published until March 2022, at least version 1.05 applies.

Referenced OPC UA Specification Parts

- OPC UA Part 1: OPC Unified Architecture - Part 1: Overview and Concepts
- OPC UA Part 2: OPC Unified Architecture - Part 2: Security Model
- OPC UA Part 3: OPC Unified Architecture - Part 3: Address Space Model
- OPC UA Part 4: OPC Unified Architecture - Part 4: Services
- OPC UA Part 5: OPC Unified Architecture - Part 5: Information Model
- OPC UA Part 6: OPC Unified Architecture - Part 6: Mappings
- OPC UA Part 7: OPC Unified Architecture - Part 7: Profiles
- OPC UA Part 8: OPC Unified Architecture - Part 8: Data Access
- OPC UA Part 16: OPC Unified Architecture - Part 16: State Machines
- OPC UA Part 18: OPC Unified Architecture - Part 18: Role-Based Security
- OPC UA Part 20: OPC Unified Architecture - Part 20: File Transfer

OPC UA Companion Specification Template

The OPC UA Companion Specification Template is provided by the OPC Foundation and can be used to define and publish OPC UA companion specifications or OPC UA information model specifications developed by joint working groups or other organizations.

The structure and concepts of this OPC UA information model documentation are mostly derived from this template.

1.2.2 HEIDENHAIN

Note that these documents always apply to a specific software version of the various numeric control models; see 1.3 "Associated HEIDENHAIN CNC Controls" for details.

Brochure: HEIDENHAIN OPC UA NC Server

The brochure provides an overview of the range of functions offered by the HEIDENHAIN OPC UA NC Server.

HEIDENHAIN OPC UA NC Server: The industry standard for machine tools

User’s Manual: Setup, Testing and Running NC Programs, or Setup and Program Run

The User’s Manual contains information relevant to setting up an OPC UA connection to the controller. This includes, for example, enabling the software option for the OPC UA NC Server, the network and firewall settings, or the management of digital certificates and the license settings for the application.

In addition to setting up the connection, the User’s Manual describes how to configure the contents of the *Machine's OperatorInfo* (of 4.9 "OperatorMachineInfoType") via the machine configuration.

TNCguide

Technical Manual (for machine manufacturers)

The Technical Manual for machine manufacturers describes how to set the contents of the *Machine's ManufacturerInfo* (of 4.10 "ManufacturerInfoType") via the machine configuration.

It also describes how the machine manufacturer can configure the *Machine*'s *ManufacturerExtensions* (see 10 "Extensions of the Machine Manufacturer") and extend the provided information with his own, for example from his PLC programm.

Machine manufacturers can find the Technical Manual via their Filebase access.

Klartext Programming

This manual is intended to help you quickly learn to handle the most important procedures on the HEIDENHAIN control. For an OPC UA Client developer this manual can be helpful in case he uses the HEIDENHAIN programming station (VirtualBox image) for testing purposes.

TNCguide

Information Model OPC UA NC Server

New versions of this document can be found here:

www.heidenhain.de/opcua-nc-server

1.3 Associated HEIDENHAIN CNC Controls

Current Core Information Model Version 1.05

The Core Information Model version 1.05 described within this document is supported by HEIDENHAIN OPC UA NC Server starting with the following products and versions:

- TNC7: NC software 817620-18, 817621-18, 817625-18
- TNC7 basic: NC software 817621-18, 817625-18
- TNC 640: NC software 340590-18, 340591-18, 340595-18
- TNC 620: NC software 817600-18, 817601-18, 817605-18

In general, all products listed here support the newly added elements (see "What's New?", Page 9); only the scope of application differs depending on the available software options.

Overview of Core Information Model Versions

An overview of the different Core Information Model versions supported by the OPC UA NC Server included within the specific products and versions is given in Table 2.

For the changes and extensions of the different versions, see "What's New?", Page 9.

Table 2: Supported Core Information Model Versions

Core Information Model Version	Product	As of NC software version
1.00	TNC 640	34059x-10
1.01	TNC 640	34059x-11
	TNC 620	81760x-08
1.02	TNC7	81762x-16
	TNC 640	34059x-16
	TNC 620	81760x-16
1.03	TNC7	81762x-17
	TNC 640	34059x-17
	TNC 620	81760x-17
1.04	TNC7	81762x-17 SP01
	TNC 640	34059x-17 SP01
	TNC 620	81760x-17 SP01
1.05	TNC7	81762x-18
	TNC7 basic	81762x-18
	TNC 640	34059x-18
	TNC 620	81760x-18

HEIDENHAIN has simplified the version schema, starting with NC software version 16:

- The publication period determines the version number.
- All control models of a publication period have the same version number.
- The version number of the programming stations corresponds to the version number of the NC software.

2

Fundamentals

2.1 Introduction to OPC Unified Architecture

2.1.1 What is OPC UA?

OPC UA is an open and royalty-free set of standards designed as a universal communication protocol.

While there are numerous communication solutions available, OPC UA has some key advantages:

- A state-of-the-art security model (see OPC UA Part 2)
- A fault-tolerant communication protocol
- An information-modeling 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 the OPC UA NC Server Core Information Model, 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 example. For a more complete overview, see OPC UA Part 1.

2.1.2 Basics of OPC UA

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

As an extensible standard, OPC UA provides a set of Services (see OPC UA Part 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 Image 1.

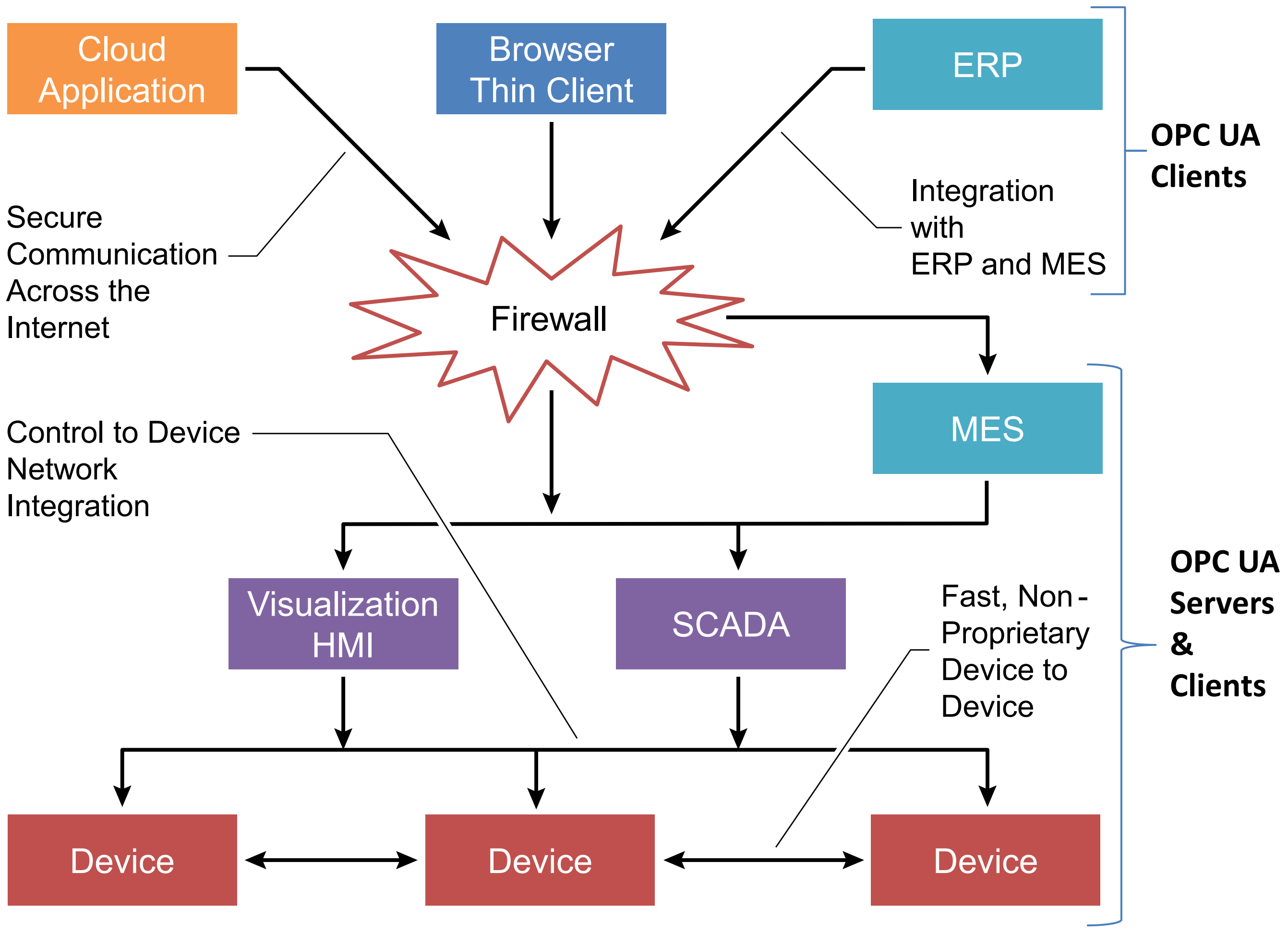
Cloud Application
Browser Thin Client
ERP
OPC UA Clients
Secure Communication Across the Internet
Integration with ERP and MES
Firewall
MES
Control to Device Network Integration
Visualization HMI
SCADA
Fast, Non-Proprietary Device to Device
OPC UA Servers & Clients
Device
Device
Device

Figure 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-performance data exchange solution. Security is built into OPC UA as security requirements become increasingly important, especially since production environments are connected to the office network or the internet and attackers are starting to focus on automation systems.

2.2 Information Modeling in OPC UA

This chapter provides an overview about the concepts of information modeling in OPC UA, their representation in this document, and introduces the related terms. These and the conventions as defined in 2.4 "Conventions used in this document" are used in this document to describe the Core Information Model.

Defined terms of the OPC UA specification, types, and their components in the OPC UA specification and in this specification are *italicized* in this document.

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 a *BrowseName*. An *Object* representing a 'Reservation' is shown in Image 2.

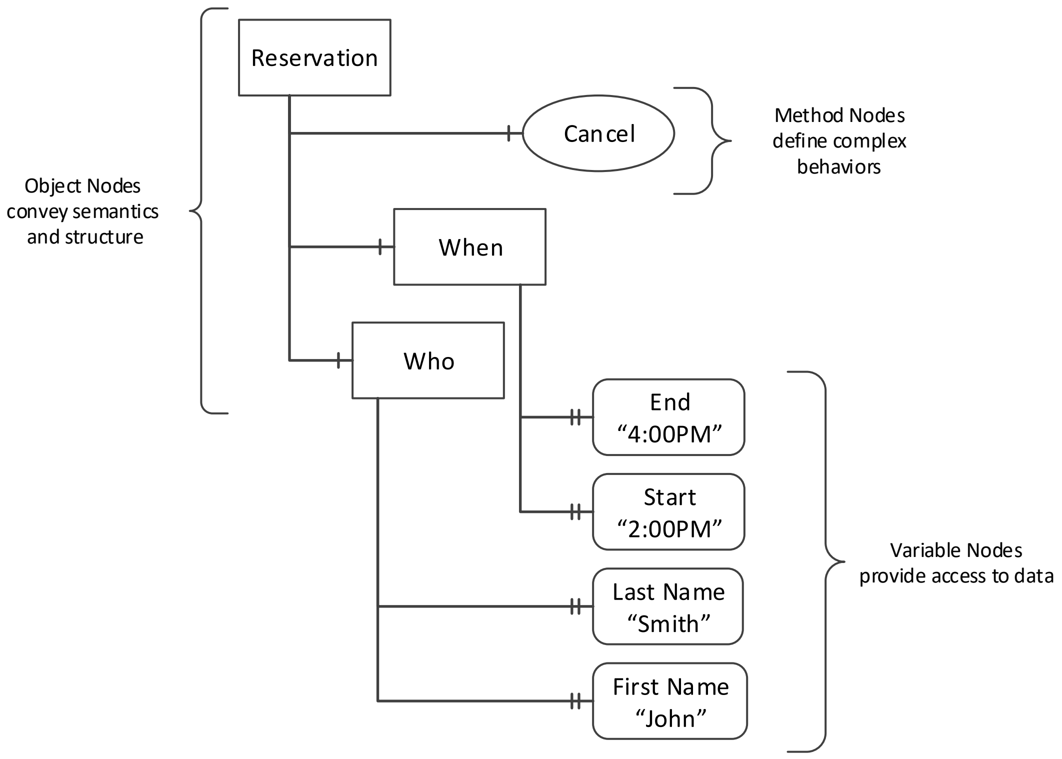
Reservation
Cancel
Method Nodes define complex behaviors
Object Nodes convey semantics and structure
When
Who
End "4:00PM"
Start "2:00PM"
Last Name "Smith"
First Name "John"
Variable Nodes provide access to data

Figure 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. Image 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 Image 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.

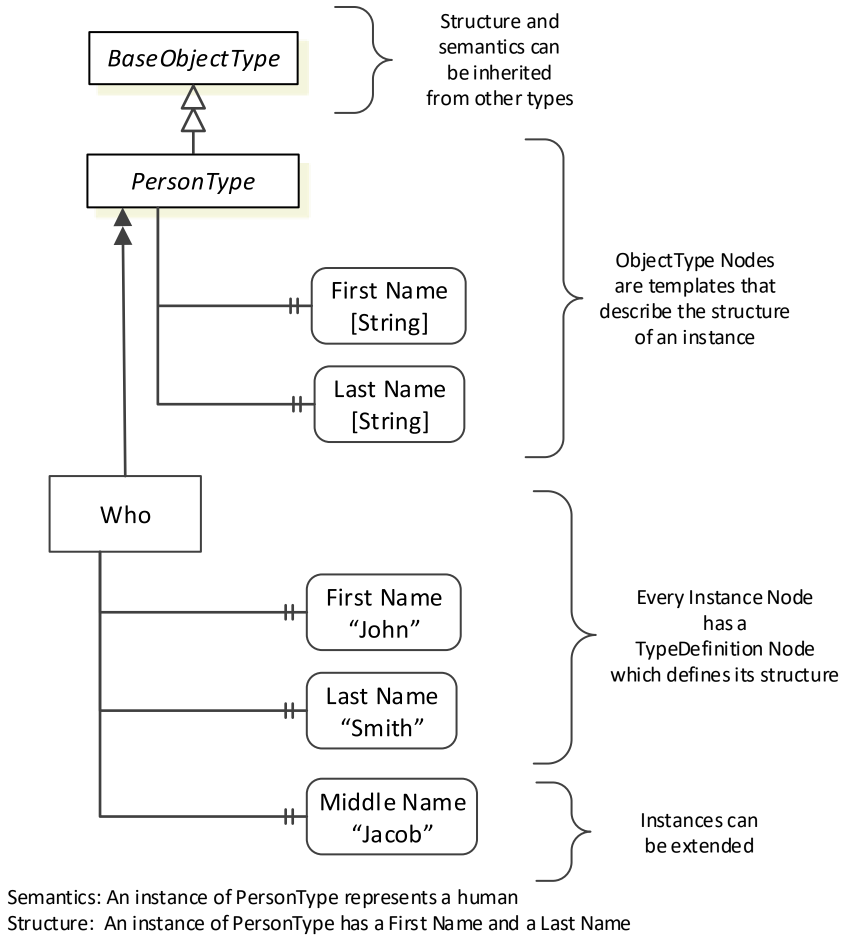
BaseObjectType
PersonType
First Name [String]
Last Name [String]
Who
First Name "John"
Last Name "Smith"
Middle Name "Jacob"
Structure and semantics can be inherited from other types
ObjectType Nodes are templates that describe the structure of an instance
Every Instance Node has a TypeDefinition Node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

Figure 3: The Relationship between Type Definitions and Instances

OPC UA also supports the concept of sub-typing. This allows an information model developer to take an existing type and extend it. Rules regarding sub-typing are defined in OPC UA Part 3, but in general they allow the extension of a given type or the restriction of a *DataType*. For example, the modeler may decide that the existing *ObjectType* in some cases needs an additional *Variable*. The modeler 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 subtype could restrict it to a float.

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. Image 4 depicts several References, connecting different *Objects*.

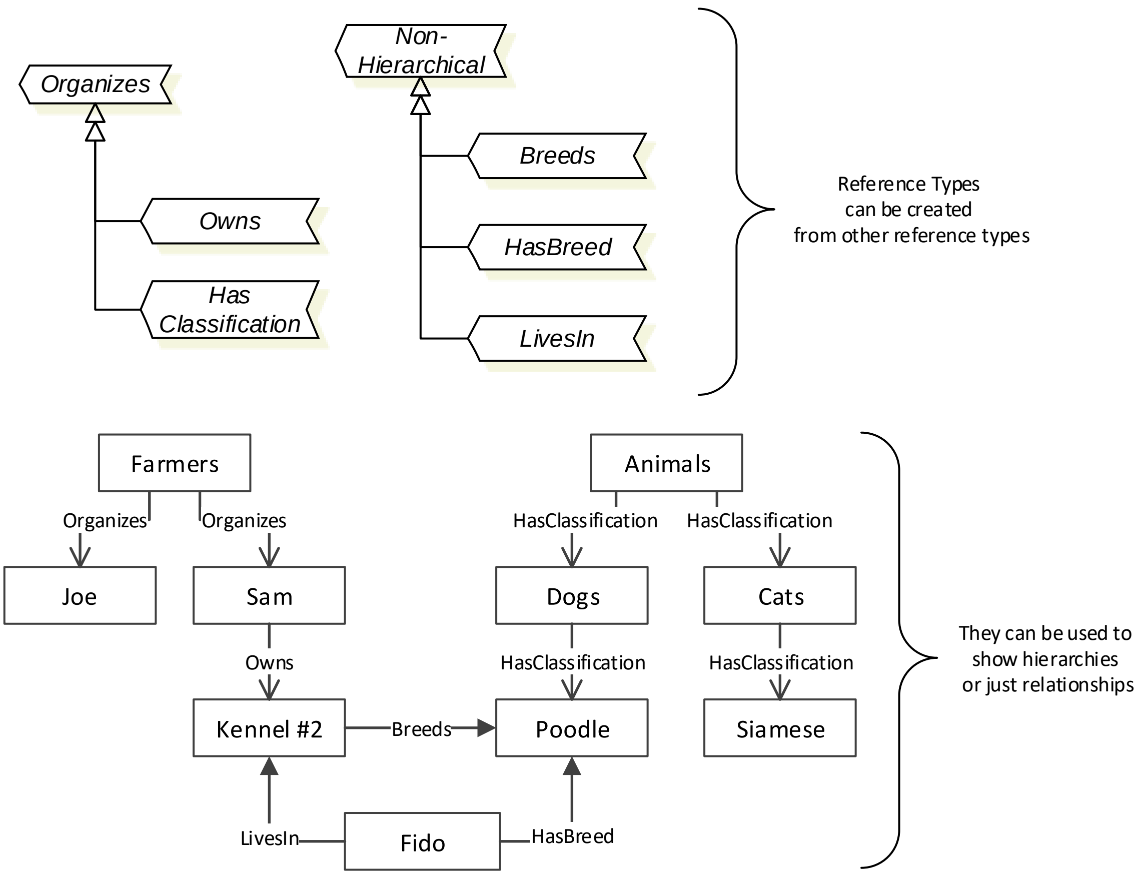

Figure 4: Examples of References between Objects

The figures above use a notation that was developed for the OPC UA specification. The notation is summarized in Image 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*.

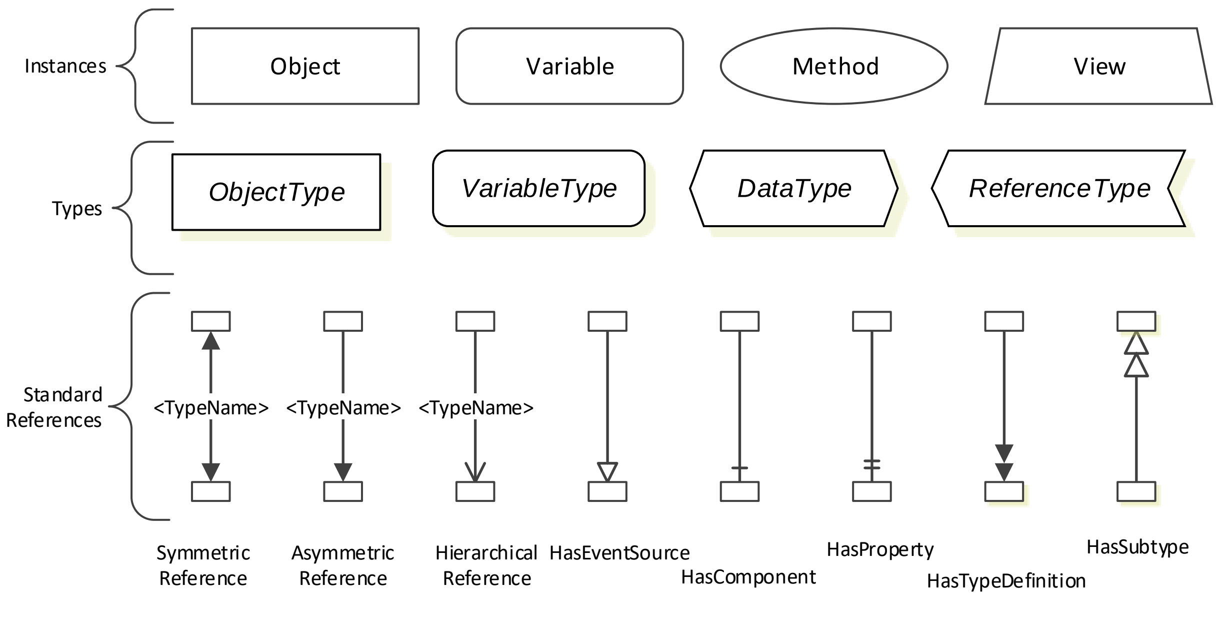

Figure 5: The OPC UA Information Model Notation

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

The 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 UA Part 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.

2.3 Abbreviations and Terms

Abbreviations

DCM Dynamic Collision Monitoring

CC Controller Computer

FS Functional Safety

JSON JavaScript Object Notation

M3D Mesh3D

MC Main Computer

NC Numeric Control

PLC Programmable Logic Control

SPLC Safety Programmable Logic Control

STL Standard Tessellation Language

URI Uniform Resource Identifier

XML Extensible Markup Language

Terms

Tools and **tool data** at associated HEIDENHAIN CNC controls:

To one **tool** belongs a set of data to identify the tool and describe the tool's characteristics. Next to information about the tool life and wear there is also data regarding the production technology and usage of the tool. A HEIDENHAIN control stores the data of one physical tool in at least one **tool record**.

A tool record is addressed by a tool number and a tool index. The **tool number** identifies a tool within the scope of one machine. If more than one record is needed to describe a tool (e.g., in case of a stepped tool), the additional **tool index** refers to the separate records.

If a tool is defined at the control using more than one tool record, it is called **indexed tool**. HEIDENHAIN recommends having always a tool record with index 0, the so called **main tool**. Additional indices take the value range from 1 to 9.

The **name** of a tool does not uniquely identify a tool in a machine. It is often used to identify tools of the same kind with the similar (geometrical) characteristics (e.g., "MILL_D6_ROUGH"). Each index of a tool can have a specific name.

Often a tool management system is used to manage the tools and their data on a shop floor. To identify a tool within the entire company, they assign a unique identifier to each physical tool. The identifier is often used as a (tool) **database ID** to link the physical tool to its data in the central tool data management system.

If a tool is moved from one machine to the other the tool number may change, but its database ID remains the same.

2.4 Conventions used in this document

The following sections contain a short overview about how the Core Information Model is described within this document.

It is assumed that basic concepts of OPC UA information modeling are understood in this specification. This document will use these concepts to describe the Core Information Model. For the purposes of this document, the terms and definitions given in OPC UA Part 3, OPC UA Part 4, OPC UA Part 5 and OPC UA Part 8 apply.

As already mentioned, defined terms of the OPC UA specification, types, and their components in the OPC UA specification and in this specification are *italicized* in this document.

For readability and less complex descriptions, an instance of a specific type can be denoted similar to the type without the -*Type* suffix (e.g., an *Event* of type *ErrorEventType* is called *ErrorEvent*).

2.4.1 Conventions for Node descriptions

Node definitions are specified using tables for *Attributes* (see 2.4.3 "Common Attributes") and *References* (see Table 4).

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 target *Node,* and its *NodeClass*.

- If the target *Node* 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, *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, *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 UA Part 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 UA Part 3) and *ArrayDimensions* is set to null or is omitted. Examples are given in Table 3.

Table 3: Examples of DataTypes

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

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

Table 4: Type Definition Table

References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
ReferenceType name	*NodeClass* of the target *Node*	*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*.	Referenced *ModellingRule* of the referenced *Object*.

NOTE Notes referencing footnotes of the table content.

Components of *Nodes* can be complex, that is they themself contain components. The *TypeDefinition*, *NodeClass*, *DataType* and *ModellingRule* can be derived from the type definitions, and the symbolic name can be created as defined in 2.4.3 "Common Attributes". Therefore, the contained components are not explicitly specified; they are implicitly specified by the type definitions.

2.4.2 NodeIds and BrowseNames

NodeIds

The *NodeIds* of all *Nodes* described in this standard are only symbolic names.

The symbolic name of each *Node* defined in this specification is its *BrowseName*, or, when it is part of another *Node*, the symbolic name is the *BrowseName* of the other *Node*, a period (.), 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* that are not part of another *Node* have a unique name in this specification, the symbolic name is unique.

The NamespaceUri for all *NodeIds* defined in this specification is defined in 8.1 "Namespace Metadata". The NamespaceIndex for this NamespaceUri is vendor-specific and depends on the position of the NamespaceUri in the server namespace table.

Note that this specification 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 specification, because they are not defined by this specification but generated by the *Server*.

BrowseNames

The text part of the *BrowseNames* for all *Nodes* defined in this specification is specified in the tables defining the *Nodes*. The NamespaceUri for all *BrowseNames* defined in this specification is defined in 8.1 "Namespace Metadata".

If the *BrowseName* is not defined by this specification, 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 specification. Table 202 provides a list of namespaces and their indexes as used in this specification.

2.4.3 Common Attributes

General

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

For all *Nodes* specified in this specification, the *Attributes* named in Table 5 shall be set as specified in the table.

Table 5: Common Node Attributes

Attribute	Value
DisplayName	The *DisplayName* is a *LocalizedText*. Each server shall provide the *DisplayName* identical to the *BrowseName* of the *Node* for the LocaleId “en”. Whether the server provides translated names for other LocaleIds is server-specific.
Description	Optionally a server-specific description is provided.
NodeClass	Shall reflect the *NodeClass* of the *Node*.
NodeId	The *NodeId* is described by *BrowseNames* as defined in "BrowseNames" .
WriteMask	Optionally the *WriteMask Attribute* can be provided. If the *WriteMask Attribute* is provided, it shall set all non-server-specific *Attributes* to "not writable". For example, the *Description Attribute* may be set to "writable" since a *Server* may provide a server-specific description for the *Node*. The *NodeId* shall not be writable, because it is defined for each *Node* in this specification.
UserWriteMask	Optionally the *UserWriteMask Attribute* can be provided. The same rules as for the *WriteMask Attribute* apply.
RolePermissions	Optionally server-specific role permissions can be provided.
UserRolePermissions	Optionally the role permissions of the current *Session* can be provided. The value is server-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 specification, the *Attributes* named in Table 6 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 6: 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 specification, the *Attributes* named in Table 7 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 7: Common Variable Attributes

Attribute	Value
MinimumSamplingInterval	Optionally, a server-specific minimum sampling interval is provided.
AccessLevel	The access level for *Variables* used for type definitions is server-specific; for all other *Variables* defined in this specification, the access level shall allow reading; other settings are server-specific.
UserAccessLevel	The value for the *UserAccessLevel Attribute* is server-specific. It is assumed that all *Variables* can be accessed by at least one user.
Value	For *Variables* used as *InstanceDeclarations*, the value is server-specific; otherwise it shall represent the value described in the text.
ArrayDimensions	If the *ValueRank* does not identify an array of a specific dimension (i.e. *ValueRank* <= 0) the *ArrayDimensions* can either be set to null or the *Attribute* is missing. This behavior 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 specification, the *Attributes* named in Table 8 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 8: Common VariableType Attributes

Attribute	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 behavior 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 specification, the *Attributes* named in Table 9 shall be set as specified in the table. The definitions for the *Attributes* can be found in OPC UA Part 3.

Table 9: Common Method Attributes

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

3

Core Information Model Overview

3.1 Core Types Overview

Model Overview

In this overview the modeling concepts of the OPC UA NC Server Core Information Model are explained and several examples are presented. The aim of this outline is to give an overview of the different components and concepts of the model, so that the detailed descriptions of the types in the following chapters are easier to understand.

The object- and event-type hierarchy of the OPC UA Information Model is displayed in Image 6.

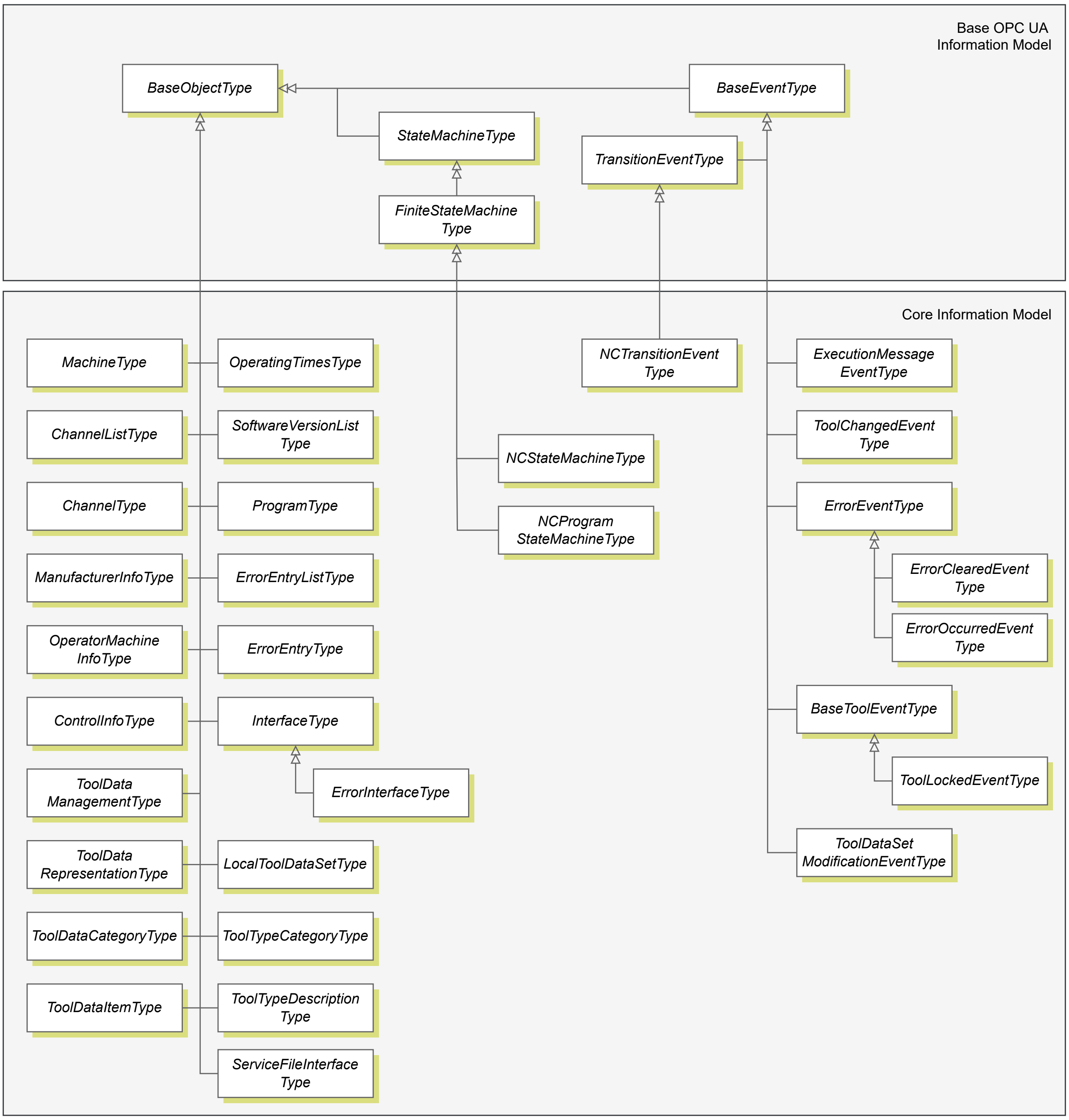

Figure 6: Types of the Core Information Model

Each manufacturing system with its control is represented by an instance of the *MachineType*. The object instance *Machine* is located in a *HEIDENHAIN NC* folder inside the OPC UA *Server's* AddressSpace *Objects* folder.

3.2 Machine Instance

Each machine tool is represented by a *Machine* object of type *MachineType* as shown in Image 7.

The *Machine* object has several components that contain all information and functions:

- The *Machine* has one or more machining *Channels* (NC channels) (see 3.5 "Machining Channel")
- The *State* of the NC can be retrieved and monitored via the *Machine*'s *State* object (see 3.4 "NC State Machine")
- Error information can be retrieved and monitored via the *Machine's Errors* object (see 3.7 "Errors, Warnings and Notifications")
- The *Machine's OperatingTimes* contains several operating times, such as the machine's up-time (see 3.8 "Operating Times of the Machine")
- *ControlInfo*, *ManufacturerInfo* and *OperatorInfo* contain all kinds of additional information related to the machine, like the model number, software versions, etc., with an overview described in 3.3 "General Information about the Machine" and details in 4.10 "ManufacturerInfoType" and 4.9 "OperatorMachineInfoType".
- *FileSystem* provides access to the TNC and PLC partitions of the machine's file system (see 3.9 "File System Access" and 9 "Machine File System Access").
- *ManufacturerExtensions* provides access to machine-specific information and is configured by the machine manufacturer (see 10 "Extensions of the Machine Manufacturer"). Objects and variables referenced by *ManufacturerExtensions* are not specified by the Core Information Model and can differ on various machines.
- *ToolDataManagement* provides access to the tool data of the *Machine* (see 3.10 "Tool Data Management" and 4.16 "ToolDataManagementType").
- *Diagnostics* provides information and functions for machine diagnostics like creation of service files (see 4.23 "ServiceFileInterfaceType").

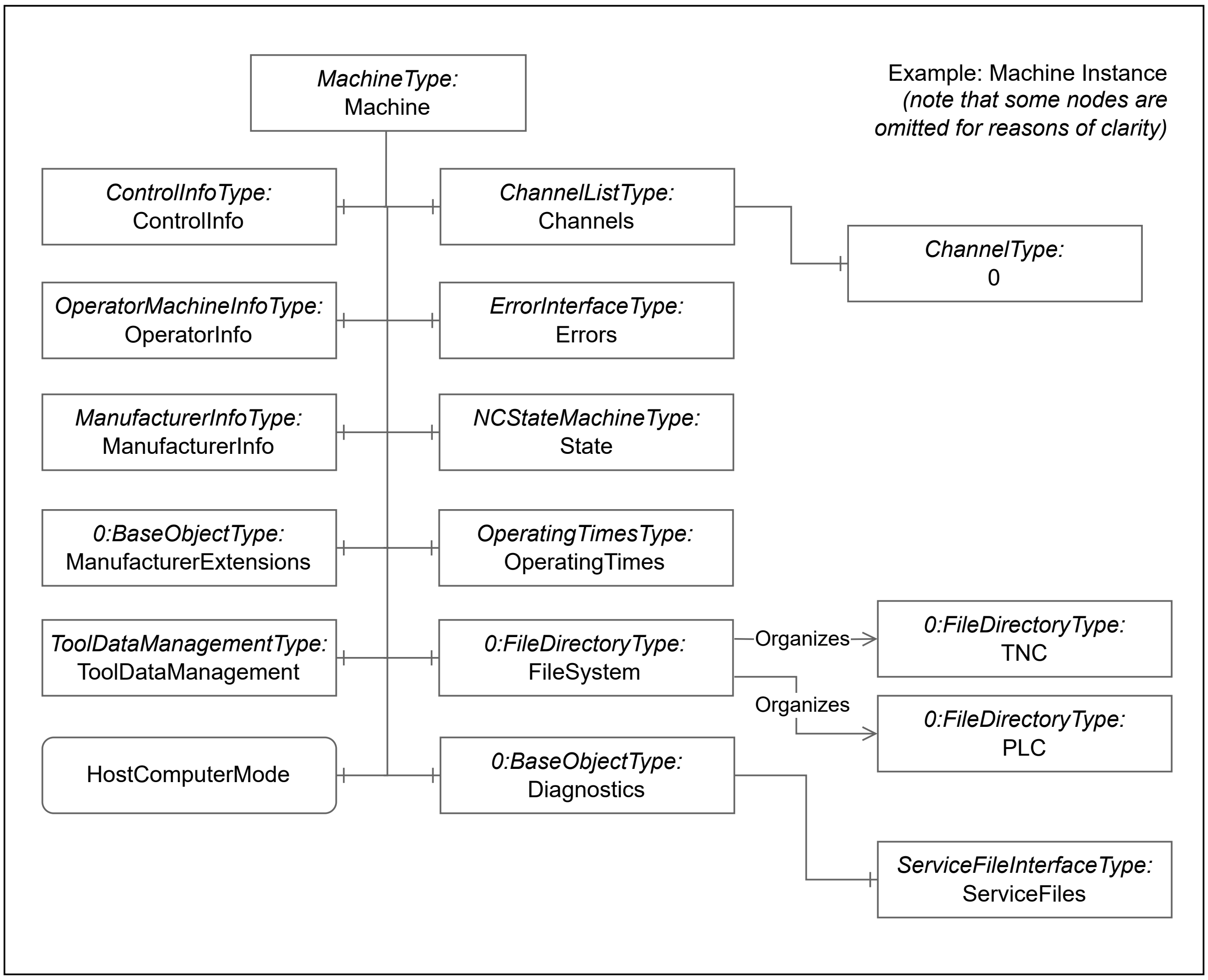

Figure 7: Example *Machine* instance of *MachineType*

3.3 General Information about the Machine

Aside from information directly related to manufacturing and functions to control the operation of the machine, several info objects of *Machine* provide additional information. This additional information identifies the machine and its software components or can provide information like the location of the machine or the manufacturer service contact.

This additional machine information is grouped into three categories:

- *ControlInfo* is the category where HEIDENHAIN provides information like the control model and software version (see 4.8 "ControlInfoType").
- *ManufacturerInfo* is the category where the machine manufacturer can provide information like product type, serial number, a (local) service contact, etc. (see 4.10 "ManufacturerInfoType").
- *OperatorInfo* is the category where the owner of the machine can provide information like the location of the machine or an inventory number (see 4.9 "OperatorMachineInfoType").

(For the complete list of all components of the three machine information object types, refer to the chapters describing the individual types.)

The values for the components of *ManufacturerInfo* and *OperatorInfo* at the *Machine* object can be set in the machine's configuration. The configuration parameters are described in the technical documentation of the specific control type or their user manuals.

3.4 NC State Machine

OPC UA Server and the Control

The *Server* on a machine is able to run independently from the control software. If, for example, only the control software is restarted and the operating system remains active, the *Server* is also not restarted. But during this time the *Server* cannot provide all information about the control.

Whether the *Server* has a connection to the control and in which state the control is at the moment is represented by the *State* component of a *Machine*.

Control State Machine

The *Machine's State* object is of *NCStateMachineType*, which is a subtype of OPC UA *FiniteStateMachineType*, as shown in Image 8. For more information on OPC UA StateMachines, refer to OPC UA Part 16.

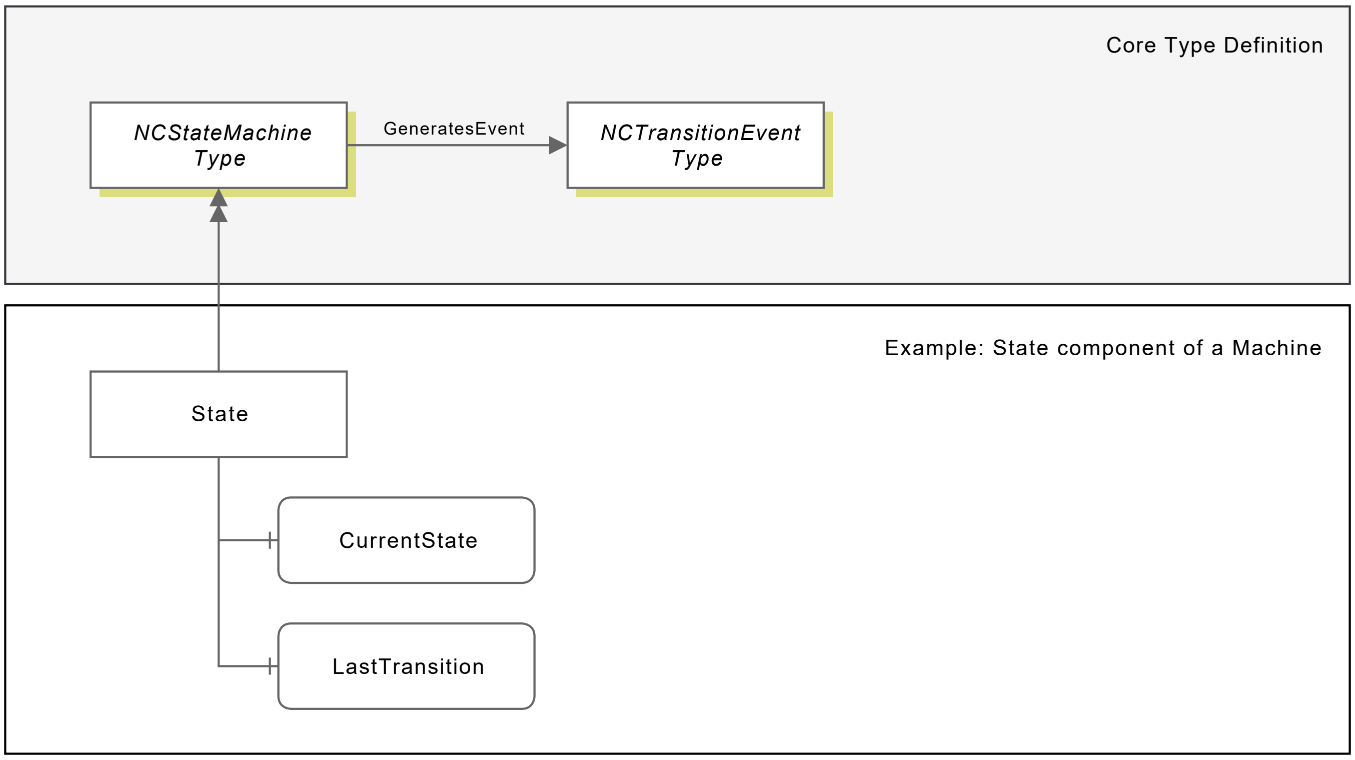

Figure 8: Example *StateMachine* instance *State* of type *NCStateMachineType*

With its states and transitions as displayed in Image 9 the *NCStateMachineType* covers the connection status of the *Server* to the control and different states of the control during startup and initialization.

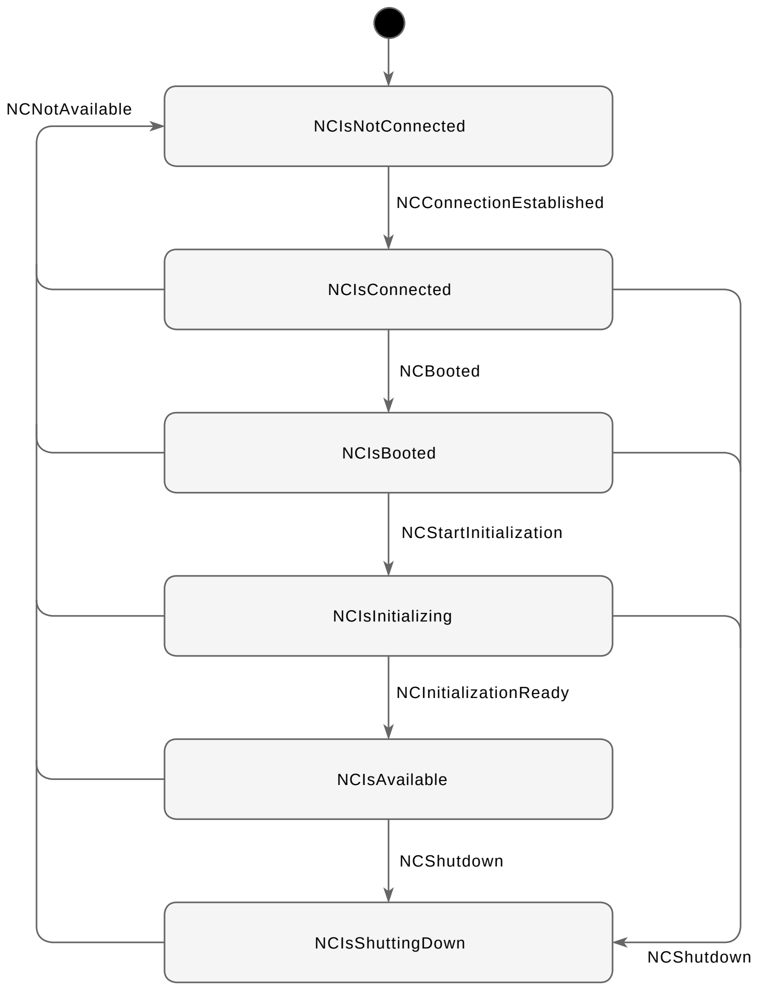

Figure 9: State machine diagram of *NCStateMachineType*

When the state of an *NCStateMachine* changes, events of type *NCTransitionEventType* are emitted. In addition to the *TransitionEventType* components, the *NCTransitionEventType* has a *TransitionReason*.

3.5 Machining Channel

Channels of a Machine

Machine axes can be grouped into several machining channels. Each machining channel executes its program as a separate control. A multi-channel capable control can execute several programs simultaneously and, for example, machine the front side in one channel, then transfer the workpiece to the second channel and machine the rear side there while the first channel machines the front side of the next workpiece.

As shown in Image 7, a *Machine* has a *Channels* list containing one *ChannelType* instance for every machining channel. The *BrowseName* and *DisplayName* of a *Channel* object are derived from the *Id* of the machining channel.

The different components of a *Channel*, illustrated in Image 10, provide, for example, the possibility to monitor and control the program execution (see also 3.6 "NC Program Execution Monitoring and Control" and further methods and information described within the following sections).

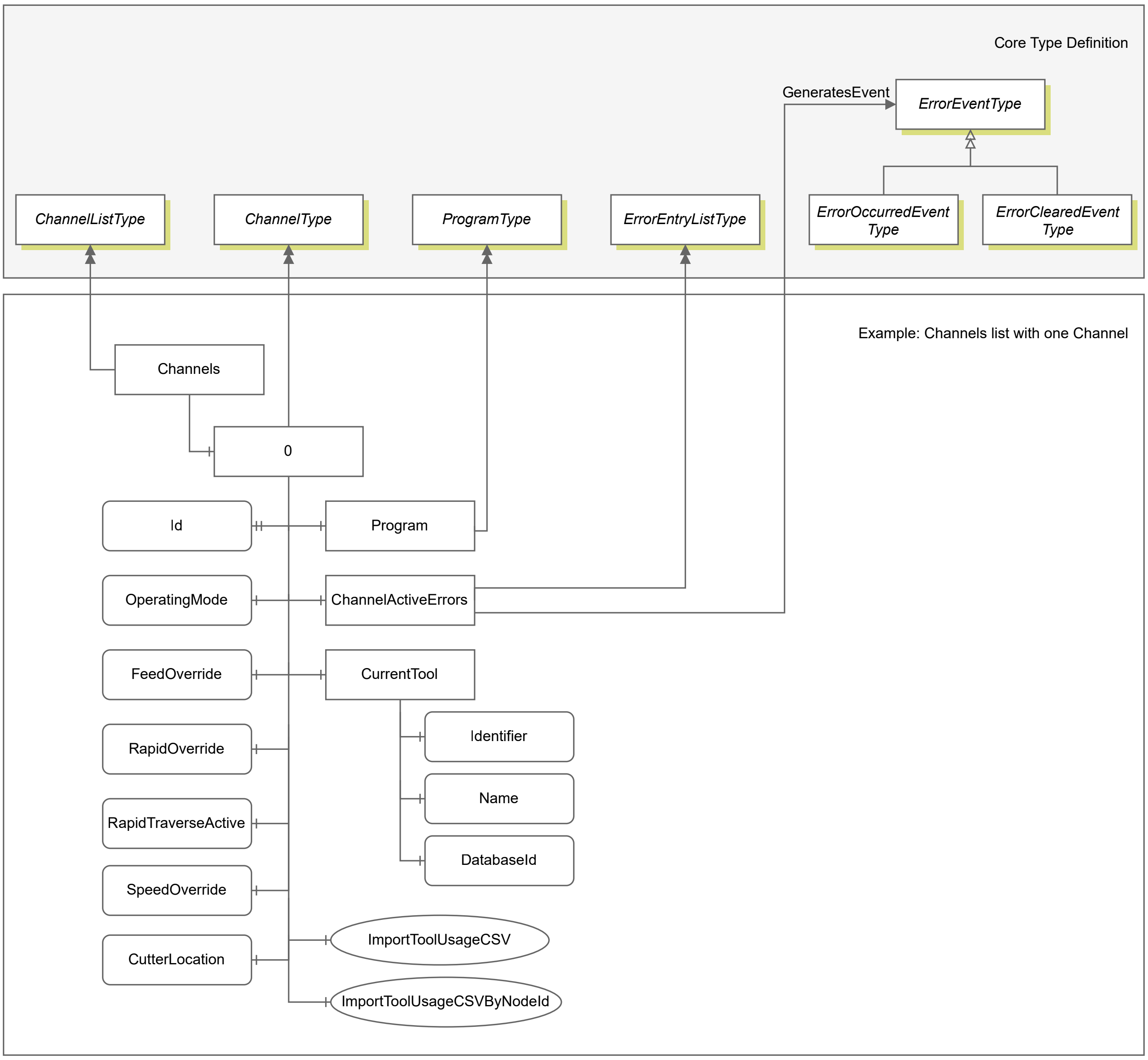

Figure 10: Example *Channels* list with one *Channel 0*

A *Channel* has a list of active errors and warnings related to this machining channel and also emits the corresponding *ErrorEvents*. For the concept of errors and warnings including their representation in the address space, see 3.7 "Errors, Warnings and Notifications".

Operating Mode

The *OperatingMode* variable displays the machine's active mode of operation at the moment and also provides the possibility of changing the operating mode of the machine. Its *DataType* is *NCOperatingMode*, an enumeration with values like *Automatic* or *SingleStep* for NC program execution.

Overrides

The override values for Speed, Rapid and Feed are provided as variables of *AnalogItemType*. Their *EngineeringUnits* and *EURanges* are given as properties.

For example the *FeedOverride* variable value is given in percent as described by the *EngineeringUnits* and normally ranges between 0% and 150% as given by the *Low* and *High* value of the *EURange* property. Note that an *EURange* always is only informative — it describes the range the value is normally in. Other values outside of the given range are also possible.

Additionally, the *RapidTraverseActive* variable indicates whether rapid traverse for this channel is active, i.e. which override is active.

Changing the feed, speed or rapid override percentage by writing a new value to these variables can only be done by client users with NC.RemoteProgram rights. If a client user has insufficient rights on the control the request is denied with *BadUserAccessDenied*.

Current Cutter Location

CutterLocation describes the current cutter location using the Input Coordinate System (I-CS).

CutterLocation is a *CutterLocationArrayType* variable. Reading the value of *CutterLocation* will return an array of *CutterLocationDataType* structures with each element of the array representing one coordinate of the current cutter location in I-CS. A coordinate structure contains a *CoordinateName*, *Position* and *PositionEngineeringUnits*.

Current Tool

The *CurrentTool* object has three component variables providing identification information of the tool used in the channel.

The tool record identifier, consisting of *ToolNumber* and *ToolIndex*, is provided by the *Identifier* variable. Additionally the *Name* and *DatabaseId* of the current tool are given. (For a description of the terms including the difference between a tool's number and database ID, see "Terms", Page 24.)

With the tool record identifier the data of the tool can be accessed using the methods provided by the *ToolDataManagement* component of a *Machine* (see 3.10 "Tool Data Management").

3.6 NC Program Execution Monitoring and Control

ProgramType Overview

The *Program* object groups together functionality and information related to executing NC programs on an NC *Channel*. The following sections describe the components shown in Image 11.

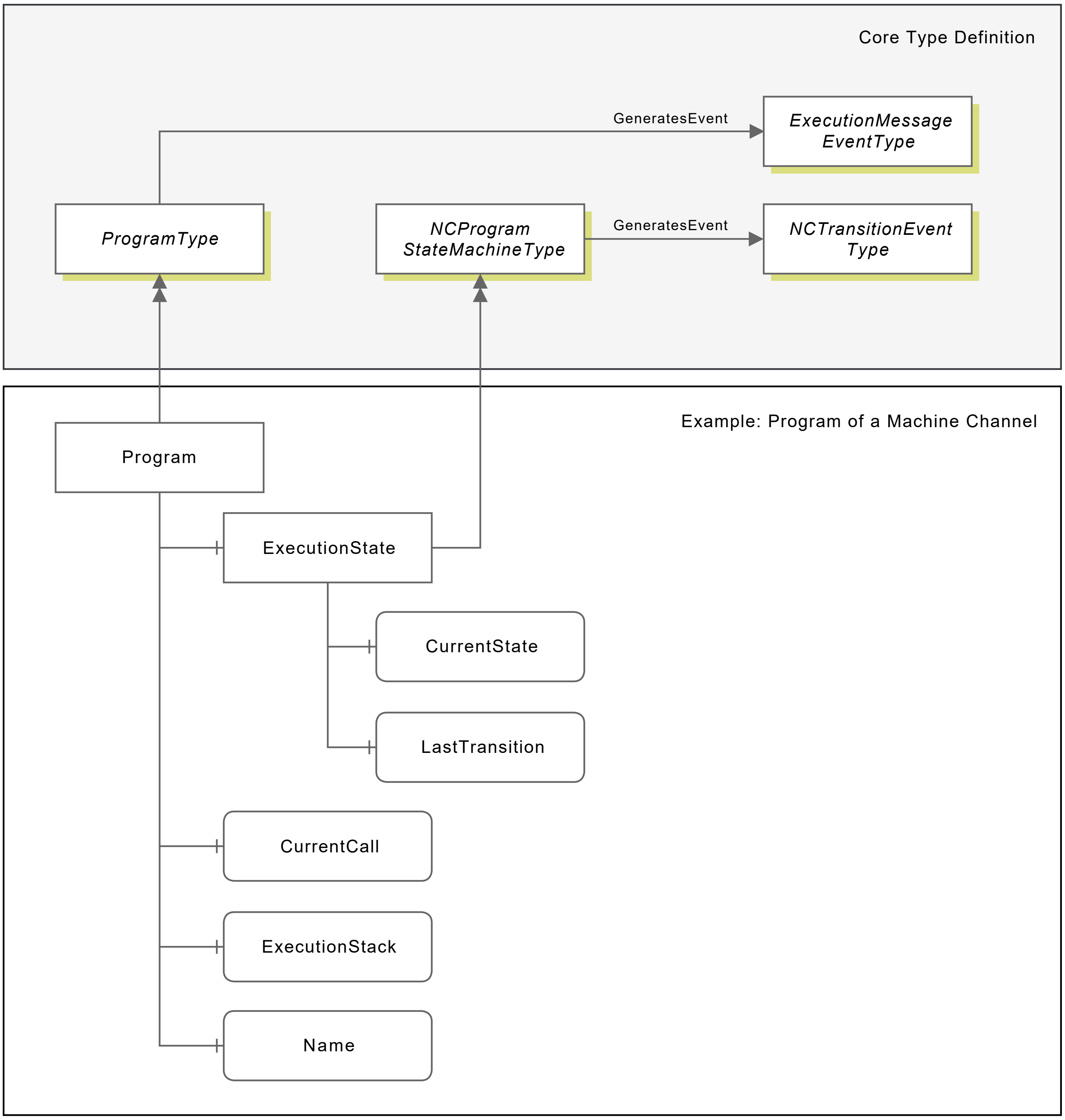

Figure 11: Example *Program* instance of type *ProgramType*

Name and Execution Call Stack of the Running Program

The *Program* object provides information about the running program with different granularity. The loaded main program is denoted as *Name* variable of *Program*. The *CurrentCall* variable contains the subprogram executed at the moment, if one is called. Next to that also the full program execution stack together with the block numbers and block content can be read from the *ExecutionStack* variable.

ExecutionStack is a *ProgramPositionArrayType* variable. Reading the value of *ExecutionStack* will return an array of *ProgramPositionDataType* structures with each element of the array representing one program position beginning with the main program and containing all called subprograms down to the currently active subprogram of the NC program. A program position structure contains a *ProgramName*, *BlockNumber*, *BlockContent* and *CallStackLevel*.

Program Execution State Machine

The execution of a program can be monitored and controlled through the *ExecutionState* component of *Program*. The *ExecutionState* is an *NCProgramStateMachine* with the states and transitions as shown in Image 12. The *NCProgramStateMachineType* is a subtype of *FiniteStateMachineType*.

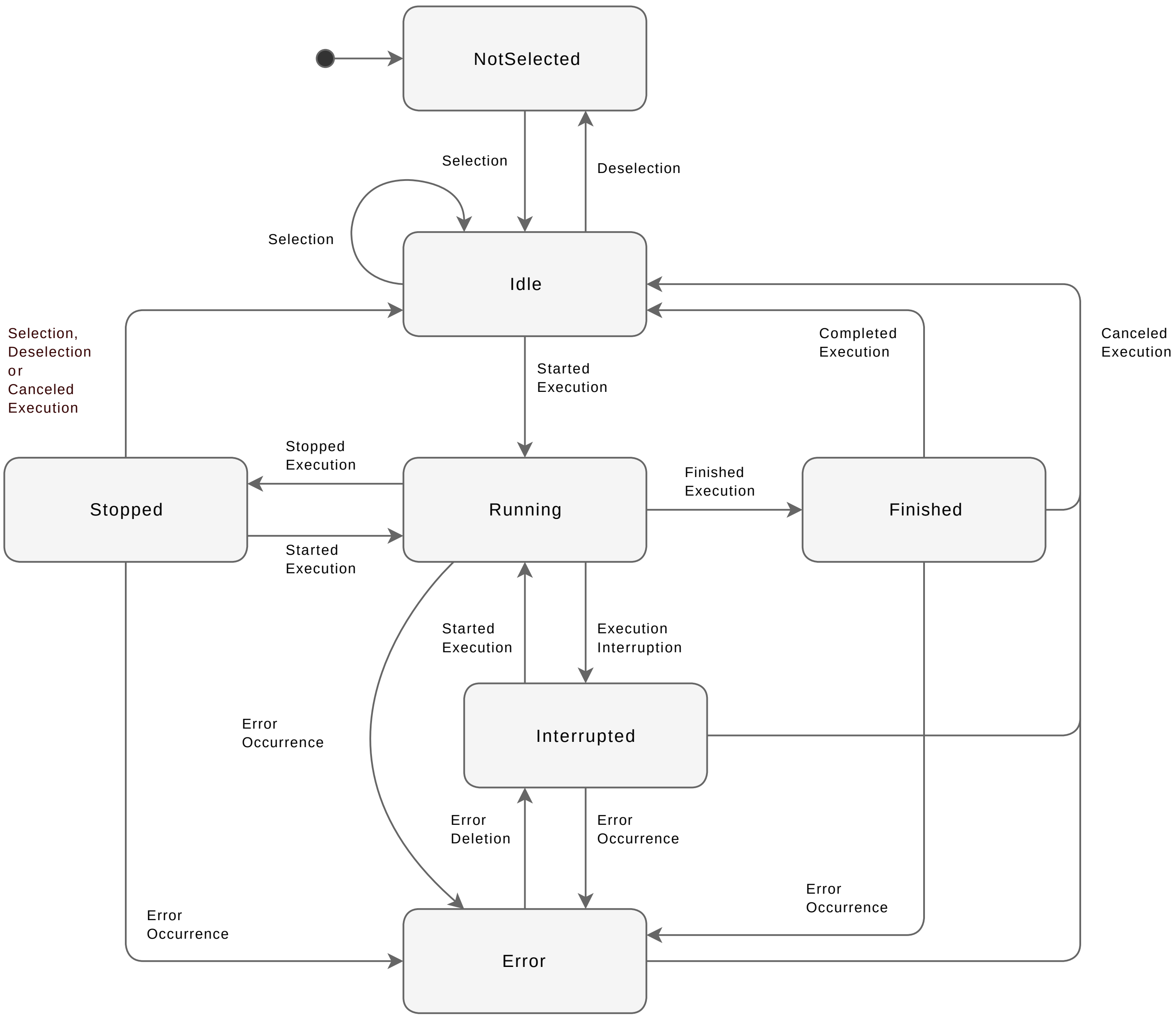

Figure 12: *States* and *Transitions* of an *NCProgramStateMachine*

As defined in OPC UA Part 16, the *CurrentState* and *LastTransition* variables provide information about the current state of the state machine regarding the program execution (e.g., *Running*) and the transition that is completed to get to this state (e.g., *IdleToRunning*). The *Id* properties of the *CurrentState* and *LastTransition* variables contain the *NodeId* of the corresponding *State* and *Transition* in the *NCProgramStateMachineType*.

For a detailed description of all states, transitions and events see 4.15 "NCProgramStateMachineType".

When a transition from one state to another occurs, an event of type *NCTransitionEventType* is reported by the *ExecutionState* object. So state changes of the program execution can be monitored using subscriptions to *NCTransitionEvents* on the *ExecutionState* object or data change subscriptions to the *CurrentState* variable value.

The *ExecutionState* state machine provides with its *Methods* like *SelectProgram*, *Start*, *Stop* and *Cancel* the functionality to control the program execution. For a more detailed description of these *Methods*, their signature, and possible result codes, see 4.15 "NCProgramStateMachineType".

Program Execution and File System

If the OPC UA NC Server also provides access to the machine's file system (see 3.9 "File System Access"), the NC program can also be selected using the *SelectProgramByNodeId* method. Next to that, the *Name* variable is extended with a property *FileNodeId* containing the *NodeId* of the *File* node corresponding to the selected NC program file within the machine's file system (if it is exposed in the address space).

For more information about file system access and file transfer to and from the machine, see 3.9 "File System Access" and 9 "Machine File System Access".

Emitted Events

The *NCTransitionEvents* of the *ExecutionState* state machine are propagated to the *Program* node. Along with that the *Program* node emits events of type *ExecutionMessageEventType*. These events are reported whenever a specific NC program command (FN 38: SEND) is executed. The *Message* property of the event contains the string defined in this command. It can be used for communication between an NC program and other remote applications, for example.

3.7 Errors, Warnings and Notifications

Errors, Warnings and Notifications of an NC

The control displays errors in the header of the screen, until it is cleared or replaced by a higher-priority error (higher error class). The complete information on all pending error messages is given in the error window.

Information about active errors and warnings (including notifications) are part of the Core Information Model. Service files can be created to analyze possible issues of the machine in detail or to request assistance from customer service. See Diagnostics at 4.1 "MachineType" and 4.23 "ServiceFileInterfaceType".

In this specification the term error is used as generic term for errors, warnings, and notifications. All are represented as instances of *ErrorEntryType* and *ErrorEventType*. The *Class* property describes the classification, for example, of an *ErrorEntry* as warning.

ErrorInterfaceType, ErrorLists and ErrorEvents

The *Errors* component of the *Machine* is an instance of *ErrorInterfaceType* and provides the information and functionality for error handling. Image 13 contains an overview of the related types and an example *ErrorInterfaceType* instance.

The *AllActiveErrors* component is of type *ErrorEntryListType* and contains one object of type *ErrorEntryType* for every error present at the machine. The list is populated at the latest when the control is fully initialized, so the *Machine's State* state machine reaches the state *NCIsAvailable*, and is kept up to date as long as there is a connection between the *Server* and the control.

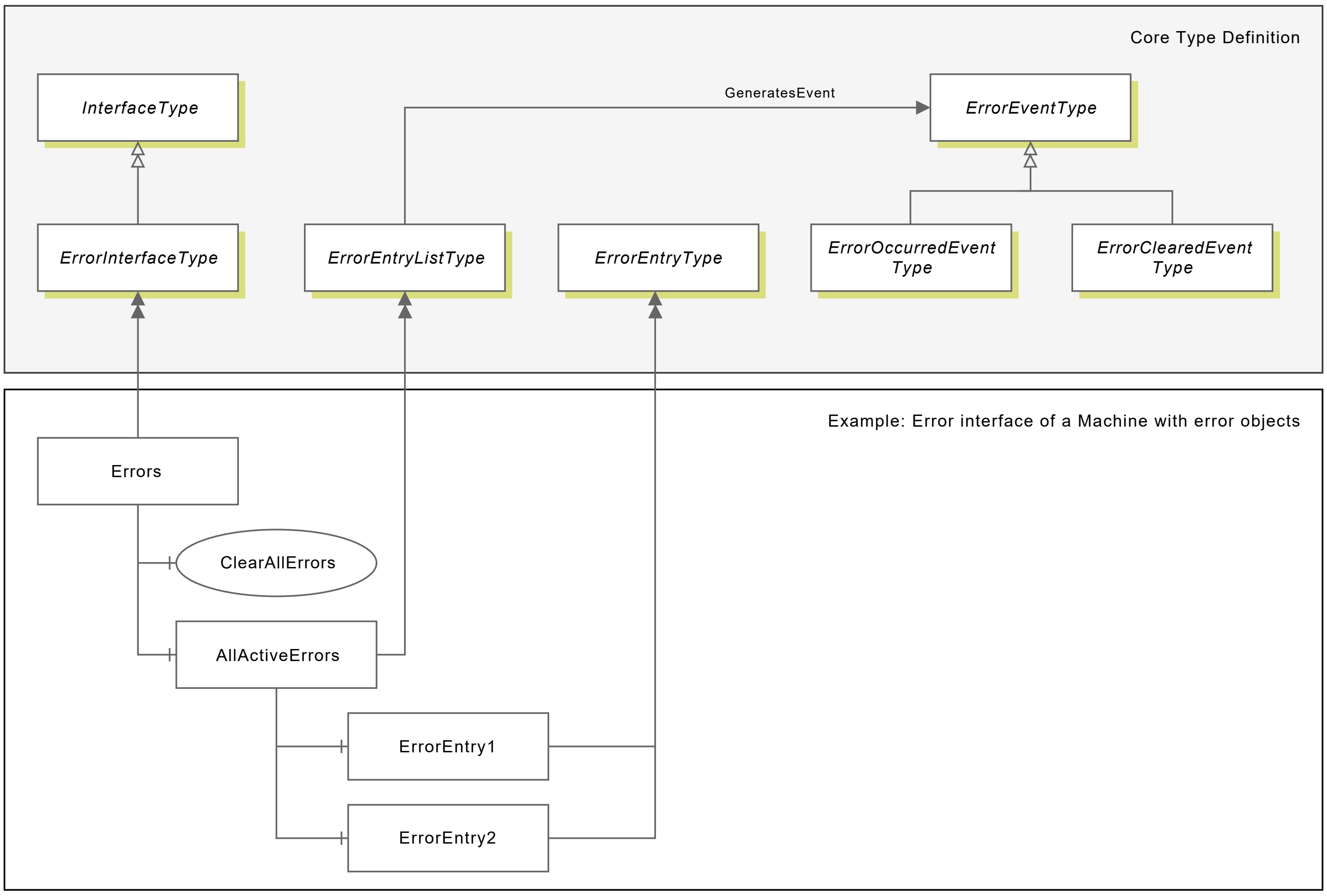

Figure 13: *Error*-related types and example instance

After the initialization, events of type *ErrorOccurredEventType* and *ErrorClearedEventType* are reported at the list object when a new error occurs or is cleared at the machine. The events are propagated upwards via the *Error* and *Machine* node to the *Server* node.

An error is represented as *ErrorEntry* object (instance of *ErrorEntryType*) and has several properties with information and details about the error. The corresponding event contains the same information and a property referring to the error object in the address space.

Some errors are related to a specific machining channel. The *Channel* property of an error contains the *ID* of this channel. Additionally each *Channel* has a *ChannelActiveError* component of *ErrorEntryListType* that provides a list of errors related to only this channel, using *Organizes* references to *ErrorEntry* nodes in the *AllActiveErrors* list.

ErrorEvents of channel-related errors are reported at the *ChannelActiveError* node of the channel and forwarded to the *AllActiveErrors* node and the channel node by *HasNotifier* references. So channel-related errors are also emitted at the channel node and the *Channels* list of a *Machine*.

Clearing an Error

Each error object has a *Clear* method which can be used to clear the error at the machine if this is possible.

But it is not possible to clear an error in every situation. If it is, for example, an emergency stop and the reason is still valid, the error cannot be cleared. When a *Clear* method is called, the *Server* sends a clear command to the NC. When this method call returns a *Good* result code, this indicates that the clear command was sent successfully, but does not necessarily mean the error has been cleared.

When an error is cleared an *ErrorClearedEvent* is reported for the corresponding *ErrorEntry* object. Immediately afterwards the *ErrorEntry* node is removed from the list.

The *ClearAllErrors* method of the *Error* interface has the same behavior. All errors which can be cleared at the moment are cleared, but errors for which a reason is still present remain active.

Severity Levels of ErrorEvents

The severity of an *ErrorOccuredEvent* and the related *ErrorClearedEvent* depends on the *Class* of an error. The following Table 10 lists the possible severity values of *ErrorEvents*.

Table 10: ErrorEventSeverityLevels

Severity	Description	Error Class
900	Errors with highest impact, the control is reset	Reset
800	Errors resulting in the machine switching to emergency stop mode	EmergencyStop
700	Program execution is aborted because of this error	ProgramAbort
600	Program execution is stopped because of this error	ProgramHold
500	Axis movements are stopped because of this error	FeedHold
400	Errors without impact on the machine's execution	Error
300	Warnings without impact on the machine's execution	Warning
200	Informational events without impact on the machine's execution	Info
100	Notifications for information, not shown in the event viewer/error list at the machine's GUI	Note
1	Not an error, lowest severity	None

3.8 Operating Times of the Machine

Several operating times, like the machine and control's up-time or the summarized program execution time, are provided in the *OperatingTimes* object of *Machine*, see 4.7 "OperatingTimesType". The operating times can be used to monitor the overall usage of the machine.

The operating time variable's data type is *Duration* and the values are updated once every minute and have a resolution of 1ms.

3.9 File System Access

Introduction

OPC UA NC Servers supporting the Core Information Model from Version 1.01 onwards can expose parts of the machine's file system in their *AddressSpace* as component of the *MachineType* instance *Machine*. Depending on the access rights, an OPC UA client user can access the different exposed file system partitions (TNC and PLC partition) of the machine.

Following the definition of "File Transfer" in OPC UA Part 20, the entry point to the file system representation is a *FileDirectoryType* instance node with the *BrowseName FileSystem*. Below the *FileSystem* node, nodes of the OPC UA standard types *FileDirectoryType* and *FileType* provide access to the exposed parts of the file system (e.g., to browse existing files and transfer files to or from the machine). Image 14 shows an example of what this can look like.

Access to the machine's file system can, for example, be used to transfer NC programs to or from the machine or to download service files. (Service files can be used to analyze issues at the control and are generated by the control in case of a crash or a manual request. The Core Information Model Version 1.05 adds the possibilty of creating a service file using the *ServiceFiles* component at *Machine Diagnostics*, see 4.23 "ServiceFileInterfaceType".)

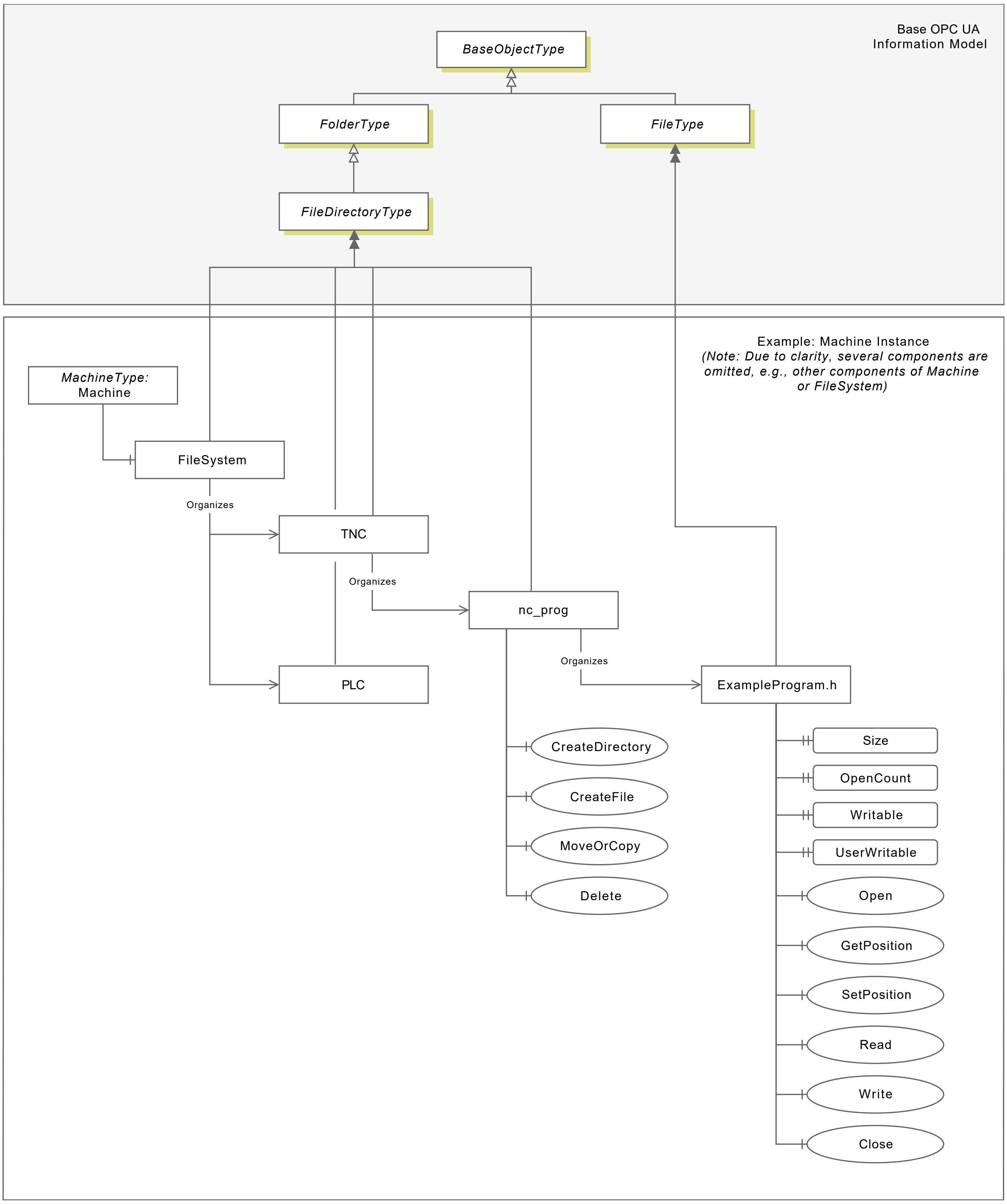

Figure 14: Shortened example of a *Machine* instance with *FileSystem*

References within this Document

9 "Machine File System Access" gives in 9.2 "FileDirectoryType" and 9.3 "FileType" an overview about how the machine's file system is exposed using the OPC UA standard types and which functionality is provided. Since not every item can be supported by the OPC UA NC Server, some restrictions and server-specific adaptations are also described.

Additional details about access to the machine's file system, like handling of the access rights, are given in chapter 9 "Machine File System Access". It also summarizes how the file system related functionality connects the other parts of the Core Information Model.

Important warnings are listed and several useful hints for OPC UA client application developers are provided in 9.4 "Warnings and Important Hints".

To simplify the descriptions within this documentation, the term *File System* is used here to describe all the nodes in the hierarchy below the *Machines FileSystem* component. As already known, the terms *File* or *FileDirectory* describe a node instance of type *FileType* or *FileDirectoryType* .

3.10 Tool Data Management

ToolDataManagementType Overview

The *Machine* component of type 4.16 "ToolDataManagementType" provides access to the machine's tool data with the following components.

- The *ToolDataRepresentation* of type 4.17 "ToolDataRepresentationType" describes all data items of the machine and groups them into categories. It lists the available tool types and links them with the data items that belong to the tool type.
- The methods provided by the *ToolDataAccess* component can be used to access the data of tools (e.g., to create new tool records, read the data of a tool and update it). Detailed information about each method is given at 4.16 "ToolDataManagementType".
- Tool data events like the 5.9 "ToolLockedEventType" are emitted at the *Notifications* node.
- The *LocalDataSet* component of type 4.22 "LocalToolDataSetType" provides the functionality to create a local copy of the machine's tool data at the OPC UA client side and keep it synchronized.
- The *Validation* component provides the possibility of checking 3D model files for use at the control (e.g., the quality of the model). See 3D Model Files for Tools

An example instance of 4.16 "ToolDataManagementType" is given in Image 15.

The tool data management is designed in a use case-oriented manner. Are you missing any functionalities for your use cases? Let us know:

OPCUA-NC-docu@heidenhain.de

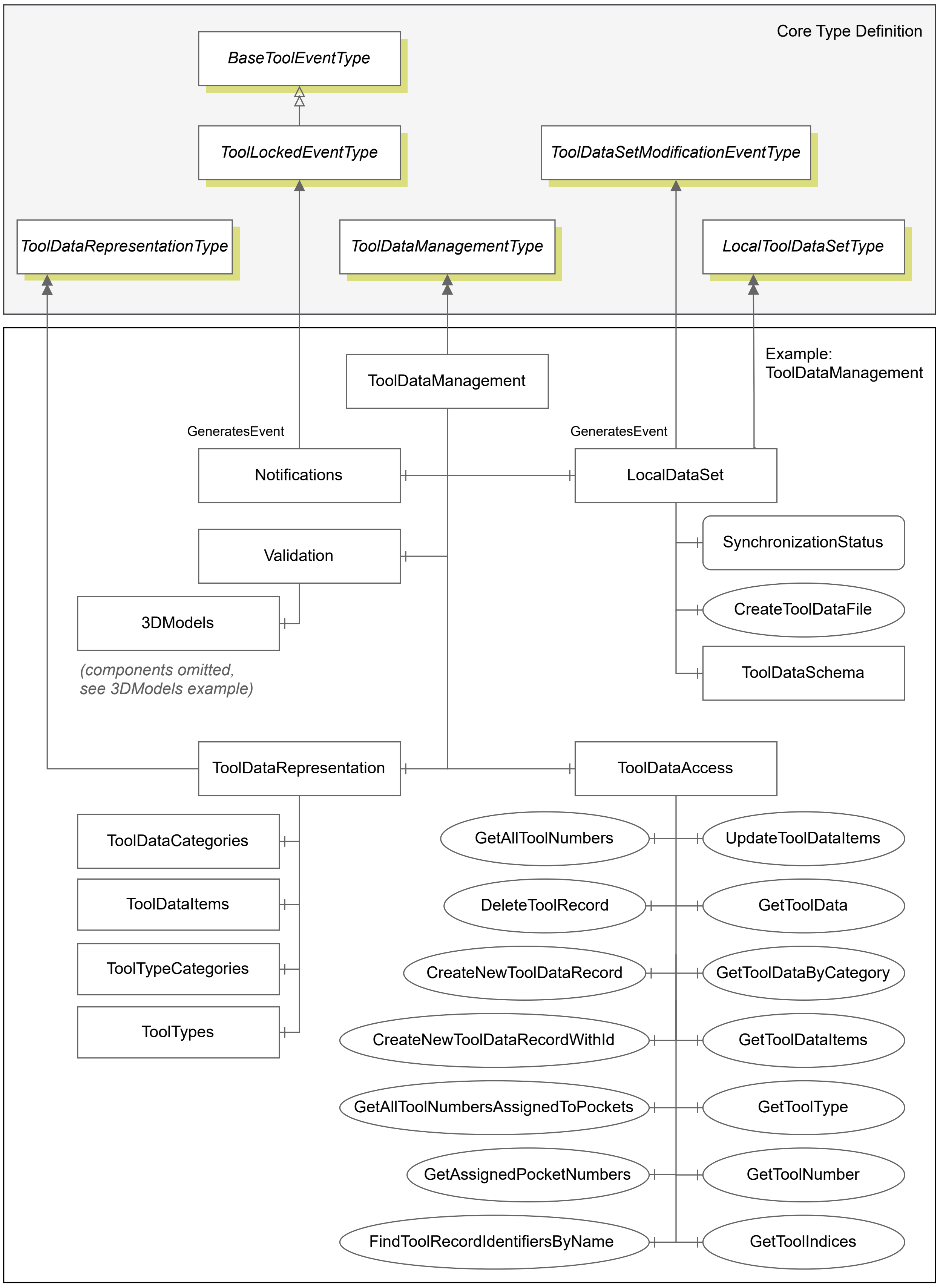

Figure 15: Example *ToolDataManagementType* instance

Description of the Machine's Tool Data

The *ToolDataRepresentation* component of type 4.17 "ToolDataRepresentationType" describes the tool data and tool types of the machine. Image 16 illustrates the concept with an example.

ToolDataRepresentationType components:

- *ToolDataItems* lists all tool data items of the machine. Each instance of 4.19 "ToolDataItemType" provides information about the data item, for example, the data type, default value, description, and more metadata like the value range or engineering units.
  Examples are the *Name* with String data type and a *0:MaxStringLength* or the *Length* using a *0:BaseAnalogType ValueDescription* variable. It provides, next to the Double data type and default value, the *0:EngineeringUnits* millimeter and values at *0:InstrumentRange* and *0:ValuePrecision*.
  If a *ToolDataItem* denotes files at a predefined location, the *FileLocation* refers to the directory in the *FileSystem*. Examples are the 3D model files for tools, see also 3D Model Files for Tools.
- *ToolDataCategories* lists groups of data items using the 4.18 "ToolDataCategoryType" (e.g., the *Identification* or *Geometry* data category).
- *ToolTypes* lists all available tool types as instances of type 4.21 "ToolTypeDescriptionType" (e.g., *MILL_TORUS* (the toroid cutter), or *TURN_RECESS* (the recessing tool)).
- *ToolTypeCategories* lists technology-based groups of tool types using 4.20 "ToolTypeCategoryType" (e.g., *MillingTools*).
- Every tool type is connected with its relevant data items with an *0:AssociatedWith* reference.

See Annex A: Tool Data Reference for an overview of the possible tool data items and tool types.

Depending on the control model and NC software version, different tool types and data items are available. It is also possible that new *AssociatedWith* references are added (e.g., due to a new NC software version that adds support of a process technology for more tool types). So new *AssociatedWith* references are added to link the relevant data items to the types. The *ToolDataRepresentation* component of the *Machine* always lists the available data items and tool types with their relations.

Machine manufacturers can extend the list of data items with their own additional tool data. These data items are referenced by a *ManufacturerExtension* data category.

The list of tool data items, including their properties, and the tool types are created based on the current configuration of the machine. They are instantiated when the *Machine* reaches the *State NCIsAvailable* the first time.

Changes to the definition of the tool data are not applied after the creation of the nodes during runtime of the server. Normally the definition of the tool data does not change. Machine manufacturers usually extend and adapt the tool data definition while commissioning the machine or when integrating software updates. The server must be restarted to apply the changes.

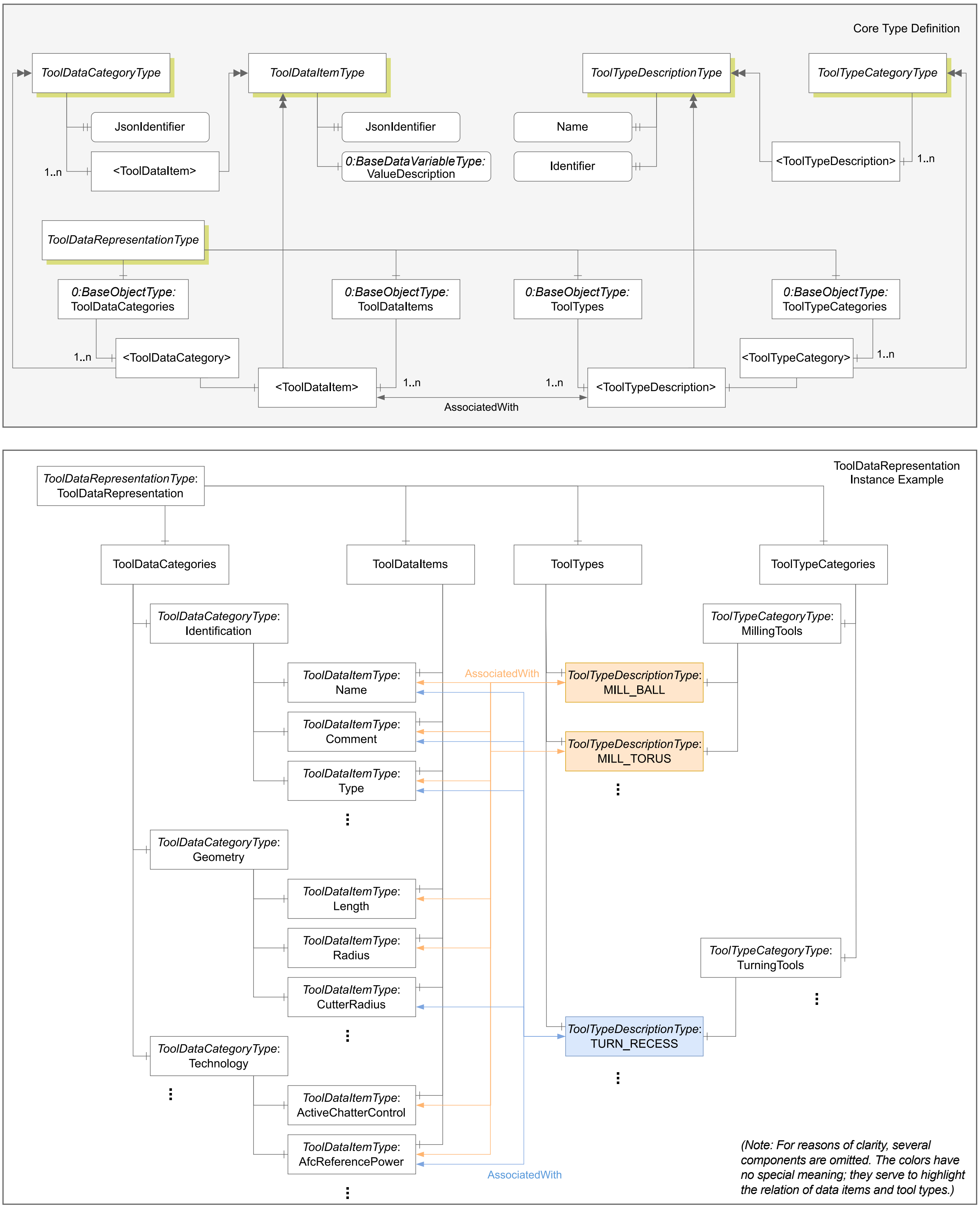
Core Type Definition
ToolDataCategoryType
JsonIdentifier
1..n
<ToolDataItem>
ToolDataItemType
JsonIdentifier
0:BaseDataVariableType: ValueDescription
ToolTypeDescriptionType
Name
Identifier
ToolTypeCategoryType
<ToolTypeDescription>
1..n
ToolDataRepresentationType
0:BaseObjectType: ToolDataCategories
0:BaseObjectType: ToolDataItems
0:BaseObjectType: ToolTypes
0:BaseObjectType: ToolTypeCategories
1..n
<ToolDataCategory>
<ToolDataItem>
1..n
1..n
<ToolTypeDescription>
<ToolTypeCategory>
1..n
AssociatedWith
ToolDataRepresentation Instance Example
ToolDataRepresentationType: ToolDataRepresentation
ToolDataCategories
ToolDataItems
ToolTypes
ToolTypeCategories
ToolDataCategoryType: Identification
ToolDataItemType: Name
ToolDataItemType: Comment
ToolDataItemType: Type
ToolDataCategoryType: Geometry
ToolDataItemType: Length
ToolDataItemType: Radius
ToolDataItemType: CutterRadius
ToolDataCategoryType: Technology
ToolDataItemType: ActiveChatterControl
ToolDataItemType: AfcReferencePower
AssociatedWith
ToolTypeCategoryType: MillingTools
ToolTypeDescriptionType: MILL_BALL
ToolTypeDescriptionType: MILL_TORUS
ToolTypeCategoryType: TurningTools
ToolTypeDescriptionType: TURN_RECESS
AssociatedWith
(Note: For reasons of clarity, several components are omitted. The colors have no special meaning; they serve to highlight the relation of data items and tool types.)

Figure 16: *ObjectTypes* used for the representation of the machine's tool data including an instance example

Synchronizing a Local Tool Data Set with the Machine

Methods and events for the most common use cases are provided by *ToolDataAccess* and *Notifications*. More specific use cases can be realized on a local mirror of the machine's tool data directly at the OPC UA client application. These use cases are addressed by the *LocalDataSet* component shown in Image 15.

The *LocalDataSet* component of type 4.22 "LocalToolDataSetType" provides the functionality to create a local copy of the machine's tool data and keep it up to date using update events. This also makes it possible to synchronize the tool data base of a central tool management system with the machine.

The method *CreateToolDataFile* triggers the creation of a JSON file. The file contains the current tool data of the machine. It can be downloaded using the *FileSystem* functionality.

Events of type 5.7 "ToolDataSetModificationEventType" contain tool data update information that needs to be applied to the local copy of the tool data.

The availability and status of the functionality is provided by the *SynchronizationStatus* variable.

The *ToolDataSchema* component of type *0:FileType* represents the JSON schema that describes the created JSON tool data file.

Image 17 shows an example sequence. More details are given at 4.22 "LocalToolDataSetType".

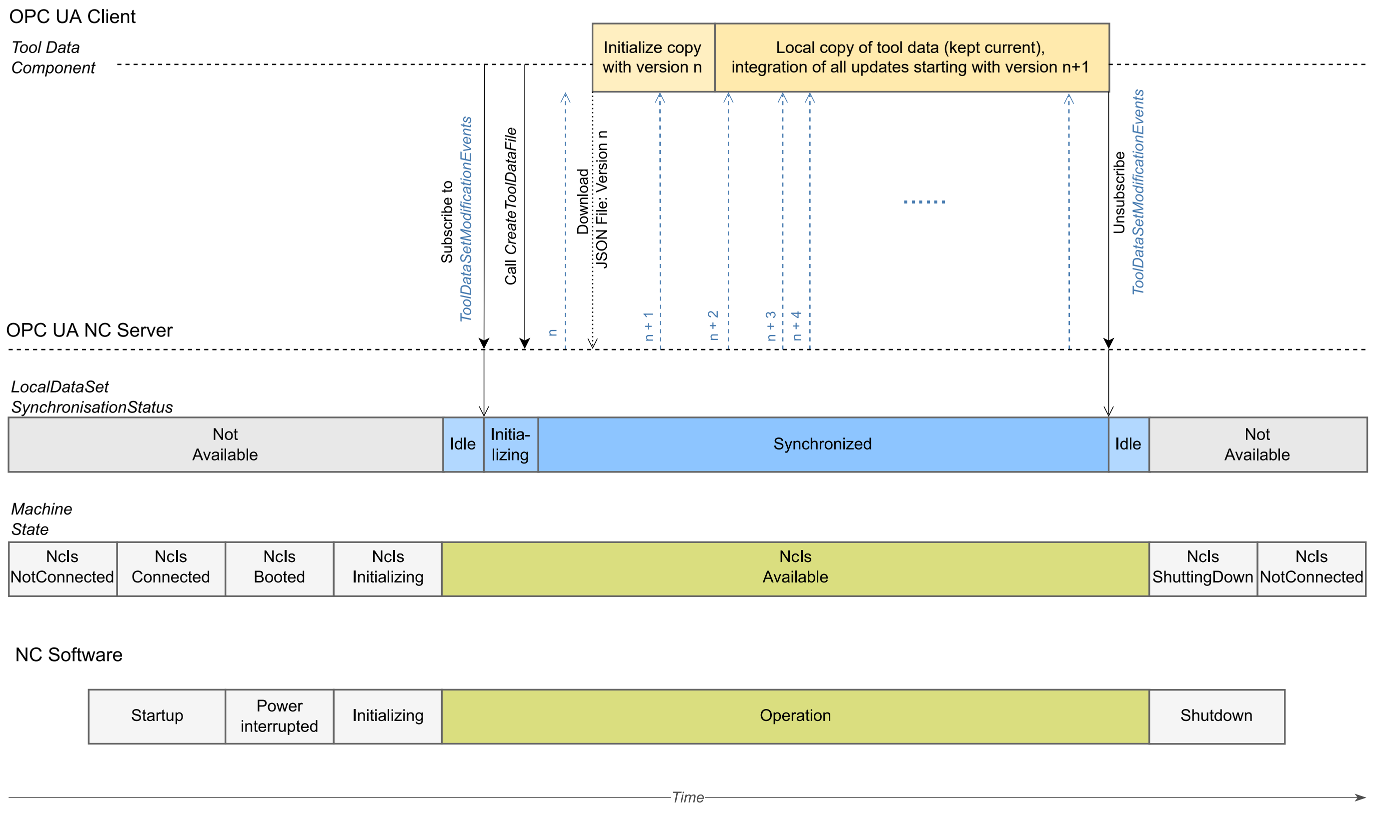

Figure 17: Example sequence how to create and synchronize a local copy of the machine's tool data at client side

3D Model Files for Tools

The control can use 3D models of tools and their carriers for material removal simulation of an NC program before execution and also during program run for collision monitoring and avoidance.

The prior validation of production-relevant data, such as the 3D models of the tools, prevents downtime in productive operation and increases process reliability when processing orders in general.

With *3DModels* , the component *Validation* provides information and additional functionality for 3D model tool data, e.g., the possibility to check several technical requirements for the 3D model file in advance.

The *RelatedToolDataItems* component refers to all supported *ToolDataItem* instances that denote 3D models (e.g., for tools and tool carriers). Image 18 shows a *ToolDataManagement* example instance with two *ToolDataItems* for the 3D models *ToolShape* and *CarrierKinematics*. The *FileLocation* and *FileLocationManufacturer* variables at *ToolDataItemType* instances denote the correct storage location of their 3D model files in the *FileSystem* of the *Machine*.

With the *Validate3DModelFile* method a *File* in the *FileSystem* can be validated for use as *ToolShape* or *CarrierKinematics*. In case of a poor result, a description for each possible validation issue can be found at the *EnumValues* property of the *Issues* argument description variable.

See Validation 3DModels in 4.16 "ToolDataManagementType", FileLocation in 4.19 "ToolDataItemType", and 3D Model Files for Tools: Validation Issues List in Annex A: Tool Data Reference for more information.

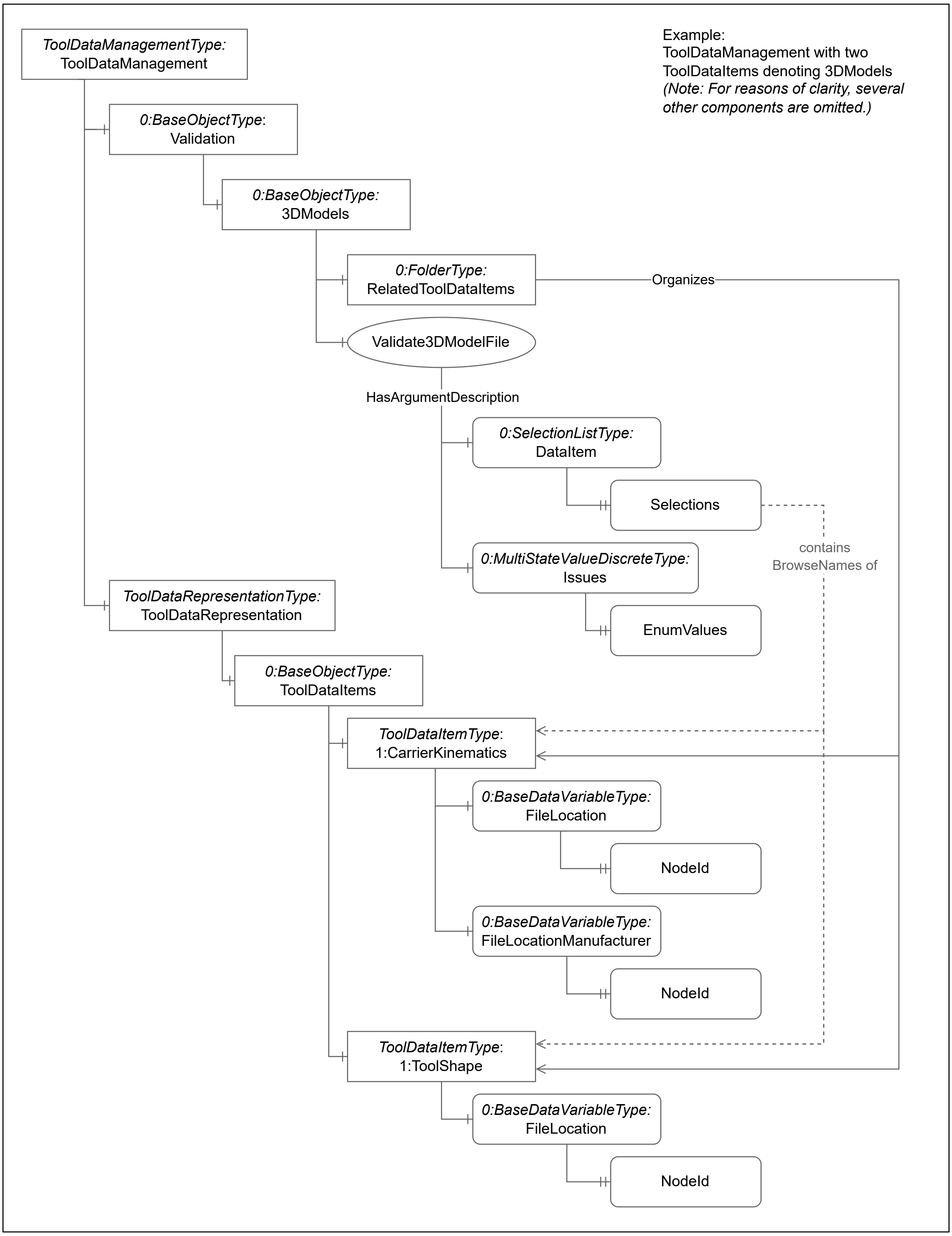
Example:
ToolDataManagement with two ToolDataItems denoting 3DModels
(Note: For reasons of clarity, several other components are omitted.)
ToolDataManagementType: ToolDataManagement
0:BaseObjectType: Validation
0:BaseObjectType: 3DModels
0:FolderType: RelatedToolDataItems
Organizes
Validate3DModelFile
HasArgumentDescription
0:SelectionListType: DataItem
Selections
contains BrowseNames of
0:MultiStateValueDiscreteType: Issues
EnumValues
ToolDataRepresentationType: ToolDataRepresentation
0:BaseObjectType: ToolDataItems
ToolDataItemType: 1:CarrierKinematics
0:BaseDataVariableType: FileLocation
NodeId
0:BaseDataVariableType: FileLocationManufacturer
NodeId
ToolDataItemType: 1:ToolShape
0:BaseDataVariableType: FileLocation
NodeId

Figure 18: Example *ToolDataManagementType* instance with *Validation* for *3DModels* including *RelatedToolDataItems*

4

OPC UA ObjectTypes

4.1 MachineType

Overview

A *MachineType* instance represents a HEIDENHAIN control-based manufacturing system and provides machine-related information, monitoring, and control functions.

For an overview of the different components and concepts, see also 3.2 "Machine Instance".

Attributes

Table 11: MachineType Definition Attributes

Attribute	Value
BrowseName	MachineType
IsAbstract	false

References

MachineType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 12: MachineType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	Error		ErrorInterfaceType	Mandatory
HasComponent	Object	State		NCStateMachine-Type	Mandatory
HasComponent	Object	ControlInfo		ControlInfoType	Mandatory
HasComponent	Object	ManufacturerInfo		ManufacturerInfo-Type	Mandatory
HasComponent	Object	Channels		ChannelListType	Mandatory
HasComponent	Object	Diagnostics		BaseObjectType	Optional
HasComponent	Object	FileSystem		FileDirectoryType	Optional
HasComponent	Object	OperatorInfo		OperatorMachine-InfoType	Mandatory
HasComponent	Object	OperatingTimes		Operating-TimesType	Mandatory
HasComponent	Object	Manufacturer-Extensions		BaseObjectType	Optional
HasComponent	Object	ToolData-Management		ToolData-ManagementType	Optional
HasComponent	Variable	HostComputerMode	Boolean	BaseDataVariable-Type	Mandatory

Additional Subcomponents

Some components of *MachineType* have additional components, which are defined in Table 13.

Table 13: MachineType Additional Subcomponents

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Diagnostics	HasComponent	Object	ServiceFiles		ServiceFileInter-faceType	Mandatory
FileSystem	HasComponent	Object	TNC		FileDirectory-Type	Optional
FileSystem	HasComponent	Object	PLC		FileDirectory-Type	Optional
Manufacturer-Extensions	Organizes	Object	<Manufacturer-Object>		BaseObject-Type	OptionalPlace-holder
Manufacturer-Extensions ▶<Manufacturer-Object>	HasComponent	Variable	<Boolean-DataVariable>	Boolean	DataItemType	OptionalPlace-holder
Manufacturer-Extensions ▶<Manufacturer-Object>	HasComponent	Variable	<Number-DataVariable>	Number	BaseAnalog-Type	OptionalPlace-holder
Manufacturer-Extensions ▶<Manufacturer-Object>	HasComponent	Variable	<String-DataVariable>	String	DataItemType	OptionalPlace-holder

Error

Error is a 4.4 "ErrorInterfaceType" instance that provides information about the errors and warnings that are currently active on the machine.

State

State is a 4.2 "NCStateMachineType" instance that provides information and monitoring possibilities of the state of the control.

ControlInfo

ControlInfo is a 4.8 "ControlInfoType" instance that provides information about the control; it is set by the control manufacturer HEIDENHAIN.

ManufacturerInfo

ManufacturerInfo is a 4.10 "ManufacturerInfoType" instance that provides information about the machine; it is set by the manufacturer of the machine.

The values of the different components can be set by the machine manufacturer within the machine's configuration at **CfgOemInfo** (no. 131600).

Channels

Channels is a 4.13 "ChannelListType" instance and lists all control channels of the machine.

For every channel of the control an object of type 4.12 "ChannelType" is instantiated as a component of the *Channels* object. The *BrowseName* and *DisplayName* of the channel instance nodes will be equal to their numeric identifier.

The *NodeIds* of component nodes of *Channels* are generated at first instantiation and are server-specific. HEIDENHAIN recommends using *BrowsePaths* and the *TranslateBrowsePathsToNodeIds* service to create machine-independent client applications. (If the data access configuration of an application is based on one fixed set of *NodeIds*, it cannot be used for different machines.)

Diagnostics

Diagnostics is a *BaseObjectType* instance and provides information and functions for machine diagnostics.

The component *ServiceFiles* provides methods to create a service file and information to download the created service file using the *FileSystem*. The *ServiceFile* functionality is always available, meaning even while the NC software is not running (e.g., during *Machine State NCIsNotConnected*). See 4.23 "ServiceFileInterfaceType" for more information.

FileSystem

FileSystem is a *FileDirectoryType* instance. It provides access to the *TNC* and *PLC* partitions of the machine's file system according to OPC UA Part 20. For an overview of file system access in OPC UA, see 3.9 "File System Access". More details, warnings and hints are given in 9 "Machine File System Access".

The access rights to files and directories via OPC UA are connected to the rights of the client user on the control.

Note that access to the *PLC* partition is restricted in general. Only users with corresponding rights on this partition have access to it. So for example, browsing the *PLC* node is blocked if the client user doesn't have sufficient rights on the *PLC* partition. A *BadUserAccessDenied* error code is returned in this case.
Further information: Setup, Testing and Running NC Programs User's Manual, or Setup and Program Run User's Manual

OperatorInfo

OperatorInfo is a 4.9 "OperatorMachineInfoType" instance that provides information about the machine set by the operator of the machine.

The values of the different components can be set within the machine's configuration at **CfgMachineInfo** (no. 131700).

OperatingTimes

OperatingTimes is a 4.7 "OperatingTimesType" instance that provides several operating time variables like, for example, the *Machine*'s up-time.

ManufacturerExtensions

ManufacturerExtensions is a *BaseObjectType* instance and provides access to machine-specific information. Objects and variables referenced by this node are configured by the machine manufacturer. They are not defined by the HEIDENHAIN Core Information Model and can differ on various machines.

Machine-specific nodes are located within at least one additional namespace of the machine manufacturer. Information about the namespaces can be found at the *Server* object's *Namespaces* list.

Depending on the machine configuration, *ManufacturerExtensions Organizes* several *BaseObjectType* instances with variable component nodes of *DataItemType* or *BaseAnalogType*. As subtypes of the abstract Number *DataType,* instances of *BaseAnalogType* can have SByte, Int16, Int32, or Double *DataType*.

The variables can also have the additional standard properties *EURange*, *InstrumentRange*, *EngineeringUnits*, *ValuePrecision* or *MaxStringLength* to provide more information about the data value.

Read- or write-access to data values of the variables requires the corresponding user rights.

See 10 "Extensions of the Machine Manufacturer" for more information.

ToolDataManagement

ToolDataManagement is a 4.16 "ToolDataManagementType" instance that provides access to the tool data of the machine.

Tool data access is possible during *Machine State NCIsAvailable*. Editing tool data requires the corresponding user right.

See also 3.10 "Tool Data Management" for an overview.

HostComputerMode

Indication that the host computer mode on the control is active or inactive. If the host computer mode is active, manual manipulation of the machine's production process is blocked to not disturb the remote control of, for example, an automation system.

4.2 NCStateMachineType

Overview

An *NCStateMachineType* instance is a state machine that represents the state of the machine (see also 3.4 "NC State Machine").

Attributes

Table 14: NCStateMachineType Definition Attributes

Attribute	Value
BrowseName	NCStateMachineType
IsAbstract	false

References

NCStateMachineType is a subtype of *FiniteStateMachineType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 15: NCStateMachineType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
GeneratesEvent	EventType	NCTransitionEventType defined in 5.4			
HasComponent	Object	NCIsNotConnected		InitialStateType	
HasComponent	Object	NCIsConnected		StateType	
HasComponent	Object	NCIsBooted		StateType	
HasComponent	Object	NCIsAvailable		StateType	
HasComponent	Object	NCIsInitializing		StateType	
HasComponent	Object	NCIsShuttingDown		StateType	
HasComponent	Object	NCIsNotConnected-ToNCIsConnected		TransitionType	
HasComponent	Object	NCIsConnectedTo-NCIsNotConnected		TransitionType	
HasComponent	Object	NCIsConnectedTo-NCIsBooted		TransitionType	
HasComponent	Object	NCIsConnectedTo-NCIsShuttingDown		TransitionType	
HasComponent	Object	NCIsBootedTo-NCIsInitializing		TransitionType	
HasComponent	Object	NCIsBooted-ToNCIsNotConnected		TransitionType	
HasComponent	Object	NCIsBootedTo-NCIsShuttingDown		TransitionType	
HasComponent	Object	NCIsAvailableTo-NCIsShuttingDown		TransitionType	
HasComponent	Object	NCIsAvailableTo-NCIsNotConnected		TransitionType	
HasComponent	Object	NCIsShuttingDown-To-NCIsNotConnected		TransitionType	
HasComponent	Object	NCIsInitializingTo-NCIsShuttingDown		TransitionType	

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	NCIsInitializingTo-NCIsAvailable		TransitionType	
HasComponent	Object	NCIsInitializingTo-NCIsNotConnected		TransitionType	
HasComponent	Variable	0:LastTransition	Localized-Text	FiniteTransition-VariableType	Mandatory

NCTransitionEventType
An event reported when the state of a 4.2 "NCStateMachineType" instance changes.

NCIsNotConnected
The state where the OPC UA *Server* does not have a connection to the control. The status code of most of the control-related data variables is set to *BadOutOfService*.

NCIsConnected
The state where the control is connected but not booted.

NCIsBooted
The state where the control is booted and power is still interrupted.

NCIsAvailable
The state where all subsystems of the control are running and initialized, and the control is available for operation or running.

NCIsInitializing
The state where the control is initializing.

NCIsShuttingDown
The state where the control is shutting down.

NCIsNotConnectedToNCIsConnected
The initial state transition when the OPC UA *Server* established a connection to the control.

NCIsConnectedToNCIsNotConnected
The state transition from *NCIsConnected* to *NCIsNotConnected*.

NCIsConnectedToNCIsBooted
The state transition from *NCIsConnected* to *NCIsBooted*. It occurs during regular startup.

NCIsConnectedToNCIsShuttingDown
The state transition from *NCIsConnected* to *NCIsShuttingDown*.

NCIsBootedToNCIsInitializing
The state transition from *NCIsBooted* to *NCIsInitializing*. It occurs after powering on the already booted machine.

NCIsBootedToNCIsNotConnected
The state transition from *NCIsBooted* to *NCIsNotConnected*. It occurs on immediate connection loss of the *Server* to the control.

NCIsBootedToNCIsShuttingDown
The state transition from *NCIsBooted* to *NCIsShuttingDown*. It occurs when shutting down without ever powering on the machine.

NCIsAvailableToNCIsShuttingDown
The state transition from *NCIsAvailable* to *NCIsShuttingDown*. It occurs during regular shutdown of the control.

NCIsAvailableToNCIsNotConnected
The state transition from *NCIsAvailable* to *NCIsNotConnected*. It occurs on immediate connection loss.

NCIsShuttingDownToNCIsNotConnected
The state transition from *NCIsShuttingDown* to *NCIsNotConnected*. It occurs during regular shutdown when the OPC UA *Server* loses the connection to the control.

NCIsInitializingToNCIsShuttingDown
The state transition from *NCIsInitializing* to *NCIsShuttingDown*.

NCIsInitializingToNCIsAvailable
The state transition from *NCIsInitializing* to *NCIsAvailable*. It occurs during regular startup.

NCIsInitializingToNCIsNotConnected
The state transition from *NCIsInitializing* to *NCIsNotConnected*. It occurs on immediate connection loss of the Server to the control.

0:LastTransition
The *ModellingRule* has been changed to *Mandatory*.

4.3 InterfaceType

Overview

InterfaceType is an abstract type without components. It is the parent type of specific control interface subtypes.

Attributes

Table 16: InterfaceType Definition Attributes

Attribute	Value
BrowseName	InterfaceType
IsAbstract	true

References

InterfaceType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 17: InterfaceType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasSubtype	ObjectType	ErrorInterfaceType defined in 4.4			

4.4 ErrorInterfaceType

Overview

An *ErrorInterfaceType* instance provides actual error information of the machine.

See also 3.7 "Errors, Warnings and Notifications" for an overview of the HEIDENHAIN control error concept and its representation in the address space.

Attributes

Table 18: ErrorInterfaceType Definition Attributes

Attribute	Value
BrowseName	ErrorInterfaceType
IsAbstract	false

References

ErrorInterfaceType is a subtype of 4.3 "InterfaceType", which means it inherits the *InstanceDeclarations* of that *Node*.

Table 19: ErrorInterfaceType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	AllActiveErrors		ErrorEntryListType	Mandatory
HasComponent	Method	ClearAllErrors			Mandatory

AllActiveErrors

AllActiveErrors is a 4.6 "ErrorEntryListType" instance and lists all errors that are currently active. It will also report events of type 5.2 "ErrorClearedEventType" and 5.3 "ErrorOccurredEventType".

ClearAllErrors

When *ClearAllErrors* is called, a command is sent to clear all errors that are present at the control at the moment. For each error that was cleared an *ErrorClearedEvent* is reported.

Note that a good status code does not indicate that all errors are cleared. If there are some errors that cannot be cleared at the moment because of, for example, physical conditions or other constrains of the control, these errors stay active.

Table 20: ClearAllErrors Attributes

Attribute	Value
BrowseName	ClearAllErrors

Table 21: ClearAllErrors Result Codes

Result Code	Description
BadOutOfService	Returned when the server has lost its internal connection to the control.

4.5 ErrorEntryType

Overview

An *ErrorEntryType* instance represents an active occurrence of an error on the machine. It contains several properties that provide detailed information about the error.

See also 3.7 "Errors, Warnings and Notifications" for an overview of the HEIDENHAIN control error concept and its representation in the address space.

Attributes

Table 22: ErrorEntryType Definition Attributes

Attribute	Value
BrowseName	ErrorEntryType
IsAbstract	false

References

ErrorEntryType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 23: ErrorEntryType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	Action	String	PropertyType	Mandatory
HasProperty	Variable	Cause	String	PropertyType	Mandatory
HasProperty	Variable	Channel	UInt32	PropertyType	Mandatory
HasProperty	Variable	Class	ErrorClass-Type	PropertyType	Mandatory
HasProperty	Variable	Group	ErrorGroup-Type	PropertyType	Mandatory
HasProperty	Variable	Number	UInt32	PropertyType	Mandatory
HasProperty	Variable	NumberAsText	String	PropertyType	Mandatory
HasProperty	Variable	Internals	String	PropertyType	Mandatory
HasProperty	Variable	Location	Error-Location-Type	PropertyType	Mandatory
HasProperty	Variable	Text	String	PropertyType	Mandatory
HasComponent	Method	Clear			Mandatory

Action

A description of possible actions that can be taken to fix the error.

Cause

The cause that triggered the error.

Channel

The ID of the channel where the error originated.

The status code of the value is *GoodNoData* if the error does not belong to a channel.

Class

Classification of the error. The value of *Class* is a 7.1 "ErrorClassType" enumeration value.

Group

The group that the error belongs to. The value of *Group* is a 7.2 "ErrorGroupType" enumeration value.

Number

The number of the error.

Each error has a number. But because of, for example, the possibility parameterized error message texts, the number is not a unique identifier of an error object.

NumberAsText

Error number as it is displayed on the control.

Internals

Internal error details as multi-line text.

Location

The location of the error. The value of *Location* is a 7.3 "ErrorLocationType" enumeration value.

Text

Error message text that is displayed on the controls error list.

Clear

By calling *Clear* a request is sent to the control to clear the error.

Not every error can be cleared immediately. If, for example, the cause or condition of the error like an open door is still valid, the error cannot be cleared. The return code is good if the *Clear* request was successfully sent to the control; it does not indicate that the error was cleared.

If the error is cleared it is removed from all 4.6 "ErrorEntryListType" instances and an *ErrorClearedEvent* is reported.

Table 24: Clear Attributes

Attribute	Value
BrowseName	Clear

Table 25: Clear Result Codes

Result Code	Description
BadNodeIdUnknown	Returned when the error has already been cleared, and the error entry node no longer existed when the method was called.
BadObjectDeleted	Returned when the error was already cleared when the server tried to perform the clear, but the address space had not yet been updated.
BadInvalidState	Returned when sending the *Clear* request to the control is currently not possible.
BadOutOfService	Returned when the server has lost its internal connection to the control.

4.6 ErrorEntryListType

Overview

An *ErrorEntryListType* instance groups 4.5 "ErrorEntryType" instances together in a list either by *HasComponent* or *Organizes* references. An *ErrorEntry* is a component of exactly one *ErrorEntryList* and additionally it can be organized by other *ErrorEntryLists*.

Next to creation and deletion of the *ErrorEntries* below the *ErrorEntryList*, events of type 5.1 "ErrorEventType" (respectively events of the subtypes 5.3 "ErrorOccurredEventType" and 5.2 "ErrorClearedEventType") are emitted from *ErrorEntryListType* instances.

Attributes

Table 26: ErrorEntryListType Definition Attributes

Attribute	Value
BrowseName	ErrorEntryListType
IsAbstract	false

References

ErrorEntryListType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 27: ErrorEntryListType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Organizes	Object	<OrganizedError-Entry>		ErrorEntryType	OptionalPlace-holder
GeneratesEvent	EventType	ErrorEventType defined in 5.1			
HasComponent	Object	<ComponentError-Entry>		ErrorEntryType	OptionalPlace-holder

<OrganizedErrorEntry>

This is a placeholder for 4.5 "ErrorEntryType" instances that are added to the error entry list with an *Organizes* reference.

ErrorEventType

ErrorEventType is an abstract *Event* indicating a change in a 4.6 "ErrorEntryListType" instance. It contains also all information of the related error object.

See also 3.7 "Errors, Warnings and Notifications" for an overview of the HEIDENHAIN control error concept and its representation in the address space.

<ComponentErrorEntry>

This is a placeholder for 4.5 "ErrorEntryType" instances that are added to the error entry list with a *HasComponent* reference.

4.7 OperatingTimesType

Overview

An *OperatingTimesType* instance provides several operating time variables of a machine.

The version of the server in current NC software versions does not support subscription functionality for operating time variables.

Attributes

Table 28: OperatingTimesType Definition Attributes

Attribute	Value
BrowseName	OperatingTimesType
IsAbstract	false

References

OperatingTimesType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 29: OperatingTimesType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Variable	ControlUpTime	Duration	BaseDataVariable-Type	Mandatory
HasComponent	Variable	ProgramExecution-Time	Duration	BaseDataVariable-Type	Mandatory
HasComponent	Variable	MachineUpTime	Duration	BaseDataVariable-Type	Mandatory

ControlUpTime

Up-time of the control. This is the cumulative time that the control has been turned on since installation.

ProgramExecutionTime

Working time of the machine since installation. This is the cumulative machining time since installation (program is running in *Automatic* or *Single Block* operating mode).

MachineUpTime

Up-time of the machine. This is the cumulative time that the machine has been on (no emergency stop) since installation.

4.8 ControlInfoType

Overview

A *ControlInfoType* instance provides information about the control of the machine.

See 3.3 "General Information about the Machine" and 4.1 "MachineType" for more information.

Attributes

Table 30: ControlInfoType Definition Attributes

Attribute	Value
BrowseName	ControlInfoType
IsAbstract	false

References

ControlInfoType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 31: ControlInfoType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	Model	String	PropertyType	Mandatory
HasProperty	Variable	Manufacturer	String	PropertyType	Mandatory
HasComponent	Object	SoftwareVersions		SoftwareVersion-ListType	Mandatory
HasComponent	Variable	SIK	String	BaseDataVariable-Type	Mandatory

Model

The control model name (e.g., TNC 640).

Manufacturer

The name of the manufacturer of the control.

SoftwareVersions

SoftwareVersions is a SoftwareVersionListType instance and lists software versions of the control.

SIK

The SIK is the unique identification number of a board inserted in the MC. The SIK number is necessary to unlock additional software options.

4.9 OperatorMachineInfoType

Overview

An *OperatorMachineInfoType* instance provides information about the machine set by the operator.

See 3.3 "General Information about the Machine" and 4.1 "MachineType" for more information.

Attributes

Table 32: OperatorMachineInfoType Definition Attributes

Attribute	Value
BrowseName	OperatorMachineInfoType
IsAbstract	false

References

OperatorMachineInfoType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 33: OperatorMachineInfoType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	ContactEmail	String	PropertyType	Mandatory
HasProperty	Variable	ContactPhone-Number	String	PropertyType	Mandatory
HasProperty	Variable	Department	String	PropertyType	Mandatory
HasProperty	Variable	Image	Image	PropertyType	Mandatory
HasProperty	Variable	InventoryNumber	String	PropertyType	Mandatory
HasProperty	Variable	Location	String	PropertyType	Mandatory
HasProperty	Variable	MachineNickname	String	PropertyType	Mandatory
HasProperty	Variable	Responsibility	String	PropertyType	Mandatory

ContactEmail

The e-mail address of the responsible person or department.

ContactPhoneNumber

The phone number of the responsible person or department.

Department

The department that the machine belongs to.

Image

An icon or image of the machine. It can be a JPG or PNG file.

Starting with NC software version 34059x-11 SP3 and 81760x-08 SP3, the *DataType* of the property is adapted according to the data type of the represented image. If an image has been configured at the control, the corresponding *Image* subtype *ImagePNG* or *ImageJPG* is used.

InventoryNumber

The resource planning inventory number of the machine.

Location

The description of the location of the machine.

MachineNickname

The nickname of the machine which makes it easily identifiable for operators.

Responsibility

The person or department responsible for the machine.

4.10 ManufacturerInfoType

Overview

A *ManufacturerInfoType* instance provides information about the machine from the machine manufacturer.

See 3.3 "General Information about the Machine" and 4.1 "MachineType" for more information.

Attributes

Table 34: ManufacturerInfoType Definition Attributes

Attribute	Value
BrowseName	ManufacturerInfoType
IsAbstract	false

References

ManufacturerInfoType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 35: ManufacturerInfoType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	BuildYear	UInt32	PropertyType	Mandatory
HasProperty	Variable	Service	String	PropertyType	Mandatory
HasProperty	Variable	ServiceEmail	String	PropertyType	Mandatory
HasProperty	Variable	ServicePhone-Number	String	PropertyType	Mandatory
HasProperty	Variable	Name	String	PropertyType	Mandatory
HasProperty	Variable	ProductFamily	String	PropertyType	Mandatory
HasProperty	Variable	ProductType	String	PropertyType	Mandatory
HasProperty	Variable	SerialNumber	String	PropertyType	Mandatory
HasProperty	Variable	Image	Image	PropertyType	Mandatory
HasProperty	Variable	InstallationTime	UtcTime	PropertyType	Mandatory
HasComponent	Object	SoftwareVersions		SoftwareVersion-ListType	Mandatory

BuildYear

The year the machine was built.

Service

The name of the department or engineer of the manufacturer responsible for servicing this machine.

ServiceEmail

The e-mail address of the department or engineer of the manufacturer responsible for servicing this machine.

ServicePhoneNumber

The telephone number of the department or engineer of the manufacturer responsible for servicing this machine.

Name

The name of the manufacturer of the machine.

ProductFamily

The name of the brand within the machine manufacturers group.

ProductType

The product type of the machine, containing the production series and size of the machine often written on the machines front for example.

SerialNumber

The serial number of the machine.

Image

The icon or image of this machine type or manufacturer, it can be a JPG or PNG file.

Starting with NC software version 34059x-11 SP3 and 81760x-08 SP3, the *DataType* of the property is adapted according to the data type of the represented image. If an image or icon has been configured at the control, the corresponding *Image* subtype *ImagePNG* or *ImageJPG* is used.

InstallationTime

The date and time of installation of the machine. (The installation time is set within the machine's configuration.)

The status code *BadSyntaxError* indicates that the given value could not be converted to UtcTime format.

SoftwareVersions

SoftwareVersions is a 4.11 "SoftwareVersionListType" instance and lists software versions of the manufacturer (e.g., the PLC version).

4.11 SoftwareVersionListType

Overview

A *SoftwareVersionListType* instance provides a list of software version variables of several components of a machine. It is used for example at the 4.8 "ControlInfoType" to describe the software of the control and their versions.

Attributes

Table 36: SoftwareVersionListType Definition Attributes

Attribute	Value
BrowseName	SoftwareVersionListType
IsAbstract	false

References

SoftwareVersionListType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 37: SoftwareVersionListType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Variable	PLC	String	BaseDataVariable-Type	Optional
HasComponent	Variable	MCU	String	BaseDataVariable-Type	Optional
HasComponent	Variable	NCKERN	String	BaseDataVariable-Type	Optional
HasComponent	Variable	SPLC	String	BaseDataVariable-Type	Optional
HasComponent	Variable	FS_MCU	String	BaseDataVariable-Type	Optional
HasComponent	Variable	FS_CCU	String	BaseDataVariable-Type	Optional
HasComponent	Variable	CCU	String	BaseDataVariable-Type	Optional
HasComponent	Variable	<SoftwareVersion>	String	BaseDataVariable-Type	OptionalPlace-holder

PLC

The software version of the PLC program from the machine manufacturer.

MCU

The software version of a main computing unit.

NCKERN

The product and version number of the NC kernel.

SPLC

The software version of the SPLC program from the machine manufacturer.

FS_MCU

The software version of a functional safety main computing unit.

FS_CCU

The software version of a functional safety control computing unit.

CCU

The software version of a control computing unit.

<SoftwareVersion>
A placeholder for machine-specific components like a second CCU or machine manufacturer-specific software (e.g., when used at 4.10 "ManufacturerInfoType").

4.12 ChannelType

Overview

A *ChannelType* instance represents an NC channel of the control. (An NC channel groups several axes and their common attributes.) It provides information, settings and functions related to the execution of an NC program.

For an overview of the concepts and functionality, see 3.5 "Machining Channel".

Attributes

Table 38: ChannelType Definition Attributes

Attribute	Value
BrowseName	ChannelType
IsAbstract	false

References

ChannelType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 39: ChannelType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	Id	UInt32	PropertyType	Mandatory
HasComponent	Object	ChannelActiveErrors		ErrorEntryListType	Mandatory
HasComponent	Object	Program		ProgramType	Mandatory
HasComponent	Object	CurrentTool		BaseObjectType	Optional
HasComponent	Variable	CutterLocation	Cutter-Location-DataType	CutterLocation-ArrayType	Mandatory
HasComponent	Variable	OperatingMode	NCOpera-tingMode	BaseDataVariable-Type	Mandatory
HasComponent	Variable	FeedOverride	UInt32	AnalogItemType	Mandatory
HasComponent	Variable	SpeedOverride	UInt32	AnalogItemType	Mandatory
HasComponent	Variable	RapidOverride	UInt32	AnalogItemType	Mandatory
HasComponent	Variable	RapidTraverseActive	Boolean	BaseDataVariable-Type	Mandatory
HasComponent	Method	ImportTool-UsageCSV			Mandatory
HasComponent	Method	ImportToolUsage-CSVByNodeId			Optional

Additional Subcomponents

Some components of the *ChannelType* have additional components, which are defined in Table 40.

Table 40: ChannelType Additional Subcomponents

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
CurrentTool	HasComponent	Variable	DatabaseId	String	BaseDataVariable-Type	Mandatory
CurrentTool	HasComponent	Variable	Identifier	ToolRecord-Identifier-DataType	BaseDataVariable-Type	Mandatory
CurrentTool	HasComponent	Variable	Name	String	BaseDataVariable-Type	Mandatory

Id
The numeric identifier of the machining channel.

ChannelActiveErrors
ChannelActiveErrors is a 4.6 "ErrorEntryListType" instance and lists all error entries that are currently active on the NC channel.

If an error or warning on the machine related to this NC channel occurs or is cleared the related event of type 5.1 "ErrorEventType" is also emitted from the *ChannelActiveErrors* node. See 3.7 "Errors, Warnings and Notifications" for a description of the concept of channel-related errors and their representation in the address space.

CurrentTool
The *CurrentTool* component provides information to identify the currently used tool respectively the record.

The *Identifier* variable contains the tool record identifier of 7.10 "ToolRecordIdentifierDataType", consisting of the tool number and the index. The *Name* variable provides the index-specific name of a tool. (See also "Terms", Page 24.)

The tool record identifier can be used to access the data of the tool. See 4.16 "ToolDataManagementType" ToolDataAccess and 3.10 "Tool Data Management".

If the *DatabaseId* is managed as recommended by HEIDENHAIN, the variable contains the unique identifier of the physical tool from the central tool (data) management system. The *DatabaseId* is independent of the tool record identifier and equal for all indices of a tool.

Program
Program is a 4.14 "ProgramType" instance providing functions and information about the NC program of this channel.

CutterLocation
The *CutterLocation* provides the coordinates of the current location of the cutter within the input coordinate system (I-CS). The value of *CutterLocation* is an array of 7.7 "CutterLocationDataType" elements each representing one coordinate.

The version of the server in current NC software versions does not support subscription functionality for this variable.

OperatingMode
The operating mode for this channel. The value of *OperatingMode* is a 7.4 "NCOperatingMode" enumeration value.

Note that not all of the possible values of the 7.4 "NCOperatingMode" enumeration can be set by a write request (or are immediately changed by the control again). Some values cannot be set from remote in general like *Other* (return code *BadWriteNotSupported*) and some are not possible while the control is in a state that does not allow it (e.g., while a program is running in *Automatic* mode (return code *BadInvalidState*)).

FeedOverride
The current feed override percentage. The absolute feed is related to the programmed feed rate, the machine dynamics, and the feed override percentage. *FeedOverride* is a variable of *AnalogItemType* and provides its *EURange* and *EngineeringUnits*.

Changing the feed override percentage by writing a new value to this variable can only be done by client users with NC.RemoteProgram rights. If a client user has insufficient rights on the control the request is denied with *BadUserAccessDenied*.

The versions of the server support subscription functionality for this variable, starting with NC software version 18 and Service Packs 17 SP02, 16 SP03, 34059x-11 SP07, and 81760x-08 SP07.

Subscribing the *FeedOverride*:
Client applications should set an appropriate sampling interval and reduce the received notifications according to their interest using *MonitoredItem DataChangeFilter* with absolute or percent *Deadband*. See OPC UA Part 4 "MonitoredItem model" and "MonitoringFilter parameters" for *DataChangeFilter*.
Filter example: After a notification, the subsequent notification is sent if the new value differs by more than 5% from the previously sent value. Minor value variations within the specified range are not reported.

SpeedOverride
The current speed override percentage. The absolute spindle speed is related to the programmed speed and the speed override percentage. *SpeedOverride* is a variable of *AnalogItemType* and provides its *EURange* and *EngineeringUnits*.

Note that there are restrictions on which values are safe and allowed to set by a write request to this variable, for example, too small values are automatically raised to the minimum value by the control.

Changing the speed override percentage by writing a new value to this variable can only be done by client users with NC.RemoteProgram rights. If a client user has insufficient rights on the control the request is denied with *BadUserAccessDenied*.

The versions of the server support subscription functionality for this variable, starting with NC software version 18 and Service Packs 17 SP02, 16 SP03, 34059x-11 SP07, and 81760x-08 SP07.

Subscribing the *SpeedOverride*:
Client applications should set an appropriate sampling interval and reduce the received notifications according to their interest using *MonitoredItem DataChangeFilter* with absolute or percent *Deadband*. See OPC UA Part 4 "MonitoredItem model" and "MonitoringFilter parameters" for *DataChangeFilter*.
Filter example: After a notification, the subsequent notification is sent if the new value differs by more than 5% from the previously sent value. Minor value variations within the specified range are not reported.

RapidOverride

The current rapid override percentage. The absolute rapid feed is related to the configured feed rate, the machine dynamics, and the rapid feed override percentage. *RapidOverride* is a variable of *AnalogItemType* and provides its *EURange* and *EngineeringUnits*.

Changing the rapid override percentage by writing a new value to this variable can only be done by client users with NC.RemoteProgram rights. If a client user has insufficient rights on the control the request is denied with *BadUserAccessDenied*.

The versions of the server support subscription functionality for this variable, starting with NC software version 18 and Service Packs 17 SP02, 16 SP03, 34059x-11 SP07, and 81760x-08 SP07.

Subscribing the *RapidOverride*:
Client applications should set an appropriate sampling interval and reduce the received notifications according to their interest using *MonitoredItem DataChangeFilter* with absolute or percent *Deadband*. See OPC UA Part 4 "MonitoredItem model" and "MonitoringFilter parameters" for *DataChangeFilter*.
Filter example: After a notification, the subsequent notification is sent if the new value differs by more than 5% from the previously sent value. Minor value variations within the specified range are not reported.

RapidTraverseActive

Indicates whether the rapid traverse mode is active.

If an NC block with FMAX programmed is executed then rapid traverse is active. Depending on this condition, either the *FeedOverride* or the *RapidOverride* is active.

ImportToolUsageCSV

Import a tool usage file by providing the path to a tool usage CSV file present in the file system. This file can be used to import the externally calculated tool run time into the control.

Note that depending on the machine configuration, the HEIDENHAIN control system calculates the tool run time according to the actual machine dynamics and kinematics. If a precise run time is required, it should always be calculated by the HEIDENHAIN control system.

Table 41: ImportToolUsageCSV Attributes

Attribute	Value
BrowseName	ImportToolUsageCSV

Signature

```
    ImportToolUsageCSV{
        [in] String ToolUsageCSVName
    };
```

Table 42: ImportToolUsageCSV Signature

Argument	Description
ToolUsageCSVName	Path of the comma-separated file to import. The CSV file contains the tool usage of NC programs

Table 43: ImportToolUsageCSV Result Codes

Result Code	Description
BadInvalidArgument	Returned when the validation of the input argument fails or the file referred to is currently not usable as a tool usage CSV.
BadFileNotFound	Returned when the file to be loaded could not be found.
Bad	The file could not be used for the intended purpose.
BadOutOfService	Returned when the server has lost its internal connection to the control.

ImportToolUsageCSVByNodeId

Import a tool usage file by providing the *NodeId* of the *File* object instance representing the CSV file in the file system. This file can be used to import the externally calculated tool run time into the control.

Note that depending on the machine configuration, the HEIDENHAIN control system calculates the tool run time according to the actual machine dynamics and kinematics. If a precise run time is required, it should always be calculated by the HEIDENHAIN control system.

The functionality of this method is equal to *ImportToolUsageCSV*. This method is provided for convenience to simplify the usage in case the file has a representation in the *AddressSpace* of the OPC UA NC Server. That means that the corresponding *File* object instance is browsable starting at the *TNC* or *PLC* nodes of the *Machine's FileSystem*. See 9 "Machine File System Access" for more information about file system access.

Table 44: ImportToolUsageCSVByNodeId Attributes

Attribute	Value
BrowseName	ImportToolUsageCSVByNodeId

Signature

```
    ImportToolUsageCSVByNodeId{
        [in] NodeId FileNodeId
    };
```

Table 45: ImportToolUsageCSVByNodeId Signature

Argument	Description
FileNodeId	*NodeId* of the *File* object instance representing the comma-separated file to import. The CSV file contains the tool usage of NC programs.

Table 46: ImportToolUsageCSVByNodeId Result Codes

Result Code	Description
BadInvalidArgument	Returned when the validation of the input argument fails or the file referred to is currently not usable as a tool usage CSV.
BadNotFound	Returned when the provided *NodeId* is not valid or cannot be found in the *AddressS-pace*.
Bad	The file could not be used for the intended purpose.
BadOutOfService	Returned when the server has lost its internal connection to the control.

4.13 ChannelListType

Overview

A *ChannelListType* instance groups 4.12 "ChannelType" instances together in a list.
It is used, for example, at the 4.1 "MachineType".

Attributes

Table 47: ChannelListType Definition Attributes

Attribute	Value
BrowseName	ChannelListType
IsAbstract	false

References

ChannelListType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 48: ChannelListType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	<Channel>		ChannelType	OptionalPlace-holder

<Channel>

This is a placeholder for 4.12 "ChannelType" instances that are added to the channel list.

4.14 ProgramType

Overview

A *ProgramType* instance provides functions and information about NC programs.

An instance of type *ProgramType* contains information about which NC program is currently running, what part of the program is being executed, and has an *ExecutionState StateMachine* with methods to control the selection and execution of NC programs.

The *Program* reports 5.5 "ExecutionMessageEventType" events which are messages sent by the NC program on specific NC program commands.

See also 3.6 "NC Program Execution Monitoring and Control" for an overview.

Attributes

Table 49: ProgramType Definition Attributes

Attribute	Value
BrowseName	ProgramType
IsAbstract	false

References

ProgramType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 50: ProgramType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
GeneratesEvent	EventType	ExecutionMessageEventType defined in 5.5			
HasComponent	Object	ExecutionState		NCProgramState-MachineType	Mandatory
HasComponent	Variable	ExecutionStack	Program-Position-DataType	ProgramPosition-ArrayType	Mandatory
HasComponent	Variable	Name	String	BaseDataVariable-Type	Mandatory
HasComponent	Variable	CurrentCall	String	BaseDataVariable-Type	Mandatory

Additional Subcomponents

Some components of *ProgramType* have additional components, which are defined in Table 51.

Table 51: ProgramType Additional Subcomponents

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Name	HasProperty	Variable	FileNodeId	NodeId	PropertyType	Optional

ExecutionMessageEventType

ExecutionMessageEventType events are messages sent by the NC program on specific NC program commands (FN 38: SEND). It can be used for communication between an NC program and a remote application.

ExecutionState

ExecutionState is a 4.15 "NCProgramStateMachineType" instance that represents the execution status of the NC program. Additionally the *ExecutionState* contains methods to handle the NC program execution.

ExecutionStack

The *ExecutionStack* provides the call stack of the program execution. The value of *ExecutionStack* is an array of 7.8 "ProgramPositionDataType" instances, each representing a program position (see also 6.2 "ProgramPositionArray-Type").

The version of the server in current NC software versions does not support subscription functionality for this variable. (When executing NC programs with very short blocks on a fast HEIDENHAIN control, the *ExecutionStack* can change every 100 µs.) The *ExecutionStack* is not intended for continuous monitoring while a program is running, but for use when the program is stopped or interrupted.

Name

Name of the selected NC program. The selected main program does not change while subprograms or macros of the machine manufacturer are executed.

Name has an optional property *FileNodeId*. If the selected NC program file has a representation within the *Machine's FileSystem*, *FileNodeId* contains the *NodeId* of this *File* object instance. (See 9 "Machine File System Access" for more information about file system access.) If the NC program file does not have a representation within the address space (e.g., if it is not stored within the *TNC* or *PLC* partitions of the machine's file system), the value is empty with *StatusCode GoodNoData*. Note that access to this information is also protected by user access rights. If the *StatusCode* of the value is *BadUserAccessDenied*, the user does not have the right to access the corresponding file.

CurrentCall

The *CurrentCall* variable contains the name of the subprogram that is currently being executed. The variable value is empty if no subprogram is called. (Macros of the machine manufacturer are not considered.)

4.15 NCProgramStateMachineType

Overview

An *NCProgramStateMachineType* instance is a state machine that represents the execution status of an NC program. *NCProgramStateMachineType* also contains methods to handle the NC program execution.

For an overview about the representation of an NC program as OPC UA *FiniteStateMachine*, see 3.6 "NC Program Execution Monitoring and Control".

Attributes

Table 52: NCProgramStateMachineType Definition Attributes

Attribute	Value
BrowseName	NCProgramStateMachineType
IsAbstract	false

References

NCProgramStateMachineType is a subtype of *FiniteStateMachineType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 53: NCProgramStateMachineType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
GeneratesEvent	EventType	NCTransitionEventType defined in 5.4			
HasComponent	Object	Error		StateType	
HasComponent	Object	Finished		StateType	
HasComponent	Object	Idle		StateType	
HasComponent	Object	Interrupted		StateType	
HasComponent	Object	Running		StateType	
HasComponent	Object	Stopped		StateType	
HasComponent	Object	NotSelected		InitialStateType	
HasComponent	Object	ErrorToIdle		TransitionType	
HasComponent	Object	ErrorToInterrupted		TransitionType	
HasComponent	Object	FinishedToIdle		TransitionType	
HasComponent	Object	IdleToNotSelected		TransitionType	
HasComponent	Object	IdleToRunning		TransitionType	
HasComponent	Object	InterruptedToIdle		TransitionType	
HasComponent	Object	InterruptedTo-Running		TransitionType	
HasComponent	Object	RunningToError		TransitionType	
HasComponent	Object	RunningToFinished		TransitionType	
HasComponent	Object	RunningTo-Interrupted		TransitionType	
HasComponent	Object	RunningToStopped		TransitionType	
HasComponent	Object	StoppedToIdle		TransitionType	
HasComponent	Object	NotSelectedToIdle		TransitionType	
HasComponent	Object	StoppedToRunning		TransitionType	
HasComponent	Object	StoppedToError		TransitionType	
HasComponent	Object	IdleToIdle		TransitionType	

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	InterruptedToError		TransitionType	
HasComponent	Object	FinishedToError		TransitionType	
HasComponent	Variable	0:LastTransition	Localized-Text	FiniteTransition-VariableType	Mandatory
HasComponent	Method	Cancel			Mandatory
HasComponent	Method	Deselect			Mandatory
HasComponent	Method	SelectProgram			Mandatory
HasComponent	Method	SelectProgram-ByNodeId			Optional
HasComponent	Method	Start			Mandatory
HasComponent	Method	Stop			Mandatory
HasComponent	Method	SelectBlockNumber			Mandatory

NCTransitionEventType
An event reported when the state of a 4.2 "NCStateMachineType" instance changes.

Error
The state when an error has occurred during execution of an NC program.

Finished
The state when an NC program has finished execution.

Idle
The state when an NC program is selected and the execution can be started.

Interrupted
The state when a running NC program was stopped (e.g., by NC Stop or remotely by a *Stop* method call).

Running
The state when a selected NC program is being executed.

Stopped
The state when a program execution has stopped but has not yet been canceled. Many conditions can trigger a stop, for example, conclusion of a block in single block mode, completed execution of an M0 or stop block, completed execution of an M1 block if a conditional stop has been defined, a manual stop by the operator, or a stop caused by a machine manufacturer function inside the PLC or a macro.

NotSelected
The state when no NC program is selected.

ErrorToIdle
The state transition from *Error* to *Idle*. Causes of this transition can be a *Cancel* called from remote or an internal stop of the control.

ErrorToInterrupted
The state transition from *Error* to *Interrupted*. It occurs when all *Error* state-related errors are cleared on the machine. The program execution can be started afterwards again.

FinishedToIdle
The state transition from *Finished* to *Idle* (e.g., after the execution of an M30 block or END PGM).

IdleToNotSelected
The state transition from *Idle* to *NotSelected*. Cause of the transition is that the program has been deselected (e.g., using the *Deselect* method or manually on the control).

IdleToRunning
The state transition from *Idle* to *Running*. It occurs when first starting a program.

InterruptedToIdle
The state transition from *Interrupted* to *Idle*. It occurs on *SelectProgram*, *Cancel* in *Single Block* mode or after an internal stop of the control.

InterruptedToRunning
The state transition from *Interrupted* to *Running*. It occurs when the program is restarted after an interruption.

RunningToError
The state transition from *Running* to *Error*. Cause of the transition is an error occurring within execution of an NC program.

RunningToFinished
The state transition from *Running* to *Finished*. It occurs on program completion.

RunningToInterrupted
The state transition from *Running* to *Interrupted*. It occurs after *Stop* invocation in *Automatic* mode or an NC stop.

RunningToStopped
The state transition from *Running* to *Stopped*. It occurs after block execution in single block mode or an M0 block for example, see *Stopped* state.

StoppedToIdle
The state transition from *Stopped* to *Idle*. It occurs on program *Select* or *Cancel* in single block mode or after an internal stop of the control.

NotSelectedToIdle
The state transition from *NotSelected* to *Idle*. It occurs when an NC program was selected using the *SelectProgram* method, manually at the machine or via other remote control possibilities.

StoppedToRunning
The state transition from *Stopped* to *Running*. It occurs on program resume using *Start* in *Single Block* mode remotely or manually at the machine.

StoppedToError
The state transition from *Stopped* to *Error*. Cause of the transition is a new error while the program is in *Stopped* state.

IdleToIdle
The state transition from *Idle* to *Idle*. This transition occurs when a program is selected in the Idle state.

InterruptedToError
The state transition from *Interrupted* to *Error*. If an error occurred while the program is in the *Interrupted* state.

FinishedToError
The state transition from *Finished* to *Error*.

0:LastTransition
The *ModellingRule* has been changed to *Mandatory*.

Cancel
Cancels the execution of an erroneous, stopped, interrupted or finished NC program and sets the program execution pointer to the first block.

Table 54: Cancel Attributes

Attribute	Value
BrowseName	Cancel

Table 55: Cancel Result Codes

Result Code	Description
BadInvalidState	Returned when the program execution state does not allow canceling the program execution (e.g., if a program is running).
BadOutOfService	Returned when the server has lost its internal connection to the control.

Deselect

Deselects the selected NC program.

Table 56: Deselect Attributes

Attribute	Value
BrowseName	Deselect

Table 57: Deselect Result Codes

Result Code	Description
BadInvalidState	Returned when the program execution state does not allow deselecting the program (e.g., if a program is running).
BadUserAccessDenied	Returned when the client user has insufficient rights on the control.
BadOutOfService	Returned when the server has lost its internal connection to the control.

SelectProgram

Select an NC program for execution. The given program name will be loaded from disk for execution by the control.

Table 58: SelectProgram Attributes

Attribute	Value
BrowseName	SelectProgram

Signature

```
SelectProgram{
    [in] String ProgramName
};
```

Table 59: SelectProgram Signature

Argument	Description
ProgramName	String containing the full path name of the NC program to be selected

Table 60: SelectProgram Result Codes

Result Code	Description
BadInvalidState	Returned when the operating mode or the program execution state does not allow selecting a program (e.g., if a program is running or the operating mode is *Manual*).
BadFileNotFound	Returned when the file to be loaded could not be found.
BadInvalidArgument	Returned when wrong arguments were provided. Note that the maximum supported program path length on controls is 255 charac-ters. Further information: Setup, Testing and Running NC Programs User's Manual, or Setup and Program Run User's Manual
BadNoDataAvailable	Returned when the file exists but is not accessible in general.
BadUserAccessDenied	Returned when the client user has insufficient rights on the control.
BadOutOfService	Returned when the server has lost its internal connection to the control.

SelectProgramByNodeId

Select an NC program using the *NodeId* of the corresponding *File* object node. The program will be loaded from disk for execution by the control.

See also 9.4 "Warnings and Important Hints" regarding access to the machine's file system within 9 "Machine File System Access".

Table 61: SelectProgramByNodeId Attributes

Attribute	Value
BrowseName	SelectProgramByNodeId

Signature

```
SelectProgramByNodeId{
    [in] NodeId FileNodeId
};
```

Table 62: SelectProgramByNodeId Signature

Argument	Description
FileNodeId	NodeId of the file object node representing the NC program to be selected

Table 63: SelectProgramByNodeId Result Codes

Result Code	Description
BadInvalidState	Returned when the operating mode or the program execution state does not allow selecting a program (e.g., if a program is running or the operating mode is *Manual*).
BadNodeIdUnknown	Returned when the given NodeId refers to a node that does not exist.
BadFileNotFound	Returned when the file to be loaded could not be found.
BadInvalidArgument	Returned when wrong arguments were provided. Note that the maximum supported program path length on controls is 255 charac-ters. Further information: Setup, Testing and Running NC Programs User's Manual, or Setup and Program Run User's Manual
BadNoDataAvailable	Returned when the file exists but is not accessible in general.
BadUserAccessDenied	Returned when the client user has insufficient rights on the control.
BadOutOfService	Returned when the server has lost its internal connection to the control.

Start

Start the selected NC program. The support of this method is machine-specific.

An NC start can only be executed if the machine-specific PLC program of the machine manufacturer supports an external start of NC programs (ApiChn.NN_ChnNcStartExternRequest). A successful start of the NC program results in a state change of the state machine.

Table 64: Start Attributes

Attribute	Value
BrowseName	Start

Table 65: Start Result Codes

Result Code	Description
BadInvalidState	Returned when the operating mode or the program execution state does not allow starting a program (e.g., if no program is selected or the operating mode is *Manual*).
BadNoDataAvailable	Returned when the file exists but is not accessible in general.
BadUserAccessDenied	Returned when the client user has insufficient rights on the control. (E.g. the right NC.RemoteProgramRun is needed to start an NC program.)

Result Code	Description
BadOutOfService	Returned when the server has lost its internal connection to the control.

Stop

Stop execution of the active NC program.

Table 66: Stop Attributes

Attribute	Value
BrowseName	Stop

Table 67: Stop Result Codes

Result Code	Description
BadOutOfService	Returned when the server has lost its internal connection to the control.

SelectBlockNumber

Set the NC program execution pointer to the selected block number.

Table 68: SelectBlockNumber Attributes

Attribute	Value
BrowseName	SelectBlockNumber

Signature

```
    SelectBlockNumber{
        [in] UInt32 BlockNumber
    };
```

Table 69: SelectBlockNumber Signature

Argument	Description
BlockNumber	The block number of the NC program to be selected

Table 70: SelectBlockNumber Result Codes

Result Code	Description
BadInvalidArgument	Returned when no valid argument was provided, no program loaded, or the argument was not a number.
BadInvalidState	Returned when the operating mode or the program execution state does not allow selecting a block (e.g., if a program is running or the operating mode is *Manual*).
BadUserAccessDenied	Returned when the client user has insufficient rights on the control.
BadOutOfService	Returned when the server has lost its internal connection to the control.

4.16 ToolDataManagementType

Overview

The *ToolDataManagementType* gathers the tool data management functionalities of a machine.
A *ToolDataManagementType* instance provides read and write access to the tool data, events, and information about the tool data items and tool types.

An overview is given at 3.10 "Tool Data Management".

Note that tool data access requires an active and fully initialized NC software. *ToolDataRepresentation* components are instantiated when the *Machine* reaches the *State NCIsAvailable*. The functionality of *LocalDataSet*, *Validation*, and *ToolDataAccess* methods is available only during *State NCIsAvailable*.

Attributes

Table 71: ToolDataManagementType Definition Attributes

Attribute	Value
BrowseName	ToolDataManagementType
IsAbstract	false

References

ToolDataManagementType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 72: ToolDataManagementType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	Notifications		BaseObjectType	Mandatory
HasComponent	Object	LocalDataSet		LocalToolDataSet-Type	Mandatory
HasComponent	Object	ToolDataAccess		BaseObjectType	Mandatory
HasComponent	Object	ToolDataRepresen-tation		ToolDataRepresen-tationType	Mandatory
HasComponent	Object	Validation		BaseObjectType	Optional

Additional Subcomponents

Some components of *ToolDataManagementType* have additional components, which are defined in Table 73.

Table 73: ToolDataManagementType Additional Subcomponents

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
ToolData-Access	HasComponent	Method	CreateNew-ToolRecord			Mandatory
ToolData-Access	HasComponent	Method	CreateNew-ToolRecord-WithId			Mandatory
ToolData-Access	HasComponent	Method	DeleteTool-Record			Mandatory
ToolData-Access	HasComponent	Method	FindTool-Record-Identifiers-ByName			Mandatory
ToolData-Access	HasComponent	Method	GetAllTool-Numbers			Optional
ToolData-Access	HasComponent	Method	GetAllTool-Numbers-AssignedTo-Pockets			Mandatory
ToolData-Access	HasComponent	Method	GetAssigned-PocketNumbers			Mandatory
ToolData-Access	HasComponent	Method	GetToolData			Mandatory
ToolData-Access	HasComponent	Method	GetToolData-ByCategory			Mandatory
ToolData-Access	HasComponent	Method	GetToolData-Items			Mandatory
ToolData-Access	HasComponent	Method	GetToolIndices			Mandatory
ToolData-Access	HasComponent	Method	GetToolNum-ber			Mandatory
ToolData-Access	HasComponent	Method	GetToolType			Mandatory
ToolData-Access	HasComponent	Method	UpdateTool-DataItems			Mandatory
ToolData-Access	HasComponent	Object	Argument-Descriptions		BaseObjectType	Optional
ToolData-Access▶Argument-Descriptions	HasComponent	Variable	ToolNumber	UInt32	BaseAnalogType	Optional
ToolData-Access▶Argument-Descriptions	HasComponent	Variable	ToolIndex	Byte	BaseAnalogType	Optional
Validation	HasComponent	Object	3DModels		BaseObjectType	Optional

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Validation▶ 3DModels	HasComponent	Method	Validate3D-ModelFile			Mandatory
Validation▶ 3DModels	HasComponent	Object	Argument-Descriptions		BaseObjectType	Optional
Validation▶ 3DModels▶ Argument-Descriptions	HasComponent	Variable	DataItem	Qualified-Name	SelectionListType	Optional
Validation▶ 3DModels▶ Argument-Descriptions	HasComponent	Variable	Issues	UInt32[]	MultiStateValue-DiscreteType	Optional
Validation▶ 3DModels	HasComponent	Object	RelatedTool-DataItems		FolderType	Mandatory
Validation▶ 3DModels▶ RelatedTool-DataItems	Organizes	Object	<ToolDataItem>		ToolDataItem-Type	OptionalPlace-holder

Additional References

Some components of *ToolDataManagementType* have additional references, which are defined in Table 74.

Table 74: ToolDataManagementType Additional References

Source Path	ReferenceType	IsForward	Target Path
Notifications	GeneratesEvent	True	ToolLockedEventType
LocalDataSet	GeneratesEvent	True	ToolDataSetModificationEventType

Notifications

Notifications is the source node of tool data related events like the 5.9 "ToolLockedEventType".

LocalDataSet

The *LocalDataSet* component provides the functionality to synchronize a local copy of the machine's tool data (see 4.22 "LocalToolDataSetType"). It is the source node of events of type 5.7 "ToolDataSetModificationEventType".

ToolDataAccess

The *ToolDataAccess* component is used to group methods to access the machine's tool data. The methods allow operations like the creation of tool records, reading tool data and updating the data.

Some of the methods use variables for additional argument descriptions. The *ArgumentDescriptions* object references the variables shared by multiple methods. If the variable describes an optional argument of a method, the variable's *Value* contains the default value.

A client user can only edit tool data if he has the corresponding NC.EditToolTable right.

ToolDataAccess CreateNewToolRecord

Creates a new tool record for a tool of the given tool type. The data items are initialized with the default values.

The tool type is specified using the *BrowseName* (data type *QualifiedName*) of a *ToolType* listed at the ToolDataRepresentation component.

Depending on the control type and software version, different types of tools are available. ("Table 216: HEIDENHAIN Tool Types" in "Annex A: Tool Data Reference" lists the unique string identifiers of the tool types.)

The arguments *ToolNumber* and *ToolIndex* are optional. If the *ToolNumber* is not provided or 0, the record is "appended at the end" of the existing tool records using the next number. If the *ToolIndex* is not provided, the record with index 0 is created.

Table 75: CreateNewToolRecord Attributes

Attribute	Value
BrowseName	CreateNewToolRecord

Signature

```
    CreateNewToolRecord{
        [in]  QualifiedName   ToolType,
        [in]  UInt32          ToolNumber,
        [in]  Byte            ToolIndex,
        [out] UInt32          NewToolNumber,
        [out] Byte            NewToolIndex
    };
```

Table 76: CreateNewToolRecord Signature

Argument	Description
ToolType	The type of tool to create; the *BrowseName* of a *ToolTypeDescriptionType* instance at the *ToolDataRepresentation* component.
ToolNumber	If given, create the new tool record for this tool number; otherwise generate it
ToolIndex	If given, create the new tool record with this index. If an index is given, the *ToolNumber* is also required
NewToolNumber	Tool number of the newly created tool record
NewToolIndex	Tool index of the newly created tool record

Along with the *InputArguments* and *OutputArguments* properties, the *CreateNewToolRecord* method has the references listed in Table 77.

Table 77: CreateNewToolRecord Additional References

ReferenceType	IsForward	Target Path
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolIndex

Table 78: CreateNewToolRecord Result Codes

Result Code	Description
BadInvalidState	The tool record cannot be created at the moment. Possible reasons are: ▪ The control is not active and fully initialized. Tool data access is possible during *Machine State NCIsAvailable*. ▪ At least one of the internal tool tables is locked for editing by the operator at the machine or another external system.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadEntryExists	The tool number already exists.
BadOutOfService	Returned when the server has lost its internal connection to the control.

Result Code	Description
BadUserAccessDenied	Returned when the client user has insufficient rights on the control. (E.g. the NC.Edit-ToolTable right is needed to edit tool data.)
BadInternalError	An error occurred during execution of internal requests to create the related tool records.
BadUnexpectedError	Some other error occurred.

Background information:
Depending on the tool type, the data is stored internally in more than one tool data table. The method automatically creates the records in the relevant tables. For touch probes, the record containing the additional data is linked via the *TouchprobeRecord* data item.

For more information about the tool data items and their mapping to the internal tables, see "Annex A: Tool Data Reference".

ToolDataAccess CreateNewToolRecordWithId

Creates a new tool record for a tool of the given tool type similar to CreateNewToolRecord.

The method is intended for use with unique tool identifiers (database ID) of a central tool management system.

The *ToolNumber* is optional. A new record with index 0 for the main tool is created and the *DatabaseId* data item initialized with the given *ToolDatabaseId*.

Additional indices can be added using the *CreateNewToolRecord* method.

See also "Terms", Page 24.

Table 79: CreateNewToolRecordWithId Attributes

Attribute	Value
BrowseName	CreateNewToolRecordWithId

Signature

```
    CreateNewToolRecordWithId{
        [in]  QualifiedName   ToolType,
        [in]  String          ToolDatabaseId,
        [in]  UInt32          ToolNumber,
        [out] UInt32          NewToolNumber,
        [out] Byte            NewToolIndex
    };
```

Table 80: CreateNewToolRecordWithId Signature

Argument	Description
ToolType	Name of the tool type to create; the *BrowseName* of a *ToolTypeDescriptionType* instance at the *ToolDataRepresentation* component.
ToolDatabaseId	The identifier from the central tool management system
ToolNumber	If given, create the new tool record for this tool number; otherwise generate it
NewToolNumber	Tool number of the newly created tool record
NewToolIndex	Tool index of the newly created tool record

Along with the *InputArguments* and *OutputArguments* properties, the *CreateNewToolRecordWithId* method has the references listed in Table 81.

Table 81: CreateNewToolRecordWithId Additional References

ReferenceType	IsForward	Target Path
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber

Table 82: CreateNewToolRecordWithId Result Codes

Result Code	Description
BadInvalidState	The tool record cannot be created at the moment. Possible reasons are: ■ The control is not active and fully initialized. Tool data access is possible during *Machine State NCIsAvailable*. ■ At least one of the internal tool tables is locked for editing by the operator at the machine or another external system.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadOutOfService	Returned when the server has lost its internal connection to the control.

Result Code	Description
BadUserAccessDenied	Returned when the client user has insufficient rights on the control. (E.g. the NC.Edit-ToolTable right is needed to edit tool data.)
BadInternalError	An error occurred during execution of internal requests to create the related tool records.
BadUnexpectedError	Some other error occurred.

ToolDataAccess DeleteToolRecord

Deletes the specified tool record.

If a tool is listed in a magazine, the record of the main tool (the record with index 0) is protected from deletion. If a tool is in use, all of its records are protected from deletion.

Table 83: DeleteToolRecord Attributes

Attribute	Value
BrowseName	DeleteToolRecord

Signature

```
DeleteToolRecord{
    [in] UInt32    ToolNumber,
    [in] Byte      ToolIndex
};
```

Table 84: DeleteToolRecord Signature

Argument	Description
ToolNumber	The number of the tool record to delete
ToolIndex	The index of the record to delete; if omitted, the record with index 0 is deleted.

Along with the *InputArguments* and *OutputArguments* properties, the *DeleteToolRecord* method has the references listed in Table 85.

Table 85: DeleteToolRecord Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolIndex

Table 86: DeleteToolRecord Result Codes

Result Code	Description
BadInvalidState	The tool record cannot be deleted at the moment. Possible reasons are: ■ The control is not active and fully initialized. Tool data access is possible during *Machine State NCIsAvailable*. ■ The tool record is locked for editing by the operator at the machine or another external system.
BadNoEntryExists	The addressed tool record does not exist.
BadNotSupported	The operation is not supported by the server.
BadNoDeleteRights	The tool is listed in a magazine or belongs to a tool in use. Deleting the tool record is not permitted.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadUserAccessDenied	Returned when the client user has insufficient rights on the control. (E.g. the NC.Edit-ToolTable right is needed to edit tool data.)
BadInternalError	An error occurred during execution of internal requests to delete the related tool records.
BadUnexpectedError	Some other error occurred.

Background information:
DeleteToolRecord deletes the tool record from all relevant internal tool data tables. If it is a tool of type touch probe, the record in the touch probe table is also deleted—if no other tool record refers to it.

ToolDataAccess FindToolRecordIdentifiersByName

Returns a list of *ToolRecordIdentifiers* where the value of the *Name* data item is equal to the given *ToolName*.

Table 87: FindToolRecordIdentifiersByName Attributes

Attribute	Value
BrowseName	FindToolRecordIdentifiersByName

Signature

```
    FindToolRecordIdentifiersByName{
        [in]  String                      ToolName,
        [out] ToolRecordIdentifierDataType[] ToolRecordIdentifiers
    };
```

Table 88: FindToolRecordIdentifiersByName Signature

Argument	Description
ToolName	The value to search at the *Name ToolDataItem* of tool records
ToolRecordIdentifiers	Identifiers of all tool records matching the given *ToolName*

Table 89: FindToolRecordIdentifiersByName Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadNotFound	No tool record with the given tool *Name* has been found.
BadInternalError	An error occurred during execution of an internal request while searching for the tool records.
BadUnexpectedError	Some other error occurred.
BadOutOfService	Returned when the server has lost its internal connection to the control.

ToolDataAccess GetAllToolNumbers

Returns a list of tool numbers that are present in the machine's tool memory (the tool table, for example, tool.t).

If there are several records for one tool with different indices, the tool number is in the list only once.

The method was added with version 1.04. It is supported by the OPC UA NC Server starting with NC software version 17 SP01, see "Associated HEIDENHAIN CNC Controls", Page 15.

Table 90: GetAllToolNumbers Attributes

Attribute	Value
BrowseName	GetAllToolNumbers

Signature

```
    GetAllToolNumbers{
        [out]  UInt32[]     ToolNumbers
    };
```

Table 91: GetAllToolNumbers Signature

Argument	Description
ToolNumbers	List of all known tool numbers

Table 92: GetAllToolNumbers Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadNotFound	There are no tools listed in the tool memory.
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request while reading the tool numbers.
BadUnexpectedError	Some other error occurred.
Uncertain	Some of the tool records have non-conforming record identifiers. The provided list might not be complete.

ToolDataAccess GetAllToolNumbersAssignedToPockets

Returns a list of tool numbers that are listed in the machine's magazines (the pocket table).

The list is not ordered and does not contain tool number duplicates, meaning that if a tool is listed twice in the magazines, the tool number is in the list only once.

Note that if a tool's number is listed in a magazine, the physical tool is not necessarily in the machine! The pocket may only be reserved for the tool. The management of tool magazines is specific to the machine manufacturer.

Table 93: GetAllToolNumbersAssignedToPockets Attributes

Attribute	Value
BrowseName	GetAllToolNumbersAssignedToPockets

Signature

```
    GetAllToolNumbersAssignedToPockets{
        [out]  UInt32[]     ToolNumbers
    };
```

Table 94: GetAllToolNumbersAssignedToPockets Signature

Argument	Description
ToolNumbers	Unordered list of tool numbers without duplicates

Table 95: GetAllToolNumbersAssignedToPockets Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadNotFound	There are no tools listed in the magazines.
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request while reading the tool numbers.
BadUnexpectedError	Some other error occurred.

ToolDataAccess GetAssignedPocketNumbers

Returns a list of magazine pocket identifiers that are assigned to the tool number.

Note that if a tool's number is assigned to a pocket, the physical tool is not necessarily in this pocket! The pocket may only be reserved for the tool (e.g., before the tool is loaded into the machine). The management of tool magazines is specific to the machine manufacturer.

Table 96: GetAssignedPocketNumbers Attributes

Attribute	Value
BrowseName	GetAssignedPocketNumbers

Signature

```
    GetAssignedPocketNumbers{
        [in]  UInt32                              ToolNumber
        [out] MagazinePocketIdentifierDataType[]  ToolPockets
    };
```

Table 97: GetAssignedPocketNumbers Signature

Argument	Description
ToolNumbers	The tool number to find in the magazines
ToolPockets	List of magazine and pocket numbers occupied by the given *ToolNumber*

Along with the *InputArguments* and *OutputArguments* properties, the *GetAssignedPocketNumbers* method has the references listed in Table 98.

Table 98: GetAssignedPocketNumbers Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber

Table 99: GetAssignedPocketNumbers Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadNotFound	The tool number has not been found in the magazines (the pocket table).
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request while searching the tool number.
BadUnexpectedError	Some other error occurred.

ToolDataAccess GetToolData

Read all type-specific data of the tool.

The method provides all tool data items that are associated with the type of the tool as a list of *0:KeyValuePairs*. The keys are equal to the *BrowseNames* of *ToolDataItems* at the ToolDataRepresentation component.

For a list of data item identifiers, see also "Annex A: Tool Data Reference".

Table 100: GetToolData Attributes

Attribute	Value
BrowseName	GetToolData

Signature

```
    GetToolData{
        [in]  UInt32            ToolNumber,
        [in]  Byte              ToolIndex,
        [out] KeyValuePair[]    ToolData
    };
```

Table 101: GetToolData Signature

Argument	Description
ToolNumber	The number of the tool
ToolIndex	The index of the tool in case of an indexed tool; if omitted, the record with index 0 is chosen.
ToolData	Array of *ToolDataItem QualifiedNames* and their corresponding values in the request-ed record

Along with the *InputArguments* and *OutputArguments* properties, the *GetToolData* method has the references listed in Table 102.

Table 102: GetToolData Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolIndex

Table 103: GetToolData Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadNoEntryExists	There is no tool data record with the given *ToolNumber* and *ToolIndex*.
BadNotSupported	The server does not support this functionality for this tool type (e.g., because the type of the tool is unknown, deprecated or not well-defined).
BadDeviceFailure	An inconsistent tool data record prevents the execution.
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request while reading the tool data.
BadUnexpectedError	Some other error occurred.

ToolDataAccess GetToolDataByCategory

Returns the tool's data that belongs to the given data category.

Compared to the similar GetToolData method, *GetToolDataByCategory* provides only the data that belongs to the given *DataCategory*. The available data categories are listed at the ToolDataRepresentation component. The method returns only data items that are associated with the type of the tool.

Table 104: GetToolDataByCategory Attributes

Attribute	Value
BrowseName	GetToolDataByCategory

Signature

```
    GetToolDataByCategory{
        [in]  QualifiedName     DataCategory,
        [in]  UInt32            ToolNumber,
        [in]  Byte              ToolIndex,
        [out] KeyValuePair[]    ToolData
    };
```

Table 105: GetToolDataByCategory Signature

Argument	Description
DataCategory	Filter for this data category The *BrowseName* of a 4.18 "ToolDataCategoryType" instance at *ToolDataRepresentation* component.
ToolNumber	The number of the tool
ToolIndex	The index of the tool in case of an indexed tool; if omitted, the record with index 0 is chosen.
ToolData	The resulting set of tool data

Along with the *InputArguments* and *OutputArguments* properties, the *GetToolDataByCategory* method has the references listed in Table 106.

Table 106: GetToolDataByCategory Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolIndex

Table 107: GetToolDataByCategory Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadNotFound	There are no data items in the specified *DataCategory* and tool type.
BadNotSupported	The server does not support this functionality for the tool type (e.g., because the type of the tool is unknown, deprecated or not well-defined).
BadDeviceFailure	An inconsistent tool data record prevents the execution.
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request while reading the tool data.
BadUnexpectedError	Some other error occurred.

ToolDataAccess GetToolDataItems

Read a list of tool data item values.

The *GetToolDataItems* method returns the values of a specified list of tool data items.

Compared to the GetToolData method, *GetToolDataItems* is not restricted to the data items that are associated with the tool type. If a data value exists in the internal tables for the requested data item and record, it is provided.

For a list of data item identifiers, see also "Annex A: Tool Data Reference".

Table 108: GetToolDataItems Attributes

Attribute	Value
BrowseName	GetToolDataItems

Signature

```
    GetToolDataItems{
        [in]  QualifiedName[]   DataItems,
        [in]  UInt32            ToolNumber,
        [in]  Byte              ToolIndex,
        [out] StatusCode[]      Status,
        [out] KeyValuePair[]    ToolData
    };
```

Table 109: GetToolDataItems Signature

Argument	Description
DataItems	Retrieve these data items for the requested tool record if they exist
ToolNumber	The number of the tool
ToolIndex	The index of the tool in case of an indexed tool; if omitted, the record with index 0 is chosen.
Status	Retrieval result per requested data item in the same order as the *DataItems* argument contents. The argument provides additional information in case of an *Uncertain* method status code. ■ *Good*: The value is provided ■ *GoodNoData*: The value is provided but empty (*Null*) ■ *BadNotFound*: There is no data item with that identifier ■ *BadNoEntryExists*: The data item identifier is valid. But there is no value in the internal tables because it belongs to other tool types only. ■ *BadConfigurationError*: The record is internally not well-defined (e.g., a touch probe data record is incomplete). The link to the requested additional touch probe data item is missing.
ToolData	The resulting values for successfully requested data items. *DataItems* with a bad status code in *Status* are not provided.

Along with to the *InputArguments* and *OutputArguments* properties, the *GetToolDataItems* method has the references listed in Table 110.

Table 110: GetToolDataItems Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolIndex

Table 111: GetToolDataItems Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadDeviceFailure	The internal data record is inconsistent (e.g., for tool that needs records in two internal tables and the second record is missing).
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request to access the data of the tool.
BadUnexpectedError	Some other error occurred.
Uncertain	Not all of the requested *DataItems* could be read successfully. See *Status* output argument for more information.

ToolDataAccess GetToolIndices

Get all index numbers belonging to the tool with the given tool number.

The method is intended for use with indexed tools.

Table 112: GetToolIndices Attributes

Attribute	Value
BrowseName	GetToolIndices

Signature

```
GetToolIndices{
    [in]  UInt32        ToolNumber,
    [out] Byte[]        ToolIndices
};
```

Table 113: GetToolIndices Signature

Argument	Description
ToolNumber	The number of the tool
ToolIndices	List of all known indices for the requested tool

Along with to the *InputArguments* and *OutputArguments* properties, the *GetToolIndices* method has the references listed in Table 114.

Table 114: GetToolIndices Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber

Table 115: GetToolIndices Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadInvalidArgument	The input argument was invalid; more information is given at the *inputArgumentRe-sults*.
BadNoEntryExists	There is no tool record with the given tool number.
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request while searching for the tool records.
BadUnexpectedError	Some other error occurred.

ToolDataAccess GetToolNumber

Get the machine-specific tool number of a tool selected by the database ID and all index numbers of the records.

The method is intended for use with unique tool identifiers of a central tool management system (database ID).

If the database ID of a tool is set at the main tool record (index 0) as recommended by HEIDENHAIN, the method *GetToolNumber* returns the machine-specific tool number and all indices. The record identifiers can be used to access the tool's data.

If the database ID is set at a specific additional index, the method provides the tool number and the dedicated index.

See also "Terms", Page 24.

Table 116: GetToolNumber Attributes

Attribute	Value
BrowseName	GetToolNumber

Signature

```
GetToolNumber{
     [in]  String        ToolDatabaseId,
     [out] UInt32        ToolNumber,
     [out] Byte[]        ToolIndices
};
```

Table 117: GetToolNumber Signature

Argument	Description
ToolDatabaseId	The identifier from the central tool management system
ToolNumber	The number of the tool with the given database identifier
ToolIndices	All indices for the tool with the given database identifier

Table 118: GetToolNumber Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadInvalidArgument	The input argument was invalid; more information is given at the *inputArgumentRe-sults*.
BadNoEntryExists	No tool record was found for the given database ID.
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request while searching for the tool records.
BadUnexpectedError	Some other error occurred.

ToolDataAccess GetToolType

Returns the type of the given tool from the tool record.

The *ToolType* is provided as *QualifiedName* equal to the *BrowseName* of the 4.21 "ToolTypeDescriptionType" instance of the tool type at the ToolDataRepresentation.

For a list of tool type identifiers, see also "Annex A: Tool Data Reference".

Table 119: GetToolType Attributes

Attribute	Value
BrowseName	GetToolType

Signature

```
    GetToolType{
         [in]  UInt32           ToolNumber,
         [in]  Byte             ToolIndex,
         [out] QualifiedName    ToolType
    };
```

Table 120: GetToolType Signature

Argument	Description
ToolNumber	The number to identify the tool record
ToolIndex	The index to identify the tool record; if omitted, the record with index 0 is chosen.
ToolType	The *BrowseName* of the requested tool's *ToolTypeDescription*

Along with to the *InputArguments* and *OutputArguments* properties, the *GetToolType* method has the references listed in Table 121.

Table 121: GetToolType Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolIndex

Table 122: GetToolType Result Codes

Result Code	Description
BadInvalidState	The control is not active and fully initialized and access to tool data is not possible at the moment. Tool data access is possible during *Machine State NCIsAvailable*.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadNoEntryExists	A tool record with the given number and index does not exist.
BadNotSupported	The type of the tool is not known (e.g., because it is deprecated, not supported by the control model, or not well defined, for example, the internal subtype information is missing).
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadInternalError	An error occurred during execution of an internal request to get the tool type.
BadUnexpectedError	Some other error occurred.

ToolDataAccess UpdateToolDataItems

Update the data of a tool record.

The specified data items are updated with the new values; other data items remain unchanged.

The method returns a *Good StatusCode* if all update operations have been executed successfully. If some of the new values cannot be written (e.g., because a value was invalid or the record has been locked for editing by the operator), none of the data items are updated.

The new data is validated at the server before the update operations are executed. If some of the data items are not known, the new value is invalid (e.g., out of range of this data item or does not match the data type), the method returns an *Uncertain StatusCode*. In this case the output argument *ValidationResult* contains the result of the validation using a specific *StatusCode* per data item to be updated.

The *ToolNumber*, *ToolIndex*, or *Type* of a tool cannot be changed using *UpdateToolDataItems*.

For a list of data item identifiers, see also "Annex A: Tool Data Reference".

Table 123: UpdateToolDataItems Attributes

Attribute	Value
BrowseName	UpdateToolDataItems

Signature

```
    UpdateToolDataItems{
        [in]  KeyValuePair[]     ToolData,
        [in]  UInt32             ToolNumber,
        [in]  Byte               ToolIndex,
        [out] StatusCode[]       ValidationResult
    };
```

Table 124: UpdateToolDataItems Signature

Argument	Description
ToolData	The *ToolDataItem* names and values to be set. *Value* type must match and the data item must exist for the tool type
ToolNumber	The number to identify the tool record to be updated
ToolIndex	The index to identify the tool record to be updated; if omitted, the record with index 0 is updated.
ValidationResult	Information per data item in case of failure For each *KeyValuePair* given in *ToolData*, a validation result is provided using one of the following StatusCodes: ■ *Good*: this element is not the reason of an error ■ *BadOutOfRange*: the new value is out of the value range of this data item ■ *BadTypeMismatch*: the *Value's DataType* does not match the data type of the data item ■ *BadNotWritable*: the data item is not writable in general (e.g., the tool number or the type of the tool) ■ *BadNotFound*: the data item is not known ■ *BadNoEntryExists*: the data item does not exist for tools of this type ■ *BadInvalidArgument*: the new value is invalid (e.g., does not match the required string pattern) ■ *BadEntryExists*: the value of the data item has to be unique across all records (e.g., the database ID of a tool), but the value already exists at another record ■ *BadConfigurationError*: The record is internally not well-defined (e.g., a touch probe data record is incomplete). The link to the requested additional touch probe data item is missing.

Argument	Description
	■ *BadUnexpectedError*: something else was not correct (some other error occurred) ■ *Uncertain*: the value of this data item or a combination of this and other data item values might not be valid

Along with the *InputArguments* and *OutputArguments* properties, the *UpdateToolDataItems* method has the references listed in Table 125.

Table 125: UpdateToolDataItems Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolNumber
HasOptionalInputArgumentDescription	True	ToolDataAccess ▶ ArgumentDescriptions ▶ ToolIndex

Table 126: UpdateToolDataItems Result Codes

Result Code	Description
BadInvalidState	The tool data cannot be updated at the moment. Possible reasons are: ■ The control is not active and fully initialized. Tool data access is possible during *Machine State NCIsAvailable*. ■ The tool record is locked for editing by the operator at the machine or another external system.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*.
BadReferenceNotAllowed	One of the data items to write has referencing semantics, and the value to write is invalid (e.g., the new replacement tool value would refer to a non-existing tool record).
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadDeviceFailure	The method could not be executed due to an inconsistency of the tool record at the control.
BadUserAccessDenied	Returned when the client user has insufficient rights on the control. (E.g. the NC.Edit-ToolTable right is needed to edit tool data.)
BadInternalError	An error occurred during execution of an internal request to update the tool data.
BadUnexpectedError	Some other error occurred.
Uncertain	The method could not be executed successfully due to invalid data items and/or values to update. No tool data has been updated. Additional information for each data item and value is provided by the output argument *ValidationResult*.

ToolDataRepresentation

The *ToolDataRepresentation* component of type 4.17 "ToolDataRepresentationType" provides information about the machine's tool data definition and available tool types.

The *BrowseName* of listed tool types can be used at the *CreateNewToolRecord* or *CreateNewToolRecordWithId* methods to create a tool record for a tool of this type. The *GetToolType* method returns the *BrowseName* of the tool type instance for an existing tool record.

The *BrowseName* of a tool data category can be used at the *GetToolDataByCategory* method to read all data of a tool record that belongs to the given category and tool type.

The *ToolDataItems* component of *ToolDataRepresentation* describes all tool data items of the machine. The *BrowseNames* of the instances are used as *Keys* of the *KeyValuePair* arguments at various *ToolDataAccess* methods. Examples are the *ToolData* arguments of the *GetToolData* and *UpdateToolData* methods.

For a list of data item identifiers and tool types, see "Annex A: Tool Data Reference".

More details are given at the 4.17 "ToolDataRepresentationType" and 4.19 "ToolDataItemType" .

Validation

Validation provides functionality to check tool-related data (e.g., to validate 3D model files for tools).

See 3D Model Files for Tools in 3.10 "Tool Data Management" for an overview including an example.

Validation 3DModels

The availability and use of 3D models of tools and tool carriers for collision monitoring during program run and material removal simulation depend on the software option Dynamic Collision Monitoring (DCM) or DCM version 2. The functionality of *Validation 3DModels* does not depend on the software option for DCM.

The *3DModels* component of *Validation* provides information about tool data items that denote 3D models and a method to check 3D model files for tool data.

3D models for tools or tool carriers need to fulfill requirements of the NC software (e.g., the model should have a closed shell or the file needs to be stored at a specific location). Otherwise the model cannot be used by the control. Validation of 3D model files before use at the machine during production processes can help to prevent interruptions (e.g., to avoid errors at TOOL CALLs).

Note that not all requirements for 3D model files for tools and tool carriers can be checked by the *Validate3DModelFile* method. The NC software (and the method) cannot determine the correct origin of the coordinate system and its orientation as it depends on the characteristics of the concrete tool.

Overview of 3D model file validation aspects for tool data items:

- Correct location of the files, see FileLocation and FileLocationManufacturer of 4.19 "ToolDataItemType"
- Compatible file name (e.g., maximum length and no special characters), see ValueDescription of 4.19 "ToolDataItemType"
- Supported 3D model file format per tool data item (e.g., M3D or STL files)
- Readability and completeness of the files (if it is a combined model)
- Size and complexity of the 3D model (e.g., maximum number of triangles for STL files)
- Quality of the 3D model (e.g., a closed shell ("watertight" mesh))

For more information about 3D models for tools and tool carriers, see Setup, Testing and Running NC Programs User’s Manual or Setup and Program Run User’s Manual of the respective control model.

The *ReleatedToolDataItems* folder *Organizes ToolDataItems* that denote 3D model files. Depending on the control model the list of *RelatedToolDataItems* can differ.

ArgumentDescriptions groups the variables for additional method argument descriptions, see Validation 3DModels Validate3DModelFile.

The components of *3DModels* (*ReleatedToolDataItems* and the method argument description variable values) are initialized when the *Machine* reaches the *State NCIsAvailable*.

Validation 3DModels Validate3DModelFile

Validate3DModelFile checks a given 3D model file at the *FileSystem* of the machine for whether it can be used for the designated use case (e.g., as 3D model for a tool carrier or a tool).

See "Validation 3DModels" for more information about the requirements of the control for 3D models and 3D Model Files for Tools in 3.10 "Tool Data Management" for an overview.

If the 3D model file passes the validation, the method returns *Good*.

If at least one issue has been found, the method returns *Uncertain*. In this case, the output argument *Issues* contains the corresponding number codes. The method *Validate3DModelFile HasArgumentDescription* variable *Issues*. *Issues* is a variable of type *MultiStateValueDiscreteType*. With its properties the variable provides more information about each possible number code. The *EnumValues* property at the instance of *ToolDataManagementType* in the address space of the OPC UA NC Server contains the name and a description of each possible validation issue.

For a list of possible issues of NC software version 18, see 3D Model Files for Tools: Validation Issues List in Annex A: Tool Data Reference.

Table 127: Validate3DModelFile Attributes

Attribute	Value
BrowseName	Validate3DModelFile

Signature

```
    Validate3DModelFile{
        [in]  QualifiedName      DataItem,
        [in]  NodeId             File,
        [out] UInt32[]           Issues
    };
```

Table 128: Validate3DModelFile Signature

Argument	Description
DataItem	Identifier of a *ToolDataItemType* instance; the file is validated for the use cases associated with this data item. The *Selections* property of the corresponding argument description variable contains the possible values.
File	*NodeId* of the 3D model file to validate
Issues	List of issues found during validation of the 3D model file. The *EnumValues* property of the argument description variable at the instance of the method provides information about the issues. If the 3D model file passes the validation, the list is empty. See 3D Model Files for Tools: Validation Issues List in Annex A: Tool Data Reference for possible issues.

Along with the *InputArguments* and *OutputArguments* properties, the *Validate3DModelFile* method has the references listed in Table 129.

Table 129: Validate3DModelFile Additional References

ReferenceType	IsForward	Target Path
HasArgumentDescription	True	Validation ▶ 3DModels ▶ ArgumentDescriptions ▶ DataItem
HasArgumentDescription	True	Validation ▶ 3DModels ▶ ArgumentDescriptions ▶ Issues

Table 130: Validate3DModelFile Result Codes

Result Code	Description
BadInvalidState	The validation cannot be executed at the moment. The control is not active and fully initialized. Validation of 3D model files for tool data is possible during *Machine State NCIsAvailable*.

Result Code	Description
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*. Result codes of input arguments at *inputArgumentResults* (except *Good*): DataItem ■ *BadInvalidArgument*: the given value is not one of the supported tool data items, meaning that it is not one of the entries listed at the *Selections* property File ■ *BadUserAccessDenied*: the user does not have access rights to the file denoted by the given *NodeId* ■ *BadNodeIdUnknown*: the given *NodeId* is not known ■ *BadNoMatch*: the node with this *NodeId* is not of type *FileType* ■ *BadNotFound*: there is no node for the *Null-NodeId*
BadOutOfService	Returned when the server has lost its internal connection to the control; validation is not possible.
Uncertain	The 3D model file did not pass the validation. At least one issue has been detected. The found issues are listed in the *Issues OutputArgument*.

4.17 ToolDataRepresentationType

Overview

A *ToolDataRepresentationType* instance describes the tool data definition, lists the tool types, and connects the different tool types with their related data items.

It is used at the 4.16 "ToolDataManagementType" to describe the machine-specific tool data and available tool types.

An *AssociatedWith* reference connects the concrete 4.21 "ToolTypeDescriptionType" instances with their related data items of type 4.19 "ToolDataItemType". The tool data items and tool types are grouped into categories.

Attributes

Table 131: ToolDataRepresentationType Definition Attributes

Attribute	Value
BrowseName	ToolDataRepresentationType
IsAbstract	false

References

ToolDataRepresentationType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 132: ToolDataRepresentationType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	ToolDataCategories		BaseObjectType	Mandatory
HasComponent	Object	ToolDataItems		BaseObjectType	Mandatory
HasComponent	Object	ToolTypeCategories		BaseObjectType	Mandatory
HasComponent	Object	ToolTypes		BaseObjectType	Mandatory

Additional Subcomponents

Some components of *ToolDataRepresentationType* have additional components, which are defined in Table 133.

Table 133: ToolDataRepresentationType Additional Subcomponents

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
ToolData-Categories	HasComponent	Object	Correction		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	Geometry		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	Identification		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	Manufacturer		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	Manufacturer-Extension		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	Technology		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	ToolLife		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	ToolWear		ToolTypeCatego-ryType	Optional
ToolData-Categories	HasComponent	Object	Touchprobe		ToolTypeCatego-ryType	Optional
ToolData-Items	HasComponent	Object	<ToolDataItem>		ToolDataItem-Type	OptionalPlace-holder
ToolType-Categories	HasComponent	Object	DressingTools		ToolTypeCatego-ryType	Optional
ToolType-Categories	HasComponent	Object	DrillingTools		ToolTypeCatego-ryType	Optional
ToolType-Categories	HasComponent	Object	GrindingTools		ToolTypeCatego-ryType	Optional
ToolType-Categories	HasComponent	Object	MillingTools		ToolTypeCatego-ryType	Optional
ToolType-Categories	HasComponent	Object	Touchprobes		ToolTypeCatego-ryType	Optional
ToolType-Categories	HasComponent	Object	TurningTools		ToolTypeCatego-ryType	Optional
ToolTypes	HasComponent	Object	<ToolType-Description>		ToolType-DescriptionType	OptionalPlace-holder

Additional References

Some components of *ToolDataRepresentationType* have additional references, which are defined in Table 134.

Table 134: ToolDataRepresentationType Additional References

Source Path	ReferenceType	IsForward	Target Path
ToolDataItems ▶ <ToolDataItem>	AssociatedWith	True	ToolTypes ▶ <ToolTypeDescription>

ToolDataCategories

ToolDataCategories provides a predefined list of 4.18 "ToolDataCategoryType" instances to group the 4.19 "ToolDataItemType" instances listed below *ToolDataItems* by area of application.

The *ManufacturerExtension* category references all machine-specific tool data items defined by the machine manufacturer. It is only instantiated in a server if the machine manufacturer has defined additional tool data items.

ToolDataItems

The *ToolDataItems* component lists all tool data items of the machine using the 4.19 "ToolDataItemType". The available tool data items may change between different machines, control models, and software versions.

Each *ToolDataItem* provides the description of a data item, the default value, and metadata like the value ranges, engineering units or the maximum string length.

Along with the data items defined by HEIDENHAIN, the additional machine-specific tool data items of the machine manufacturer are listed. For data items defined by HEIDENHAIN and more information about machine-specific tool data extension of the machine manufacturer, see "Annex A: Tool Data Reference".

Note that for some of the data items defined by HEIDENHAIN, the machine manufacturer can edit the definition, for example, change the default value or extend the maximum string length.

Each *ToolDataItem* is *AssociatedWith* one or more *ToolTypes*; see the following *ToolTypes* component description for more information.

ToolTypeCategories

The *ToolTypeCategories* component groups the tool types based on the manufacturing process using instances of type 4.20 "ToolTypeCategoryType". The server instantiates only the *ToolTypeCategories* that are supported by the machine or control.

ToolTypes

The *ToolTypes* component lists all tool types available at the specific machine or control using the 4.21 "ToolType-DescriptionType". The available tool types may change between different control models and software versions. ("Annex A: Tool Data Reference" lists the tool types with their unique string identifiers.)

Each *ToolType* refers to the *ToolDataItems* that belong to it using *AssociatedWith* references. A tool type is linked for instance with the data items for its geometrical characteristics, life time, and additional technology information and settings that might be used during machining cycles.

The assignment of tool types and data items can change (e.g., if a new control software version makes a technology or machining cycle available for more tool types). The relation is provided by the instances at the server.

Additional tool data items of the machine manufacturer:
With NC software version 17, all machine-specific tool data items of the machine manufacturer are taken as potentially relevant for all tool types. The references depend only on the internal data source table. There is no special handling or restriction of the *AssociatedWith* references between the *ToolTypes* and the *ToolDataItems* of the *ManufacturerExtension* category.

4.18 ToolDataCategoryType

Overview

A *ToolDataCategoryType* instance is used to group 4.19 "ToolDataItemType" instances of one tool data category.

It is used, for example, at the 4.17 "ToolDataRepresentationType" instance at 4.16 "ToolDataManagementType": The *BrowseNames* of *ToolDataCategory* instances can be used as argument at the *GetToolDataByCategory* method to access tool data.

Attributes

Table 135: ToolDataCategoryType Definition Attributes

Attribute	Value
BrowseName	ToolDataCategoryType
IsAbstract	false

References

ToolDataCategoryType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 136: ToolDataCategoryType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	<ToolDataItem>		ToolDataItemType	OptionalPlacehold-er

<ToolDataItem>

Placeholder for 4.19 "ToolDataItemType" instances of this data category.

4.19 ToolDataItemType

Overview

A *ToolDataItemType* instance describes one tool data item with its data type, default value and metadata like value ranges, engineering units or enumeration values.

Instances of *ToolDataItemType* are used at the 4.17 "ToolDataRepresentationType" to describe the tool data of a machine.

See Table 215 in "Annex A: Tool Data Reference" for a list of tool data items defined by HEIDENHAIN.

Attributes

Table 137: ToolDataItemType Definition Attributes

Attribute	Value
BrowseName	ToolDataItemType
IsAbstract	false

References

ToolDataItemType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 138: ToolDataItemType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	JsonIdentifier	String	PropertyType	Mandatory
HasComponent	Variable	ValueDescription	BaseDataType	BaseDataVariable-Type	Mandatory
HasComponent	Variable	FileLocation	String	BaseDataVariable-Type	Optional
HasComponent	Variable	FileLocationManu-facturer	String	BaseDataVariable-Type	Optional

Additional Subcomponents

Some components of the *ToolDataItemType* have additional components, which are defined in Table 139.

Table 139: ToolDataItemType Additional Subcomponents

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
FileLocation	HasProperty	Variable	NodeId	NodeId	PropertyType	Mandatory
FileLocation-Manufactur-er	HasProperty	Variable	NodeId	NodeId	PropertyType	Mandatory

JsonIdentifier

The identifier of this data item used at the JSON tool data file and schema.

See 3.10 "Tool Data Management" and 4.22 "LocalToolDataSetType".

ValueDescription

The *ValueDescription* variable provides the data type, default value and additional metadata using OPC UA standard *VariableTypes* with their defined properties. Possible *VariableTypes* with their *Properties* are given in Table 140. This list might be extended in future versions.

The data type and default value of a concrete data item are given at the variable's *DataType* and *Value* attributes. (It is possible that the default value is empty or *Null*.)

Table 140: ValueDescription VariableTypes with DataTypes and Properties used at instances

VariableType	DataType	Possible Properties
SelectionListType	String	Selections
		RestrictToList
BaseDataVariableType	String	MaxStringLength
	UtcTime	
	ToolRecordIdentifierDataType	
	Boolean	
TwoStateDiscreteType	Boolean	FalseState
		TrueState
MultiStateValueDiscreteType	UInt32	EnumValues
	Int32	ValueAsText
BaseAnalogType	Byte	InstrumentRange
	UInt32	InstrumentRange
	Int32	EngineeringUnits
	Double	InstrumentRange
		ValuePrecision
		EngineeringUnits

FileLocation

Some tool data item values are names of files (e.g., the name of a 3D model file of the tool carrier). These files are stored in a specific directory in the file system of the machine. The optional *FileLocation* component is used at tool data items that denote files. The value is the path of the directory.

The property *NodeId* contains the corresponding *NodeId* of the *FileDirectory* in the *FileSystem* of the machine. See 9 "Machine File System Access".

FileLocationManufacturer

Similar to *FileLocation* , the *FileLocationManufacturer* provides a secondary location for tool data files. Usually only machine manufacturers have access to this location.

If *FileLocation* and *FileLocationManufacturer* both contain a file with the name from the tool record, the machine uses the file at *FileLocation*.

4.20 ToolTypeCategoryType

Overview

A *ToolTypeCategoryType* instance is used to group 4.21 "ToolTypeDescriptionType" instances of one category.
It is used, for example, at the 4.17 "ToolDataRepresentationType".

Attributes

Table 141: ToolTypeCategoryType Definition Attributes

Attribute	Value
BrowseName	ToolTypeCategoryType
IsAbstract	false

References

ToolTypeCategoryType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 142: ToolTypeCategoryType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	<ToolTypeDescrip-tion>		ToolType-DescriptionType	OptionalPlacehold-er

<ToolTypeDescription>

Placeholder for 4.21 "ToolTypeDescriptionType" instances of this category.

4.21 ToolTypeDescriptionType

Overview

A *ToolTypeDescriptionType* instance represents a tool type.

It is used, for example, at the 4.17 "ToolDataRepresentationType" instance at 4.16 "ToolDataManagementType".

See Table 216 in "Annex A: Tool Data Reference" for a list of tool types.

Attributes

Table 143: ToolTypeDescriptionType Definition Attributes

Attribute	Value
BrowseName	ToolTypeDescriptionType
IsAbstract	false

References

*ToolTypeDescriptionType*is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 144: ToolTypeDescriptionType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	Identifier	String	PropertyType	Mandatory
HasProperty	Variable	Name	Localized-Text	PropertyType	Mandatory

Identifier

The unique string identifier of the tool type.

Name

The name of the tool type.

4.22 LocalToolDataSetType

Overview

A *LocalToolDataSetType* instance provides the functionality to initialize a local copy of the machine's tool data and keep it up to date using update events. For example, the tool database of a central tool management system can also be synchronized with the machine's tool data.

The *LocalToolDataSet* functionality consists of a method to create and download a JSON file containing the machine's current tool data, the corresponding JSON schema, and events containing change information of the tool records. The availability and status of the functionality is provided by an extra variable.

See also Synchronizing a Local Tool Data Set with the Machine in 3.10 "Tool Data Management" for an example sequence.

For data items defined by HEIDENHAIN and more information about machine-specific tool data extension of the machine manufacturer, see "Annex A: Tool Data Reference".

Attributes

Table 145: LocalToolDataSetType Definition Attributes

Attribute	Value
BrowseName	LocalToolDataSetType
IsAbstract	false

References

LocalToolDataSetType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 146: LocalToolDataSetType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
GeneratesEvent	EventType	ToolDataSetModificationEventType defined in 5.7			
HasComponent	Variable	Synchronization-Status	ToolData-Synchro-nization-StatusType	BaseDataVariable-Type	Mandatory
HasComponent	Method	CreateToolDataFile			Mandatory
HasComponent	Object	ToolDataSchema		FileType	Mandatory

SynchronizationStatus

The *SynchronizationStatus* variable provides information about the availability and status of the *LocalDataSet* functionality using the 7.6 "ToolDataSynchronizationStatusType".

If the server does not have a connection to the control or until the NC software is fully initialized after start-up of the machine, the status is *NotAvailable*. While the status is *NotAvailable*, calling the *CreateToolDataFile* method is not successful and no update events are emitted.

During the status *Idle* the server is ready to provide the *LocalDataSet* functionality. The method *CreateToolDataFile* can be used from now on.

To reduce load at the machine, the update events are only generated if at least one client is interested in the events of type 5.7 "ToolDataSetModificationEventType". A client shows its interest in the update events by subscribing to events of type 5.7 "ToolDataSetModificationEventType". As soon as at least one client subscribes to the events, the value of the variable changes from *Idle* to *Initializing*. The server initializes access to the data and starts sending the update events at tool data changes.

While the value is *Synchronized*, the events of type 5.7 "ToolDataSetModificationEventType" are emitted. The server keeps this status until the last client is no longer interested in the update events or the NC software or the machine is being shut down.

CreateToolDataFile

Triggers the creation of a JSON file containing the current tool data of the machine.

The *ResultFile* argument contains the *NodeId* of the created file. The *NodeId* belongs to a *0:FileType* instance at the *Machine's FileSystem*. The JSON file can be downloaded using the methods of the *File* object. (To save memory resources at the machine, a client application should delete the *File* after downloading it.) See 9 "Machine File System Access".

The file contains a version number. The version number corresponds to the version of the update event contents provided in the *Version* property of the 7.9 "ToolRecordModificationDataType" (see 5.7 "ToolDataSetModificationEventType"). The version numbers are not synchronized over OPC UA server life cycles.

Table 147: CreateToolDataFile Attributes

Attribute	Value
BrowseName	CreateToolDataFile

Signature

```
CreateToolDataFile{
    [out] NodeId            ResultFile
};
```

Table 148: CreateToolDataFile Signature

Argument	Description
ResultFile	NodeId of the created file

Table 149: CreateToolDataFile Result Codes

Result Code	Description
BadInvalidState	The server cannot provide the functionality at the moment (e.g., because the control is not fully initialized). The creation of the file is possible during *Machine State NCIsAvailable*.
BadOutOfMemory	Too many files have been created and are still present in the machine's memory. (A client application should delete the created file after downloading it.)
BadOutOfService	Returned when the server has lost its internal connection to the control.
BadTimeout	The server's internal request to create the file timed out. The creation of the file took too long.
BadNoMatch	The server was not able to determine the *NodeId* of the created file.
BadInternalError	Some other internal error occurred.

ToolDataSchema

The *ToolDataSchema* component is an instance of *0:FileType*. The schema file contains the JSON schema describing the tool data export JSON file.

The schema file is created based on the current configuration of the machine. It also contains the machine-specific changes and extensions of the machine manufacturer.

The ID of the schema contains a check value. If tool data relevant configuration items change, the check value indicates it.

4.23 ServiceFileInterfaceType

Overview

A *ServiceFileInterfaceType* instance provides methods to create service files and the corresponding file system download information.

A service file contains information about the current machine and operation status. Machine manufacturers or HEIDENHAIN technical support can use service files to analyze issues of the machine.

A service file contains current NC data (e.g., selected NC programs < 10 MB, tool data, and other information).

By relaying a service file, you declare you consent to your machine manufacturer or DR. JOHANNES HEIDENHAIN GmbH using this data for diagnostic purposes.

If you do not consent to this, then please first edit the service file. Remove all confidential data before relaying the file. Take into consideration that this might complicate the error analysis.

To save memory at the machine, client applications are recommended to delete no longer required service files using the *FileSystem* methods of the *Machine* (for example, after downloading the files). See 9 "Machine File System Access".

If there are too many files with similar names at the same location, the control deletes older service files automatically.

Attributes

Table 150: ServiceFileInterfaceType Definition Attributes

Attribute	Value
BrowseName	ServiceFileInterfaceType
IsAbstract	false

References

ServiceFileInterfaceType is a subtype of *BaseObjectType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 151: ServiceFileInterfaceType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	ArgumentDescriptions		BaseObjectType	Optional
HasComponent	Method	CreateServiceFile			Mandatory
HasComponent	Method	CreateServiceFileByNodeId			Mandatory
HasComponent	Variable	LastCreatedServiceFile	String	BaseDataVariableType	Mandatory

Additional Subcomponents

Some components of the *ServiceFileInterfaceType* have additional components, which are defined in Table 152.

Table 152: ServiceFileInterfaceType Additional Subcomponents

Source Path	ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
LastCreated-ServiceFile	HasProperty	Variable	FileNodeId	NodeId	PropertyType	Mandatory
Argument-Descriptions	HasComponent	Variable	FileName-Prefix	String	BaseDataVariable-Type	Optional
Argument-Descriptions	HasComponent	Variable	Location	NodeId	BaseDataVariable-Type	Optional
Argument-Descriptions	HasComponent	Variable	Path	String	BaseDataVariable-Type	Optional

ArgumentDescriptions

The *ArgumentDescriptions* component groups the variables used to provide additional information about the arguments of the methods. At instances of the *ServiceFileInterfaceType*, the variable values are filled with concrete values (e.g., the default location in the file system for new service files).

LastCreatedServiceFile

The methods *CreateServiceFile* or *CreateServiceFileByNodeId* can be used to trigger the creation of a service file. If one of the two methods was called, and after the creation process has been finished, the variable *LastCreatedServiceFile* contains the path to the resulting service file in the file system of the machine. (The creation of service files using the other interfaces of the machine does not have any effect on the variable.)

The property *FileNodeId* contains the *NodeId* of the *FileType* instance node representing the service file in the *FileSystem*.

The service file can be downloaded using the *NodeId* of the *File*. To save memory resources at the machine, a client application should delete the *File* after downloading it. After the referred file has been removed, the value of the variable is emptied and the status code is set to *GoodNoData*. See also 9 "Machine File System Access".

If the creation of the last service file was not successful, the status code of the variable is *BadDeviceFailure*.

CreateServiceFile

Triggers the creation of a service file.

If the method returns *StatusCode Good*, the creation process has been started successfully. Depending on the amount of data to integrate in the service file, the creation can take from a few seconds up to very few minutes. After the process has been finished, the variable *LastCreatedServiceFile* is updated, see LastCreatedServiceFile.

Table 153: CreateServiceFile Attributes

Attribute	Value
BrowseName	CreateServiceFile

Signature

```
    CreateServiceFile{
        [in] String            FileNamePrefix
        [in] String            Path
    };
```

Table 154: CreateServiceFile Signature

Argument	Description
FileNamePrefix	The prefix that is used for the service file name. By system default, the resulting file name will also contain the creation time stamp. If the argument is empty or omitted, the system default name is used.
Path	Full path where the service file will be stored. If the argument is empty or omitted, the system default path is used.

Along with the *InputArguments* and *OutputArguments* properties, the *CreateServiceFile* method has the references listed in Table 155.

Table 155: CreateServiceFile Additional References

ReferenceType	IsForward	Target Path
HasOptionalInputArgumentDescription	True	ArgumentDescriptions ▶ FileNamePrefix
HasOptionalInputArgumentDescription	True	ArgumentDescriptions ▶ Path

Table 156: CreateServiceFile Result Codes

Result Code	Description
BadInvalidState	The creation of a service file could not be started at the moment. Possible reasons are: ■ The machine is already creating a service file at the moment. ■ The server is not fully initialized yet or is shutting down.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*. Result codes of input arguments at *inputArgumentResults* (except *Good*): FileNamePrefix ■ *BadInvalidArgument*: the file name prefix is invalid (e.g., it contains spaces) ■ *BadNotSupported*: the file name would exceed the maximum file system path length ■ *Uncertain*: the destination path including the file name would exceed the maximum file system path length Path ■ *BadInvalidArgument*: the path does not specify a valid target directory for a service file (e.g., it is a relative path or contains invalid characters) ■ *BadOutOfRange*: the path denotes a location in the file system, where the service file cannot be stored

Result Code	Description
	■ *BadNotFound*: the path does not denote a location on an existing partition in the file system of the machine ■ *BadNoMatch*: the path (or a part of it) denotes a file instead of a directory ■ *BadUserAccessDenied*: the user has insufficient rights to create a service file at this location ■ *BadNotSupported*: the path denotes a location that exceeds the maximum file system path length ■ *Uncertain*: the path including the file name would exceed the maximum file system path length
Bad	Saving the service file failed (e.g., due to missing write-rights at the specified file system location or incomplete service file configuration). Machine-specific adaptations of the service file configuration can be done by the machine manufacturer.
BadInternalError	Some other internal error occurred; creation of a service file could not be triggered.
BadUnexpectedError	Some other unexpected occurred error during execution. In case of this issue, contact HEIDENHAIN technical support.

CreateServiceFileByNodeId

Triggers the creation of a service file similar to *CreateServiceFile*. Instead of using a string path to specify the destination directory in the *FileSystem*, the *NodeId* of the *FileDirectory* is used.

If the method returns *StatusCode Good*, the creation process has been started successfully. Depending on the amount of data to integrate in the service file, the creation can take from a few seconds up to very few minutes. After the process has been finished, the variable *LastCreatedServiceFile* is updated, see LastCreatedServiceFile.

Table 157: CreateServiceFileByNodeId Attributes

Attribute	Value
BrowseName	CreateServiceFileByNodeId

Signature

```
    CreateServiceFileByNodeId{
        [in] String            FileNamePrefix
        [in] NodeId            Location
    };
```

Table 158: CreateServiceFileByNodeId Signature

Argument	Description
FileNamePrefix	The prefix that will be used for the service file name. By system default, the resulting file name will also contain the creation time stamp. If the argument is empty or omitted, the systems default name is used.
Location	*NodeId* of the *FileDirectory* where the service file will be stored. If the argument is empty or omitted, the systems default location is used.

Along with the *InputArguments* and *OutputArguments* properties, the *CreateServiceFileByNodeId* method has the references listed in Table 159.

Table 159: CreateServiceFileByNodeId Additional References

ReferenceType	IsForward	Target Path
HasOptionalInputArgumentDescription	True	ArgumentDescriptions ▶ FileNamePrefix
HasOptionalInputArgumentDescription	True	ArgumentDescriptions ▶ Location

Table 160: CreateServiceFileByNodeId Result Codes

Result Code	Description
BadInvalidState	The creation of a service file could not be started at the moment. Possible reasons are: ■ The machine is already creating a service file at the moment. ■ The server is not fully initialized yet or is shutting down.
BadInvalidArgument	At least one of the input arguments was invalid; more information is given at the *inputArgumentResults*. Result codes of input arguments at *inputArgumentResults* (except *Good*): FileNamePrefix ■ *BadInvalidArgument*: the file name prefix is invalid (e.g., it contains spaces) ■ *BadNotSupported*: the file name would exceed the file system maximum ■ *Uncertain*: the destination path including the file name would exceed the maximum file system path length Location ■ *BadInvalidArgument*: the *NodeId* does not specify a valid target directory for a service file ■ *BadNoMatch*: the node with the given *NodeId* is not a *FileDirectory* ■ *BadNodeIdUnknown*: no node with the given *NodeId* exists

Result Code	Description
	■ *BadUserAccessDenied*: the user has insufficient rights to create a service file at this location ■ *BadNotSupported*: the location has a path length exceeds the maximum file system path length ■ *Uncertain*: the resulting path including the file name would exceed the maximum file system path length
Bad	Saving the service file failed (e.g., due to missing write-rights at the specified file system location or incomplete service file configuration). Machine-specific adaptations of the service file configuration can be done by the machine manufacturer.
BadInternalError	Some other internal error occurred; creation of a service file could not be triggered.
BadUnexpectedError	Some other unexpected occurred error during execution. In case of this issue, contact HEIDENHAIN technical support.

5

OPC UA EventTypes

5.1 ErrorEventType

Overview

ErrorEventType is an abstract *EventType* indicating a change in a 4.6 "ErrorEntryListType" instance. It contains also all information of the related error object.

See also 3.7 "Errors, Warnings and Notifications" for an overview of the HEIDENHAIN control error concept and its representation in the address space.

Attributes

Table 161: ErrorEventType Definition Attributes

Attribute	Value
BrowseName	ErrorEventType
IsAbstract	true

References

ErrorEventType is a subtype of *BaseEventType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 162: ErrorEventType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasSubtype	EventType	ErrorClearedEventType defined in 5.2			
HasSubtype	EventType	ErrorOccurredEventType defined in 5.3			
HasProperty	Variable	Action	String	PropertyType	Mandatory
HasProperty	Variable	Cause	String	PropertyType	Mandatory
HasProperty	Variable	Channel	UInt32	PropertyType	Mandatory
HasProperty	Variable	Class	ErrorClass-Type	PropertyType	Mandatory
HasProperty	Variable	Group	ErrorGroup-Type	PropertyType	Mandatory
HasProperty	Variable	Internals	String	PropertyType	Mandatory
HasProperty	Variable	Location	Error-Location-Type	PropertyType	Mandatory
HasProperty	Variable	Number	UInt32	PropertyType	Mandatory
HasProperty	Variable	NumberAsText	String	PropertyType	Mandatory
HasProperty	Variable	Text	String	PropertyType	Mandatory
HasProperty	Variable	ErrorEntryNodeId	NodeId	PropertyType	Mandatory

Action

A description of possible actions that can be taken to fix the error.

Cause

The cause that triggered the error.

Channel

The ID of the channel where the error originated. If the error is not related to a channel, the value is the data type's corresponding maximum integer value.

(The status code of the *Channel* property value of the corresponding 4.5 "ErrorEntryType" instance is set to *GoodNoData*.)

Class

Classification of the error. The value of *Class* is a 7.1 "ErrorClassType" enumeration value.

Group
The group that the error belongs to. The value of *Group* is a 7.2 "ErrorGroupType" enumeration value.

Internals
Internal error details as multi-line text.

Location
The location of the error. The value of *Location* is a 7.3 "ErrorLocationType" enumeration value.

Number
The number of the error.

Each error has a number. But because of, for example, the possibility of parameterizing the error message texts, the number is not a unique identifier of an error object.

NumberAsText
Error number as displayed on the control.

Text
Error message text that is displayed on the control's error list.

ErrorEntryNodeId
NodeId of the corresponding 4.5 "ErrorEntryType" instance within at least one *ErrorEntryList* instance.

5.2 ErrorClearedEventType

Overview

ErrorClearedEventType is a subtype of 5.1 "ErrorEventType" indicating that an error is cleared and removed from a 4.6 "ErrorEntryListType" instance. It provides all properties of the 4.5 "ErrorEntryType" instance that is cleared.

See also 3.7 "Errors, Warnings and Notifications" for an overview of the HEIDENHAIN control error concept and its representation in the address space.

Attributes

Table 163: ErrorClearedEventType Definition Attributes

Attribute	Value
BrowseName	ErrorClearedEventType
IsAbstract	false

References

ErrorClearedEventType is a subtype of 5.1 "ErrorEventType", which means it inherits the *InstanceDeclarations* of that Node.

5.3 ErrorOccurredEventType

Overview

ErrorOccurredEventType is a subtype of 5.1 "ErrorEventType" indicating that an error occurred and was added to a 4.6 "ErrorEntryListType" instance. It provides all properties of the 4.5 "ErrorEntryType" instance that occurred.

See also 3.7 "Errors, Warnings and Notifications" for an overview of the HEIDENHAIN control error concept and its representation in the address space.

Attributes

Table 164: ErrorOccurredEventType Definition Attributes

Attribute	Value
BrowseName	ErrorOccurredEventType
IsAbstract	false

References

ErrorOccurredEventType is a subtype of 5.1 "ErrorEventType", which means it inherits the *InstanceDeclarations* of that *Node*.

5.4 NCTransitionEventType

Overview

An event reported when the state of, for example, a 4.2 "NCStateMachineType" instance changes. See 3.4 "NC State Machine" and 3.6 "NC Program Execution Monitoring and Control" for details.

Attributes

Table 165: NCTransitionEventType Definition Attributes

Attribute	**Value**
BrowseName	NCTransitionEventType
IsAbstract	false

References

NCTransitionEventType is a subtype of *TransitionEventType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 166: NCTransitionEventType Definition References

ReferenceType	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	0:Message	Localized-Text	PropertyType	Mandatory
HasComponent	Variable	TransitionReason	Localized-Text	BaseDataVariable-Type	Mandatory

TransitionReason

The reason why the state transition has taken place.

5.5 ExecutionMessageEventType

Overview

ExecutionMessageEventType events are messages sent by the NC program on specific NC program commands (FN 38: SEND). It can be used for communication between an NC program and a remote application.

Attributes

Table 167: ExecutionMessageEventType Definition Attributes

Attribute	Value
BrowseName	ExecutionMessageEventType
IsAbstract	false

References

ExecutionMessageEventType is a subtype of *BaseEventType*, which means it inherits the *InstanceDeclarations* of that *Node*.

5.6 ToolChangedEventType

Overview

An event of type *ToolChangedEventType* is reported when a tool change of a channel has been completed. It provides information about the exchanged tools.

HEIDENHAIN recommends using the *CurrentTool* of 4.12 "ChannelType" added with version 1.03 to identify the currently used tool and monitor it with a data change subscription.
For compatibility reasons, events of type *ToolChangedEventType* are still provided by the OPC UA NC Server at the moment. The support of the events might be discontinued in a future version of the server.

Attributes

Table 168: ToolChangedEventType Definition Attributes

Attribute	Value
BrowseName	ToolChangedEventType
IsAbstract	false

References

ToolChangedEventType is a subtype of *BaseEventType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 169: ToolChangedEventType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	ToolInId	UInt32	PropertyType	Mandatory
HasProperty	Variable	ToolOutId	UInt32	PropertyType	Mandatory
HasProperty	Variable	ChannelId	UInt32	PropertyType	Mandatory
HasProperty	Variable	ToolInIndex	UInt32	PropertyType	Mandatory
HasProperty	Variable	ToolOutIndex	UInt32	PropertyType	Mandatory
HasProperty	Variable	0:Message	Localized-Text	PropertyType	Mandatory

ToolInId

The number of the tool that was inserted.

ToolOutId

The number of the tool that was removed.

ChannelId

The machining channel where the tool change has taken place.

ToolInIndex

The index of the tool that was inserted.

ToolOutIndex

The index of the tool that was removed.

5.7 ToolDataSetModificationEventType

Overview

ToolDataSetModificationEventType is an event describing changes of the machine's tool data.

It is part of the functionality of a *LocalToolDataSet* of 4.22 "LocalToolDataSetType".

Attributes

Table 170: ToolDataSetModificationEventType Definition Attributes

Attribute	Value
BrowseName	ToolDataSetModificationEventType
IsAbstract	false

References

ToolDataSetModificationEventType is a subtype of *BaseEventType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 171: ToolDataSetModificationEventType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Variable	Records	ToolRecord-Modifi-cation-DataType[]	BaseDataVariable-Type	Mandatory

Records

Each element of the *Records* array contains the description of the change that has happened to one tool record.

5.8 BaseToolEventType

Overview

BaseToolEventType is an abstract *EventType* and supertype to more specific tool (data) related event types.
See also "Terms", Page 24.

Attributes

Table 172: BaseToolEventType Definition Attributes

Attribute	Value
BrowseName	BaseToolEventType
IsAbstract	true

References

BaseToolEventType is a subtype of *BaseEventType*, which means it inherits the *InstanceDeclarations* of that *Node*.

Table 173: BaseToolEventType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasSubtype	EventType	ToolLockedEventType defined in 5.9			
HasProperty	Variable	ToolDatabaseId	String	PropertyType	Mandatory
HasProperty	Variable	ToolNumber	UInt32	PropertyType	Mandatory
HasProperty	Variable	ToolIndex	Byte	PropertyType	Mandatory
HasProperty	Variable	ToolName	String	PropertyType	Optional

ToolDatabaseId

The database ID of the tool that the event concerns.

ToolNumber

The number of the tool that the event concerns. The tool record is identified by *ToolNumber* and *ToolIndex*.

ToolIndex

The index of the tool that the event concerns. The tool record is identified by *ToolNumber* and *ToolIndex*.

ToolName

The name of the tool (index) that the event concerns.

5.9 ToolLockedEventType

Overview

ToolLockedEventType is a subtype of 5.8 "BaseToolEventType" indicating that the *Locked* status of a tool record has changed. In case of an indexed tool, each index has its own *Locked* status.

If a tool (index) has been locked (*Locked* is *True*) the tool (index) cannot be used anymore, meaning that it cannot successfully be called and used for machining operations. Possible reasons are, for example, expiration of the tool's life time, a detected broken tool, or a manual change of the tool's status after an visual inspection by the operator. If one index of an indexed tool is locked, other indices of the tool are not necessarily affected.

Attributes

Table 174: ToolLockedEventType Definition Attributes

Attribute	Value
BrowseName	ToolLockedEventType
IsAbstract	false

References

ToolLockedEventType is a subtype of 5.8 "BaseToolEventType", which means it inherits the *InstanceDeclarations* of that *Node*.

Table 175: ToolLockedEventType Definition References

ReferenceType	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasProperty	Variable	Locked	Boolean	PropertyType	Mandatory
HasProperty	Variable	0:Message	String	PropertyType	Mandatory

Locked

The new *Locked* status of the tool (index).

6

OPC UA Variable-Types

6.1 CutterLocationArrayType

Overview

A variable containing an array of cutter location coordinates of type 7.7 "CutterLocationDataType" used to indicate the position of the tool tip (see 4.12 "ChannelType"). Each entry of the array contains one coordinate.

Attributes

Table 176: CutterLocationArrayType Definition Attributes

Attribute	Value
BrowseName	CutterLocationArrayType
IsAbstract	false
ValueRank	1 (1 = OneDimension)
DataType	CutterLocationDataType
ArrayDimensions	{0} (0 = UnknownSize)

References

CutterLocationArrayType is a subtype of *BaseDataVariableType*, which means it inherits the *InstanceDeclarations* of that *Node*.

6.2 ProgramPositionArrayType

Overview

A variable of type *ProgramPositionArrayType* is used to describe a program call stack. Each entry describes a position within a program using the 7.8 "ProgramPositionDataType". The entry with *CallStack* 0 denotes the main program, followed by the first called subprogram with 1, if one is called. The entry with the highest *CallStack* number describes the (sub)program being executed as well as the respective current block.

Attributes

Table 177: ProgramPositionArrayType Definition Attributes

Attribute	Value
BrowseName	ProgramPositionArrayType
IsAbstract	false
ValueRank	1 (1 = OneDimension)
DataType	ProgramPositionDataType
ArrayDimensions	{0} (0 = UnknownSize)

References

ProgramPositionArrayType is a subtype of *BaseDataVariableType*, which means it inherits the *InstanceDeclarations* of that *Node*.

7

OPC UA DataTypes

7.1 ErrorClassType

Overview

The *ErrorClassType* enumeration is used for example at the 4.5 "ErrorEntryType" *Class* property to classify *ErrorEntries* as, for example, errors, warnings and notifications.

Attributes

Table 178: ErrorClassType Definition Attributes

Attribute	Value
BrowseName	ErrorClassType
IsAbstract	false

References

ErrorClassType is a subtype of *Enumeration*, which means it inherits the *InstanceDeclarations* of that *Node*.

Enumeration

Table 179: ErrorClassType Values

Value	DisplayName	Description
0	None	No error class has been specified by the underlying systems
1	Warning	A warning without effect on the machine or execution
2	FeedHold	An error with execution feed hold
3	ProgramHold	An error with execution program hold
4	ProgramAbort	An error resulting in execution program abort
5	EmergencyStop	An error resulting in a machine emergency stop
6	Reset	An error resulting in a machine emergency stop; a control reset is required
7	Info	A message without any effect on the machine or execution
8	Error	An error without any effect on the machine or execution
9	Note	A notification without any effect on the machine or execution

7.2 ErrorGroupType

Overview

The *ErrorGroupType* enumeration is used for example at the 4.5 "ErrorEntryType" *Group* property. It denotes for example whether an error is related to a programming error, the PLC or the execution of a Python script.

Attributes

Table 180: ErrorGroupType Definition Attributes

Attribute	Value
BrowseName	ErrorGroupType
IsAbstract	false

References

ErrorGroupType is a subtype of *Enumeration*, which means it inherits the *InstanceDeclarations* of that *Node*.

Enumeration

Table 181: ErrorGroupType Values

Value	DisplayName	Description
0	None	No error group has been specified by the underlying systems
1	Operating	Operating errors
2	Programming	Programming errors
3	PLC	PLC errors
4	General	General errors
5	Remote	Errors raised by remote clients
6	Python	Errors raised by Python scripts

7.3 ErrorLocationType

Overview

The *ErrorLocationType* enumeration is used for example at the 4.5 "ErrorEntryType" *Location* property. It describes whether an error is raised during a machining or editor process for example.

Attributes

Table 182: ErrorLocationType Definition Attributes

Attribute	Value
BrowseName	ErrorLocationType
IsAbstract	false

References

ErrorLocationType is a subtype of *Enumeration*, which means it inherits the *InstanceDeclarations* of that *Node*.

Enumeration

Table 183: ErrorLocationType Values

Value	DisplayName	Description
0	None	No error location has been specified by the underlying systems
1	Machine	The error originated in the machining process
2	Edit	The error originated in the editor process
3	OEM	The error originated in a process of the machine manufacturer

7.4 NCOperatingMode

Overview

The *NCOperatingMode* enumeration is used for example at the 4.12 "ChannelType" to describe the current operating mode of the machining channel.

Attributes

Table 184: NCOperatingMode Definition Attributes

Attribute	Value
BrowseName	NCOperatingMode
IsAbstract	false

References

NCOperatingMode is a subtype of *Enumeration*, which means it inherits the *InstanceDeclarations* of that *Node*.

Enumeration

Table 185: NCOperatingMode Values

Value	DisplayName	Description
0	Manual	Manual operation mode
1	MDI	Manual Data Input
2	RFP	Traversing the reference points
3	SingleStep	Program Run Single Block
4	Automatic	Program Run Full Sequence
5	Other	Operating mode is other than the known ones
6	Handwheel	Electronic handwheel operating mode

7.5 ToolRecordModificationType

Overview

The *ToolRecordModificationType* enumeration describes the different kinds of modifications of a tool record. It is used at the 7.9 "ToolRecordModificationDataType".

Attributes

Table 186: ToolRecordModificationType Definition Attributes

Attribute	Value
BrowseName	ToolRecordModificationType
IsAbstract	false

References

ToolRecordModificationType is a subtype of *Enumeration*, which means it inherits the *InstanceDeclarations* of that *Node*.

Enumeration

Table 187: ToolRecordModificationType Values

Value	DisplayName	Description
0	Insert	A new record has been added
1	Replace	Contents of an existing record have been replaced
2	Delete	An existing record has been removed

7.6 ToolDataSynchronizationStatusType

Overview

The *ToolDataSynchronizationStatusType* enumeration is used at the 4.22 "LocalToolDataSetType" to indicate the status including the availability of its functionality.

Attributes

Table 188: ToolDataSynchronizationStatusType Definition Attributes

Attribute	Value
BrowseName	ToolDataSynchronizationStatusType
IsAbstract	false

References

ToolDataSynchronizationStatusType is a subtype of *Enumeration*, which means it inherits the *InstanceDeclarations* of that *Node*.

Enumeration

Table 189: ToolDataSynchronizationStatusType Values

Value	DisplayName	Description
0	NotAvailable	Update information is not available and change notifications are paused
1	Idle	Update information is available, but synchronization is not enabled
2	Initializing	Synchronization is in progress
3	Synchronized	Updates refer to an already initialized data set

7.7 CutterLocationDataType

Overview

CutterLocationDataType is a *Structure* describing one coordinate of the location of the cutter. An array of *CutterLocations* describes the location of the cutter in a multidimensional space (see 6.1 "CutterLocationArrayType").

Attributes

Table 190: CutterLocationDataType Definition Attributes

Attribute	Value
BrowseName	CutterLocationDataType
IsAbstract	false

References

CutterLocationDataType is a subtype of *Structure*, which means it inherits the *InstanceDeclarations* of that *Node*.

Structure

Table 191: CutterLocationDataType Structure

Name	Type	ValueRank	Description
Position	Double	−1 (−1 = Scalar)	The position value
PositionEngineering-Units	EUInformation	−1 (−1 = Scalar)	The engineering unit of the position value
CoordinateName	String	−1 (−1 = Scalar)	The name of this coordinate

7.8 ProgramPositionDataType

Overview

ProgramPostitionDataType is a *Structure* describing one line within a program using the name of the program and a block number with its content. An array of *ProgramPositions* is used to describe a whole program call stack from the main program to the currently executed subprogram (see 3.6 "NC Program Execution Monitoring and Control" and 6.2 "ProgramPositionArrayType").

In order to define the order that the programs have been called in, *ProgramPositionDataType* additionally has a *CallStackLevel*.

Attributes

Table 192: ProgramPositionDataType Definition Attributes

Attribute	Value
BrowseName	ProgramPositionDataType
IsAbstract	false

References

ProgramPositionDataType is a subtype of *Structure*, which means it inherits the *InstanceDeclarations* of that *Node*.

Structure

Table 193: ProgramPositionDataType Structure

Name	Type	ValueRank	Description
ProgramName	String	−1 (−1 = Scalar)	NC program name
BlockNumber	UInt32	−1 (−1 = Scalar)	Block number of the NC program
BlockContent	String	−1 (−1 = Scalar)	Block content of the NC program
CallStackLevel	UInt32	−1 (−1 = Scalar)	The level of depth of the call stack. The main program is at CallStackLevel 0. The CallStackLevel is incremented for each subsequent subprogram

7.9 ToolRecordModificationDataType

Overview

ToolRecordModificationDataType is a *Structure* describing a change of a tool data record at the machine. If it is an insertion or update, it contains also the new record. It is used at 5.7 "ToolDataSetModificationEventType".

Attributes

Table 194: ToolRecordModificationDataType Definition Attributes

Attribute	Value
BrowseName	ToolRecordModificationDataType
IsAbstract	false

References

ToolRecordModificationDataType is a subtype of *Structure*, which means it inherits the *InstanceDeclarations* of that *Node*.

Structure

Table 195: ToolRecordModificationDataType Structure

Name	Type	ValueRank	Description
Version	UInt32	−1 (−1 = Scalar)	The data set version that this change refers to
RecordIdentifier	ToolRecord-IdentifierDataType	−1 (−1 = Scalar)	Tool number and index of the tool record
ModificationType	ToolRecordModifi-cationType	−1 (−1 = Scalar)	The kind of modification that has happened
RecordUpdate	String	−1 (−1 = Scalar)	The new record content as string

7.10 ToolRecordIdentifierDataType

Overview

ToolRecordIdentifierDataType is a *Structure* describing the identifier of a tool record. It consists of a *ToolNumber* and a *ToolIndex*. A *ToolRecordIdentifier* uniquely identifies a tool record within a machine.

See also "Terms", Page 24.

Attributes

Table 196: ToolRecordIdentifierDataTypeDefinition Attributes

Attribute	Value
BrowseName	ToolRecordIdentifierDataType
IsAbstract	false

References

ToolRecordIdentifierDataType is a subtype of *Structure*, which means it inherits the *InstanceDeclarations* of that *Node*.

Structure

Table 197: ToolRecordModificationDataType Structure

Name	Type	ValueRank	Description
ToolNumber	UInt32	−1 (−1 = Scalar)	The tool's number
ToolIndex	Byte	−1 (−1 = Scalar)	The tool's additional index

7.11 MagazinePocketIdentifierDataType

Overview

MagazinePocketIdentifierDataType is a *Structure* describing the identifier of a pocket in a tool magazine of the machine. It consists of a *MagazineNumber* and a *PocketNumber*. A *MagazinePocketIdentifier* uniquely identifies a tool pocket within a machine.

Attributes

Table 198: MagazinePocketIdentifierDataType Attributes

Attribute	Value
BrowseName	MagazinePocketIdentifierDataType
IsAbstract	false

References

MagazinePocketIdentifierDataType is a subtype of *Structure*, which means it inherits the *InstanceDeclarations* of that *Node*.

Structure

Table 199: MagazinePocketIdentifierDataType Structure

Name	Type	ValueRank	Description
MagazineNumber	Byte	−1 (−1 = Scalar)	Identifier of the magazine
PocketNumber	UInt32	−1 (−1 = Scalar)	The pocket within the magazine

8

Namespaces

8.1 Namespace Metadata

Table 200 and Table 201 define the namespace metadata for this specification. 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 UA Part 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 UA Part 5.

Attributes

Table 200: OPC UA NC Server Core NamespaceMetadata Object Attributes

Attribute	Value
BrowseName	http://heidenhain.de/NC/

References

Table 201: OPC UA NC Server Core NamespaceMetadata Object References

References	BrowseName	DataType	Value
HasProperty	NamespaceUri	String	http://heidenhain.de/NC/
HasProperty	NamespaceVersion	String	1.5.0
HasProperty	NamespacePublicationDate	DateTime	2023-06-30
HasProperty	IsNamespaceSubset	Boolean	False
HasProperty	StaticNodeIdTypes	IdType[]	{Numeric}
HasProperty	StaticNumericNodeIdRange	NumericRange[]	Null
HasProperty	StaticStringNodeIdPattern	String	Null

NamespaceVersion

Note that the version number schema of the information model (document) omits the last number as given in *NamespaceVersion* (see "About this Document", Page 8). The third number of the *NamespaceVersion* can be used by the OPC UA NC Server to indicate small changes like error corrections between information model version releases adding new contents.

IsNamespaceSubset

Note that the *IsNamespaceSubset property* is set to *False* if the UaNodeSet XML file as well as the Server's address space contain the complete namespace. Servers only exposing a subset of the namespace need to change the value to *True*.

8.2 Handling of OPC UA Namespaces

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

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

Table 202 provides a list of mandatory and optional namespaces used in an OPC UA *Server* implementing the Core Information Model.

Table 202: Namespaces used in an OPC UA Server implementing the Core Information Model

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 namespace shall have namespace index 1.	Mandatory
http://heidenhain.de/NC/	Namespace for *NodeIds* and *BrowseNames* defined in this specification. The namespace index is Server-specific.	Mandatory
Machine-specific extensions	Namespace(s) of machine-specific extensions defined by the machine manufacturer. The namespace indices are Server--specific.	Optional

Table 203 provides a list of namespaces and their indices used for *BrowseNames* in this specification. The default namespace of this specification is not listed since all *BrowseNames* without prefix use this default namespace.

Table 203: Namespaces used in this specification

NamespaceURI	Namespace Index	Example
http://opcfoundation.org/UA/	0	0:EngineeringUnits

9

Machine File System Access

9.1 Introduction

This chapter unites information about access to the machine's file system using the functionality provided by the OPC UA NC Server. Below, a general introduction states how the file system access is integrated into the different types of the Core Information Model and how access rights are handled.

9.2 "FileDirectoryType" and 9.3 "FileType" provide general information about the OPC UA standard types *FileType* and *FileDirectoryType* and additional detailed information regarding their application within the OPC UA NC Server.

Especially for OPC UA client application developers, useful hints and important warnings regarding different aspects of file system access and file transfer to and from the machine are given in 9.4 "Warnings and Important Hints".

File System Component

Following the definition of "File Transfer" in OPC UA Part 20, the entry point to the file system representation is a *FileDirectoryType* instance node with the *BrowseName FileSystem*. The *FileSystem* is an optional component of the *MachineType*.

Every directory of the machine's file system is represented with a node of type *FileDirectoryType*. Beginning with the file system partition nodes (e.g., *TNC*) below the *FileSystem* component of the *MachineType*, every *FileDirectory Organizes* the containing files and directories.

Within this chapter the term File System is used to describe all the nodes in the hierarchy below the *Machine's FileSystem* component.

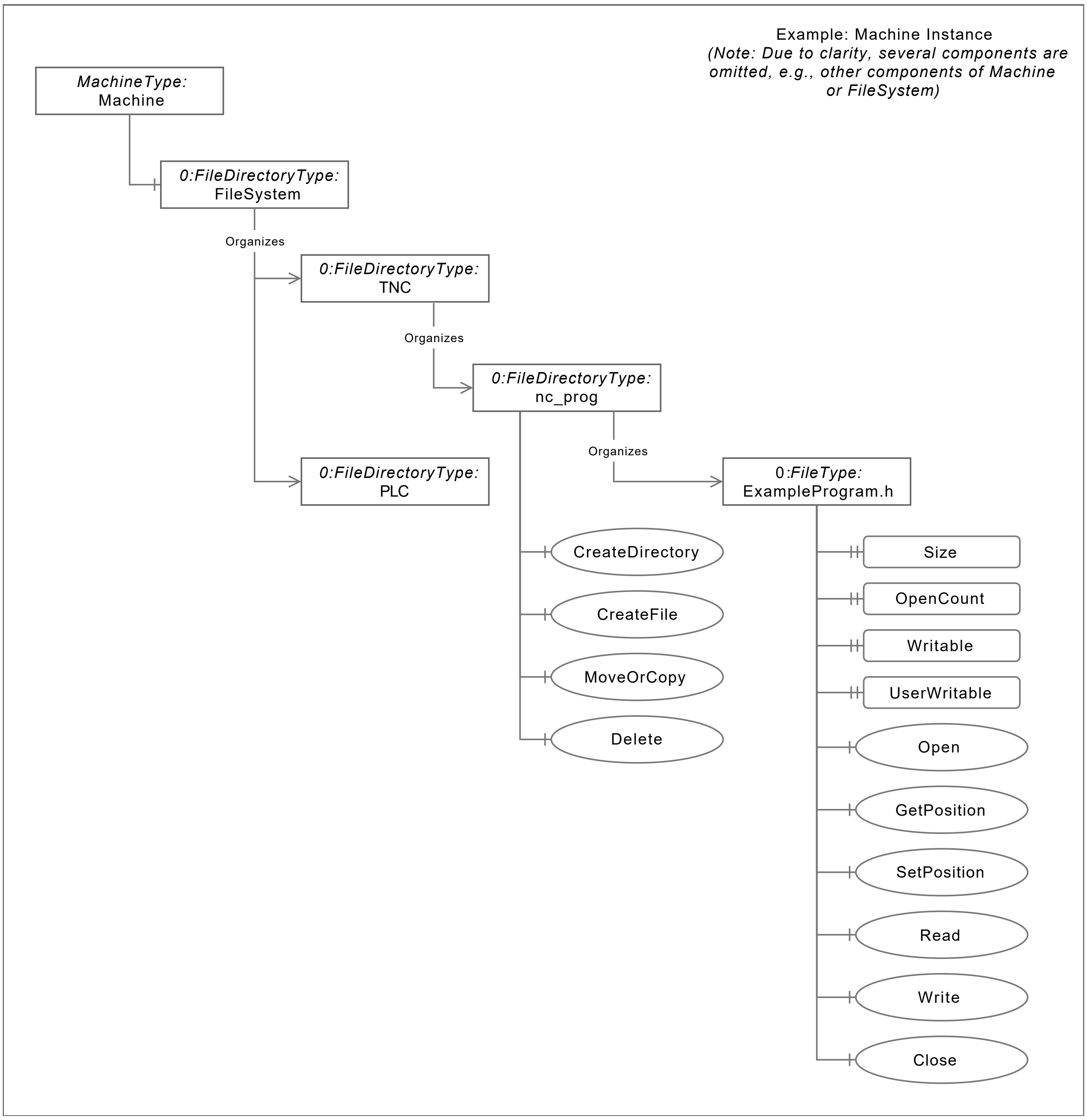
Example: Machine Instance
(Note: Due to clarity, several components are omitted, e.g., other components of Machine or FileSystem)
MachineType: Machine
0:FileDirectoryType: FileSystem
Organizes
0:FileDirectoryType: TNC
Organizes
0:FileDirectoryType: nc_prog
Organizes
0:FileDirectoryType: PLC
0:FileType: ExampleProgram.h
CreateDirectory
CreateFile
MoveOrCopy
Delete
Size
OpenCount
Writable
UserWritable
Open
GetPosition
SetPosition
Read
Write
Close

Figure 19: Shortened example of a *Machine* instance with *FileSystem*

The *FileType* and *FileDirectoryType* with their methods and properties are defined in OPC UA Part 20 and additionally described in 9.3 "FileType" and 9.2 "FileDirectoryType".

Note that it is not possible to directly use the methods of the *FileSystem* node, since the first level of the exposed file system (TNC and PLC partitions) cannot be changed. *FileDirectoryType* methods can be used starting with the *TNC* and *PLC* nodes (see 9.2 "FileDirectoryType").

File System Access Rights

Access to the different parts of the File System is restricted. To access *Files* and *FileDirectories* on the *PLC* partition, for example, an OPC UA client user needs the corresponding right (i.e. the right HEROS.FileOEM to access to the File System *PLC* partition).

Access rights are managed by the machine's operating system user administration. HEIDENHAIN control-specific information about access-right handling can be found in the respective user manuals.
Further information: Setup, Testing and Running NC Programs User’s Manual, or Setup and Program Run User’s Manual

In general the rights of the OPC UA client user to create, delete, move, copy, read and write a *File* correspond to the machine's operating system user rights. Note that write access to some *Files* is additionally restricted as described in Protected and Locked Files in 9.4 "Warnings and Important Hints".

Relations between File System and Machining Channel

The *ChannelType* and its (sub)components like the *Program's ExecutionState* (of type *NCProgramStateMachineType*) have methods like *SelectProgram* which need the string representation of a file system path as *InputArgument*. These types are extended with optional methods accepting the *NodeId* of a *File*. For example, the method *SelectProgramByNodeId* can be used instead with the *NodeId* of the *File* representing the NC program that should be selected for execution.

When an NC program is selected as *Program* of a *Channel*, the optional *FileNodeId* property of the *Program's Name* variable contains the *NodeId* of the *File* corresponding to the NC program. Using this *NodeId* the NC program *File* can, for example, be downloaded.

Additional information can be found at the respective type definitions 4.12 "ChannelType", 4.14 "ProgramType" and 4.15 "NCProgramStateMachineType".

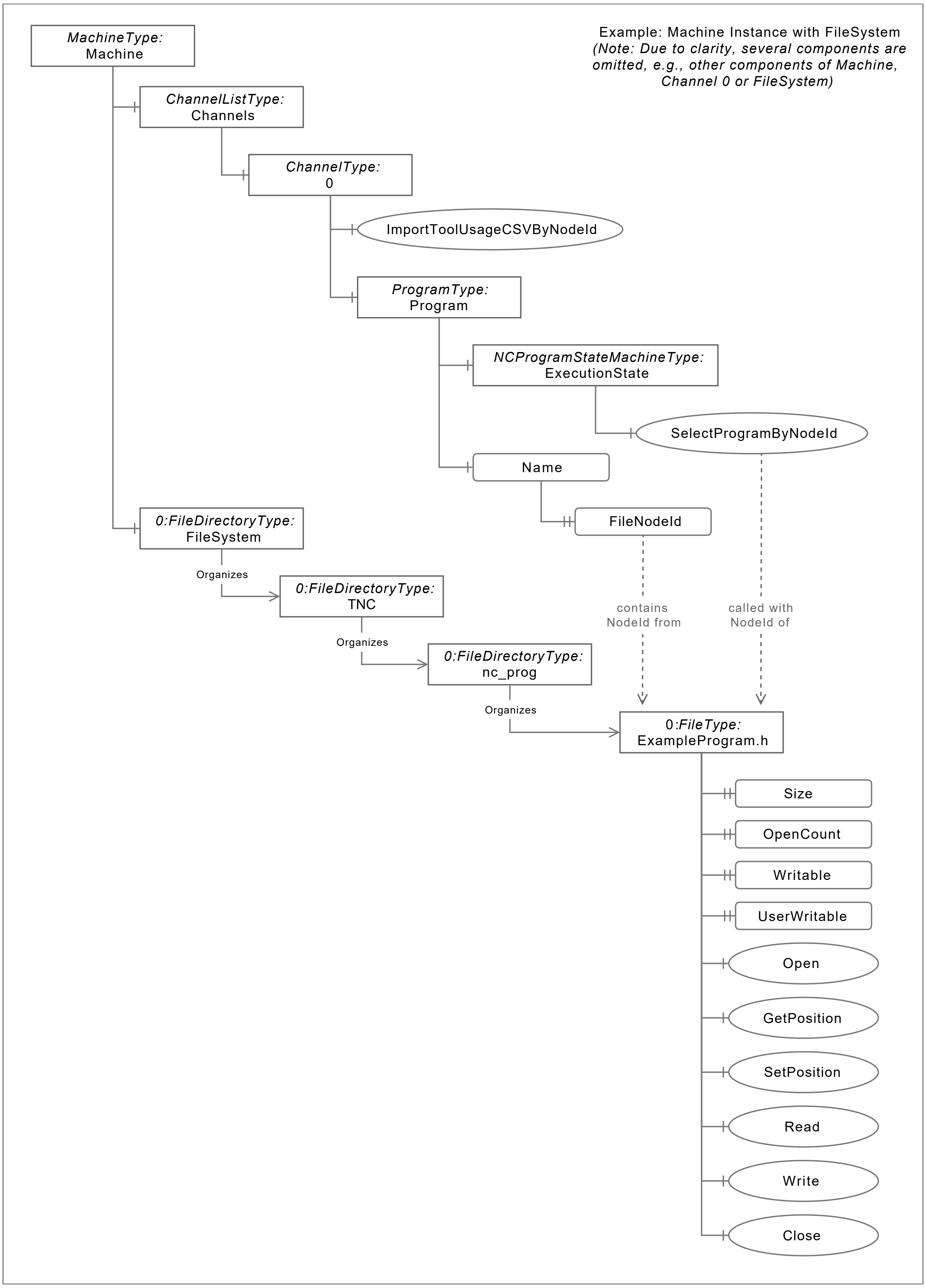
Example: Machine Instance with FileSystem
(Note: Due to clarity, several components are omitted, e.g., other components of Machine, Channel 0 or FileSystem)
MachineType: Machine
ChannelListType: Channels
ChannelType: 0
ImportToolUsageCSVByNodeId
ProgramType: Program
NCProgramStateMachineType: ExecutionState
SelectProgramByNodeId
Name
FileNodeId
0:FileDirectoryType: FileSystem
Organizes
0:FileDirectoryType: TNC
Organizes
0:FileDirectoryType: nc_prog
Organizes
contains NodeId from
called with NodeId of
0:FileType: ExampleProgram.h
Size
OpenCount
Writable
UserWritable
Open
GetPosition
SetPosition
Read
Write
Close

Figure 20: Shortened example of a *Machine* instance with the relation between machining channel-related types and *FileSystem*

9.2 FileDirectoryType

Every directory within the machine's file system is represented with a node of type *FileDirectoryType*. A *FileDirectory* provides methods to create, delete, copy and move *Files* and *FileDirectories*, and is defined in OPC UA Part 20.

FileDirectoryType Methods

Each *FileDirectory* has several methods to execute actions on *Organized Files* and *FileDirectories* below it:

- *CreateDirectory*: create a new *FileDirectory* below this node with the given name
- *CreateFile*: create a new *File* below this node with the given name (including the file type suffix)
- *Delete*: delete the specified existing *File* or *FileDirectory* node below this node
- *MoveOrCopy*: move or copy a *File* or *FileDirectory* below this node to another location (below another *FileDirectory* node)

FileDirectory methods can only be used for files or directories directly *Organized* by them. For example, if the *File ExampleProgram.h* in Image 19 should be deleted, the *Delete* method of the *nc_prog FileDirectory* has to be used.

An exception is the *Delete* method. If a directory is deleted, also all containing files and directories are deleted automatically. Note that a good return code is only given if the deletion of all contents and the directory itself was successful. If, for example, one of the contained files could not be deleted, the return codes in Table 206 apply. But all other files and directories that are also in the directory are deleted.

FileDirectoryType method return codes are defined in general in OPC UA Part 4 and in more detail in OPC UA Part 20. The tables below wrap up general information and provide additional OPC UA NC Server specific information about the different situations.

Note that versions of the OPC UA NC Server before NC software version 18 do not support *MoveOrCopy* for *FileDirectories* in general. The OPC UA NC Server as of NC software version 18 supports *MoveOrCopy* only for empty *FileDirectories*. It is possible to *MoveOrCopy Files*.

Table 204: CreateDirectory Result Codes

Result Code	Description
BadUserAccessDenied	The user does not have the right to create a directory at this location.
BadBrowseNameDuplicat-ed	A directory entry (file or directory) with the given name already exists.
BadInternalError	An internal error or erroneous situation inhibits creation of the new directory at this location.
BadNotWritable	The file system that would contain the new directory is read-only.
BadNotFound	The directory itself or another directory on the path does not exist (anymore).
BadResourceUnavailable	A directory cannot be created due to file system resource limitations.
BadNotSupported	A directory cannot be created due to technical file system limitations (e.g., if the resulting path would exceed the maximum file system path length).
BadOutOfMemory	Insufficient memory to complete the operation.

Table 205: CreateFile Result Codes

Result Code	Description
BadUserAccessDenied	The user does not have the right to create a file at this location. Note the special cases for protected files (see Protected and Locked Files in 9.4 "Warnings and Important Hints"). This code is also returned in case the resulting file would be protected with the given name at the given location.
BadBrowseNameDuplicat-ed	A directory entry (file or directory) with the given name already exists within this directory.
BadInternalError	An internal error or erroneous situation inhibits creation of the new directory at this location.
BadNotFound	The directory itself or another directory on the path does not exist (anymore).

Result Code	Description
BadNotWritable	The file system that would contain the new directory is read-only.
BadResourceUnavailable	A file cannot be created due to file system resource limitations (e.g., the maximum number of open files has been reached).
BadNotSupported	A directory cannot be created due to technical file system limitations (e.g., if the resulting path would exceed the maximum file system path length).
BadOutOfMemory	Insufficient memory to complete the operation.
BadRequestInterrupted	The call was interrupted, e.g, due to waiting for the response of a slow device.
Uncertain	The internal file descriptor to the new file was invalid, but the file may have been created.

If *CreateFile* is called with *RequestFileOpen* = true, the return codes of the *FileType* method *Open* also apply

Table 206: Delete Result Codes

Result Code	Description
BadUserAccessDenied	The user does not have the right to delete this file or directory (or an entry below the directory). Note the special cases for protected files (see Protected and Locked Files in 9.4 "Warnings and Important Hints").
BadInvalidArgument	The element to be deleted is not a file or directory or not directly *Organized* by this directory node.
BadInvalidState	The file to be deleted is open at the moment (OpenCount > 0), or locked by the control (see Protected and Locked Files in 9.4 "Warnings and Important Hints"). A file within the directory to be deleted is locked by the control.
BadNotFound	The directory or file to be deleted or another directory on the path to it does not exist (anymore).
BadInternalError	An internal error or erroneous situation inhibits the deletion of the file or directory.
BadNotSupported	A file or directory cannot be deleted due to technical file system limitations.
BadResourceUnavailable	A file or directory cannot be deleted due to file system resource limitations at the moment.
BadOutOfMemory	Insufficient memory to complete the operation.
BadNotWritable	The file system containing the file or directory is read-only.

Table 207: MoveOrCopy Result Codes

Result Code	Description
BadUserAccessDenied	The user does not have the right to move or copy this file. (Reasons might be insufficient rights on the source file and/or the target directory for example.) Note the special cases for protected files (see Protected and Locked Files in 9.4 "Warnings and Important Hints").
BadNotSupported	A file cannot be created due to technical file system limitations (e.g., if the resulting path would exceed the maximum file system path length). Note that the OPC UA NC Server before NC software version 18 does not support moving or copying directories in general. Later versions of the server allow moving or copying of empty directories.
BadInvalidArgument	The element to be moved or copied is not a file or not directly *Organized* by this directory node.
BadInvalidState	The file to be moved is locked by the control (see Protected and Locked Files in 9.4 "Warnings and Important Hints").

Result Code	Description
BadBrowseNameDuplicat-ed	A file (or directory) with the given name already exists within the target directory.
BadNotFound	The file to be moved or copied, the target directory, or another node on the path to it does not exist (anymore).
BadInternalError	An internal error or erroneous situation inhibits copying or moving the file.
BadResourceUnavailable	The file cannot be moved or copied due to file system resource limitations.
BadOutOfMemory	Insufficient memory to complete the operation.
BadNotWritable	The file system containing the file or target directory is read-only.

9.3 FileType

Every file within the machine's file system is represented with a node of type *FileType*. A *File* has several properties and methods to transfer a file to or from the machine and is defined in OPC UA Part 20.

FileType Properties

Some of the following *File* properties have an OPC UA NC Server specific adaptation:

- *Size* contains the size of the *File* in bytes.
- The optional property *MimeType* is not supported and thus is not present at *File* instances in the address space of the OPC UA NC Server.
- *Writable* indicates whether the *File* is writable in general, i.e., independent from concrete OPC UA client user rights.
- *UserWritable* indicates whether a *File* can be written by the current user, i.e. if the authenticated OPC UA client user has the right to write to the file. If the NC software is running, several files that are important for the NC software are protected from modifying access via OPC UA. The *UserWritable* property of a *File* is set to false in this case. See Protected and Locked Files in 9.4 "Warnings and Important Hints" for more information.
- *OpenCount* counts only file handles of OPC UA client applications connected to the OPC UA NC Server. If a file is opened on the machine by another UI application or remote via another interface, this is not reflected in the *OpenCount*.

FileType Methods

File transfer can be realized using the methods of the *FileType*:

- *Open*: open a *File* with the given mode (e.g., for writing) and get the resulting file handle. The file handle is the reference for all following operations, like reading or writing.
- *SetPosition*: sets the given file handle to the given position within the file
- *GetPosition*: returns the current position of the given file handle
- *Read*: returns the data of the file beginning at the given file handle's current position and ending after the given length. The current position of the file handle is automatically shifted according to the given length (bytes that were read).
- *Write*: writes the given data at the given file handle's current position. The current position of the file handle is automatically shifted according to the given data (bytes that were written).
- *Close*: close the *File* using the given file handle

For a detailed description of the methods with their parameters, see OPC UA Part 20.

FileType method return codes are defined in general in OPC UA Part 4 and in more detail in OPC UA Part 20. The tables below wrap up general information and provide additional OPC UA NC Server specific information about the different situations.

Table 208: Open Result Codes

Result Code	Description
BadUserAccessDenied	The user does not have the right to open this file (for writing). Note the special cases for protected files (see Protected and Locked Files in 9.4 "Warnings and Important Hints").
BadInvalidState	The file was already open when trying to open it for *Write*. The file is locked and thus not writable.
BadNotReadable	The file was already open for *Write* when trying to open it for *Read*. The file to be opened for *Read* is not readable.
BadNotWritable	The file to be opened for *Write* is not writable or the file system containing the file is read-only.
BadNotFound	The file to be opened or another node on the path to it does not exist (anymore).
BadInternalError	An internal error or erroneous situation inhibits opening of the file.
BadResourceUnavailable	The file cannot be opened due to file system resource limitations.
BadNotSupported	A file cannot be opened due to technical file system limitations.

Result Code	Description
BadOutOfMemory	Insufficient memory to complete the operation.
BadRequestInterrupted	The call was interrupted (e.g., due to waiting for the response of a slow device).

Table 209: Close Result Codes

Result Code	Description
BadInvalidArgument	The given *FileHandle* is invalid.
BadInvalidState	The file to be closed is locked by the control at the moment (see Protected and Locked Files in 9.4 "Warnings and Important Hints").
BadNotFound	The file to be closed or one node on the path to it does not exist (anymore).
BadInternalError	An internal error or erroneous situation inhibits closing of the file.
BadResourceUnavailable	The file cannot be closed due to file system resource limitations.
BadNotSupported	A file cannot be closed due to technical file system limitations.
BadUserAccessDenied	The user does not have the right to close this file. Note the special cases for protected files (see Protected and Locked Files in 9.4 "Warnings and Important Hints").
BadOutOfMemory	Insufficient memory to complete the operation.

Table 210: Read Result Codes

Result Code	Description
BadInvalidArgument	The given *FileHandle* is invalid or the given length is not positive.
BadInvalidState	The file was not opened for read access.

Table 211: Write Result Codes

Result Code	Description
BadInvalidArgument	The given *FileHandle* is invalid.
BadInvalidState	The file was not opened for write access.
BadNotWritable	The file might be locked and thus not writable.

Table 212: GetPosition Result Codes

Result Code	Description
BadInvalidArgument	The given *FileHandle* is invalid.

Table 213: SetPosition Result Codes

Result Code	Description
BadInvalidArgument	The given *FileHandle* is invalid.

9.4 Warnings and Important Hints

Within this chapter the term *File System* is used to describe all the nodes in the hierarchy below the *Machine's FileSystem* component.

Dynamic File System AddressSpace

The File System represents the file system of the machine. The machine's file system is not static (e.g., new files and directories are created, deleted or moved on disk). The File System part of the AddressSpace is updated during runtime of the OPC UA NC Server to reflect these changes. This means that *File* and *FileDirectory* nodes are created and deleted during runtime of the OPC UA NC Server.

Note that the OPC UA NC Server does not support reporting of model changes in the File System using events. This means that no *BaseModelChangeEvents* are sent to publish changes to the File System.

Browsing a *FileDirectory* returns the references to all current *Organized Files* and *FileDirectories*.

Nodes that correspond to files or directories that have been deleted on disk are also deleted from the AddressSpace. The OPC UA NC Server answers requests using their *NodeIds* accordingly. If, for example, the *NodeId* of a node that corresponds to a *File* (e.g., the *NodeId* of its *Size* property) that has been deleted since the retrieval of the *NodeId* is used within a request (e.g., to read the value), the request is answered with *BadNodeIdUnknown.*

Due to the size and dynamicity of the file system, unnecessary recursive browsing of the full File System is strongly recommended against!

NodeIds of File System Nodes

The numeric *NodeIds* of nodes inside the File System are dynamically assigned during runtime of the OPC UA NC Server. As long as a *File* or a *FileDirectory* stays unchanged (for example, not renamed) at the same location, the *NodeIds* are not changed while the OPC UA NC Server is running.

When the OPC UA NC Server restarts, the numeric *NodeIds* are newly assigned. This means that the *NodeIds* of *Files* and *FileDirectories* are not persistent over OPC UA NC Server lifecycles.

It is recommended to use the *Browse* or *TranslateBrowsePathsToNodeIds* requests to get the *NodeIds* of *Files* and *FileDirectories* (especially after an OPC UA NC Server restart or new connection to the Server). The File System root *FileSystem* can be used as start-node of the *BrowsePath* in a *TranslateBrowsePathsToNodeIds* request.

An exception is the handling of the method nodes like *CreateFile*. To simplify the usage of the OPC UA standard methods for file transfer, every *File* or *FileDirectory* does not have its own instances of the methods, but instead refers to the method nodes in the type tree. So File System methods can be called using their *NodeIds* defined in the OPC UA Specification (respectively their NodeSet.xml or NodeIds.csv files).

Protected and Locked Files

Some files that are accessible within the machine's file system are highly important for a functional control and machine. Changes to these files or format violations can cause major problems, up to the machine stopping the production process or NC software start-up issues.

Therefore, access to the files listed in **CfgTablePath** (no. 102500) and **CfgConfigDataFiles\dataFiles** (no. 106303) is protected while the NC software is running, by restricting access to read-only access (*UserWritable* is *False*). See also "File System Access without Running NC Software"

Along with explicitly protected files, some files are locked by the NC software while they are in use, such as a selected and running NC program. (The *Close* method will fail with the status code *BadInvalidState* in case the file is protected by the NC software at that moment.)

File System Access without Running NC Software

Since the OPC UA NC Server runs independently from the NC software, it is in general also possible to access the file system and use the related functionality while the NC software is not running. This enables OPC UA client applications, for example, to download control service files from the machine also while the NC software is not running. (Note that "running" within this context does not mean that an NC program is being executed on the machine, but rather only that the NC software is running.)

This state is indicated by the *Machine's State CurrentState* being *NCIsNotConnected*. During this state not all NC software file protection mechanisms are in place. So it is possible to harm files that are important for a running NC software and control!

> Manipulating actions on the File System while the NC software is not running can harm the functionality of the control.
>
> Only manipulate files while the NC software is not running if you are sure that it will not cause issues regarding the functionality of the NC software. Along with that, actions on the file system should not be executed during the start-up or shut-down process of the NC software. The start-up and initialization of the NC software is finished when the *State* (*CurrentState*) of the *Machine* is *NCIsAvailable* (see 3.4 "NC State Machine").

The possibility to access the machine's file system while the NC software is not running is new compared to the functionality provided by HEIDENHAIN DNC (see the Connected Machining brochure).

10

Extensions of the Machine Manufacturer

10.1 Introduction

Starting with version 1.02 of the Core Information Model, a Machine instance of 4.1 "MachineType" can provide machine-specific extensions at the *ManufacturerExtensions* component.

These extensions are configured by the machine manufacturer. So they differ between manufacturers and can also be different from machine to machine.

Machine manufacturers can extend the OPC UA NC Server with access to data of additional sensors, machine subsystems, or values from PLC programs. Along with providing only the raw data, they can also provide additional information like units of measure or value ranges.

This chapter describes the concepts with an example, contains hints for OPC UA client application developers, and lists 10.3 "Information for Machine Manufacturers".

Within this chapter, the terms machine-specific nodes and Manufacturer Extensions describe all the nodes in the hierarchy below the *Machine's ManufacturerExtensions* component.

Initialization of Manufacturer Extensions

The OPC UA NC Server creates the machine-specific nodes once after the PLC program has been translated.

When the *Machine's State CurrentState* changes the first time to *NCIsAvailable* the OPC UA NC Server creates the machine-specific nodes. During a server life cycle, the created objects and variables do not change. Changes of the machine manufacturer like new nodes, name changes, or node deletions are applied after a server restart.

Note that the OPC UA NC Server does not support reporting of model changes for Manufacturer Extensions using events. This means that no *BaseModelChangeEvents* are sent to publish changes to machine-specific nodes.

Namespace and NodeIds of Machine-Specific Nodes

Machine-specific nodes are located within additional OPC UA namespaces. They are not part of the server's namespace with index 1. Machine manufacturers define at least one own namespace to describe the information and data they provide. The namespaces are added to the *Server's NamespaceArray* when the Manufacturer Extensions are initialized. More information about the namespaces like the version or the publication date can be found at the corresponding *NamespaceMetadataType* instance at the *Server's Namespaces* node.

Machine manufacturers can configure *NodeIds* for machine-specific objects and variables using String or Numeric *Identifiers*. If a manufacturer does not explicitly configure an *Identifier*, a Numeric *Identifier* is automatically generated by the OPC UA NC Server. *NodeIds* of the *NamespaceMetadataType* instance nodes per namespace and properties of data variables are always automatically assigned by the OPC UA NC Server.

As far as possible the OPC UA NC Server keeps the *NodeIds* of machine-specific nodes stable after server restarts. But in particular changes to the configuration of the Manufacturer Extensions can cause changes of *NodeIds*.

10.2 Manufacturer Extensions

Manufacturer Extension nodes are located in the address space hierarchy below the optional *ManufacturerExtensions* component of a *Machine* instance. An example of machine-specific nodes is shown in Image 21.

The *ManufacturerExtensions* node *Organizes* the machine-specific objects. These objects are always instances of *BaseObjectType*. Nested structures are possible, meaning an object can also *Organize* more machine-specific objects. Machine-specific variables are normally components of a machine-specific object. (But it is also possible that they are directly referred by the *ManufacturerExtensions* node.)

Depending on the type and the configuration by the machine manufacturer, the variables can have several properties. The properties provide additional information about the data value of the variable, for example, the *EngineeringUnits*. Properties defined in the OPC UA Standard are used to provide this metadata.

VariableTypes with DataTypes and Metadata Properties

Machine-specific variables can be instances of *DataItemType* or *BaseAnalogType* (OPC UA Part 8) with their specified optional properties as listed in Table 214. If the Server cannot determine the data type of a value or cannot access the data, the *DataType* is *BaseDataType* and the *Value* has a corresponding bad *StatusCode*.

The variables have scalar data values, so the *ValueRank* is always -1 (Scalar).

Table 214: VariableTypes with DataTypes and Properties used for machine-specific nodes

VariableType	DataType	Possible Properties
DataItemType	Boolean	-
	String	MaxStringLength
BaseAnalogType	SByte	EngineeringUnits EURange InstrumentRange
	Int16	
	Int32	
	Double	EngineeringUnits EURange InstrumentRange ValuePrecision

The *MaxStringLength* property provides the maximum number of bytes of the String value.

If an *InstrumentRange* is given, the OPC UA NC Server allows only write requests with values within the defined range.

If a *ValuePrecision* is given, it specifies the supported number of digits after the decimal point (OPC UA Part 8). This property is provided if the corresponding internal value is of an Integer data type. If the variable is writable, providing the valid value ranges using the corresponding properties is recommended to machine manufacturers.

User Access Rights

Every authenticated user can browse the Manufacturer Extensions and read most of the node attributes like the *BrowseName, Description* or the *DataType* of variables. Read- or write-access to the *Value* attribute of data variables is restricted to users with the corresponding user rights at the control. The *Values* of properties like the *EngineeringUnits* of a data variable can be read by every authenticated user.

The read and write permissions of the authenticated user, i.e. access rights to a specific data value, are exposed at the *UserRolePermissions* attribute of the variable.

Note that even if a user has write-access to a variable the write request can be denied by a subsystem (e.g., because it is not allowed in the current situation).

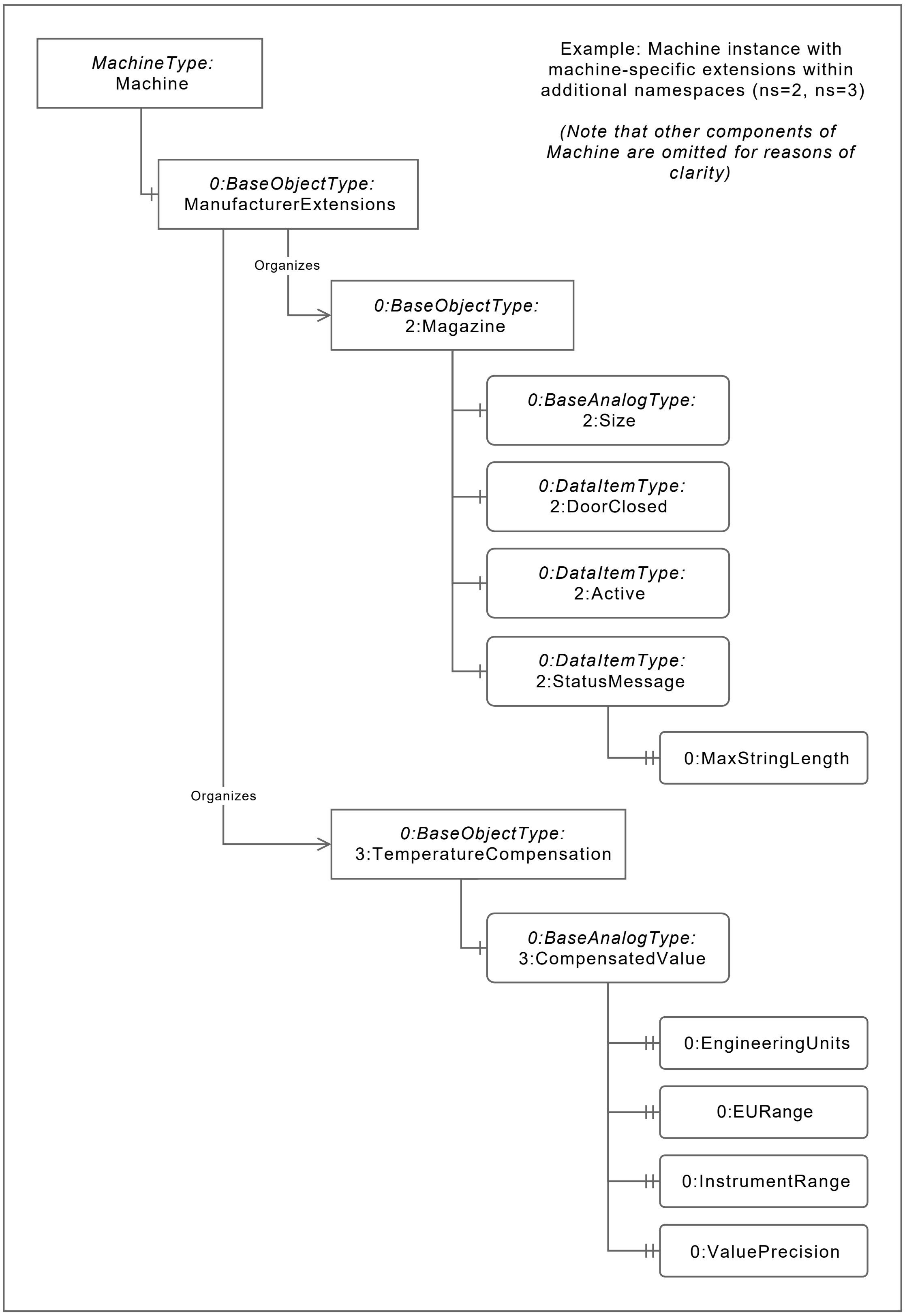
Example: Machine instance with machine-specific extensions within additional namespaces (ns=2, ns=3)
(Note that other components of Machine are omitted for reasons of clarity)
MachineType: Machine
0:BaseObjectType: ManufacturerExtensions
Organizes
0:BaseObjectType: 2:Magazine
0:BaseAnalogType: 2:Size
0:DataItemType: 2:DoorClosed
0:DataItemType: 2:Active
0:DataItemType: 2:StatusMessage
0:MaxStringLength
Organizes
0:BaseObjectType: 3:TemperatureCompensation
0:BaseAnalogType: 3:CompensatedValue
0:EngineeringUnits
0:EURange
0:InstrumentRange
0:ValuePrecision

Figure 21: Shortened example of a *Machine* instance with *ManufacturerExtensions*

10.3 Information for Machine Manufacturers

Configuration of Machine-Specific Extensions

Machine-specific extensions of the OPC UA NC Server are configured using the three machine configuration parameters **CfgOpcUaNamespace** (no. 133900), **CfgOpcUaObject** (no. 134000) and **CfgOpcUaPlcVar** (no. 134100). Detailed information on how to configure machine-specific namespaces and nodes is given in the parameter descriptions and the Technical Manual of the specific control model. Additionally the OPC UA NC Server validates the configuration and reports found issues as machine events.

To access the data values of the configured variables, the OPC UA client application user needs the corresponding user rights at the control. For information about the access rights, roles, and their management, refer to the Technical Manual of the specific control model.

HEIDENHAIN PLC Basic Program Example

Starting with version 16, the PLC Basic Program contains an example configuration of machine-specific extensions of the OPC UA NC Server.

On HEIDENHAIN programming stations the example is also integrated starting with the corresponding NC software versions; see "Associated HEIDENHAIN CNC Controls", Page 15.

11

Lists

11.1 List of Tables

11.2 List of Figures

Annex

Annex A: Tool Data Reference

HEIDENHAIN Tool Data and Tool Types

Table 215 lists the tool data items defined by HEIDENHAIN as of NC software version 18. The tool types including the unique identifiers are listed in Table 216.

The OPC UA NC Server in NC software version 18 does not support tool-type-specific data items for grinding and dressing tools yet.

Depending on the control model and version, different data items and tool types are made available by the OPC UA NC Server.

More information about each data item is given in the Technical Manual of the respective control as well as its Setup, Testing and Running NC Programs User's Manual or Setup and Program Run User's Manual.

Regardless from the current configuration of the machine, all tool data item values are provided and interpreted based on the metric system.

Table 215: HEIDENHAIN Tool Data Items

Category	OPC UA Identifier	Internal Table[1)]	Internal Column
Identification	ToolNumber	T, TRN, GRD, DRS[2)]	T
	ToolIndex		
	Type[3)]	T	TYP
		TRN	TYPE
		GRD	TYPE
		DRS	TYPE
	Name[4)]	T	NAME
		TRN	NAME
		GRD	NAME
		DRS	NAME
		TCH	TNAME
	DatabaseId	T	DB_ID
	SerialNumber	TP	SERIAL
	Comment[4)]	T	DOC
Geometry	Length[4)]	T	L
	LengthOffset[4)]	T	L-OFFS
	Radius[4)]	T	R
	RadiusOffset[4)]	T	R-OFFS
	CutterEdgeRadius	T	R2
	NumberOfCutterEdges	T	CUT
	CutterEdgeLength	T	LCUTS
	UsableLength	T	LU
	NeckRadius	T	RN
	MaximumPlungeAngle	T	ANGLE
	PointAngle	T	T-ANGLE
	ThreadPitch	T	PITCH
	RadiusAtTip	T	R_TIP
	CarrierKinematics[4)7)]	T	KINEMATIC

Category	OPC UA Identifier	Internal Table[1)]	Internal Column
	ToolShape [4)7)]	T	TSHAPE
	LengthX	TRN	XL
	LengthY	TRN	YL
	LengthZ	TRN	ZL
	CutterRadius	TRN	RS
	ToolOrientation	TRN	TO
	SpindleOrientation	TRN	ORI
	BAngleOffset	TRN	SPB-INSERT
	CutterEdgeAngle	TRN	T-ANGLE
	CutterEdgePointAngle	TRN	P-ANGLE
	CutterEdgeWidth	TRN	CUTWIDTH
	CutterEdgeSecondaryWidth	TRN	CUTLENGTH
Correction	LengthOversize	T	DL
	RadiusOversize	T	DR
	EdgeRadiusOversize	T	DR2
	EdgeRadiusCompensationTable	T	DR2TABLE
	LengthXOversize	TRN	DXL
	LengthYOversize	TRN	DYL
	LengthZOversize	TRN	DZL
	CutterEdgeRadiusOffset	TRN	DRS
	CutterEdgeWidthOffset	TRN	DCW
	WorkpieceDiameterCorrection[5)4)]	TRN	WPL-DX-DIAM
	WorkpieceLengthCorrection[5)4)]	TRN	WPL-DZL
ToolLife	LockedStatus	T	TL
	ReplacementToolNumber	T	RT
	MaximumLifetime	T	TIME1
	MaximumLifetimeToolCall	T	TIME2
	AllowedOvertime	T	OVRTIME
	CurrentLifetime	T	CUR_TIME
	LastUsage	T	LAST_USE
ToolWear	LengthTolerance	T	LTOL
	RadiusTolerance	T	RTOL
	EdgeRadiusTolerance	T	R2TOL
	LengthBreakageTolerance	T	LBREAK
	RadiusBreakageTolerance	T	RBREAK
Technology	MaximumSpeed	T	NMAX
	Liftoff	T	LIFTOFF
	ActiveChatterControl	T	ACC
	ActiveFeedControlStrategy	T	AFC

Category	OPC UA Identifier	Internal Table[1)]	Internal Column
	AfcReferencePower	T	AFC-LOAD
	AfcOverloadWarning	T	AFC-OVLD1
	AfcOverloadSwitchoff	T	AFC-OVLD2
	FrontfaceCutterWidth	T	RCUTS
	CuttingDirection	T	DIRECT
	ToolEdgeMaterial	T	TMAT
	CuttingData	T	CUTDATA
Touchprobe	TouchprobeModel	TP	TYPE
	TouchprobeRecord[6)]	TP	NO
		T	TP_NO
	StylusShape	TP	STYLUS
	ProbingFeedRate	TP	F
	SelectionPrePositioningMaxFeed	TP	F_PREPOS
	PrePositioningMaxFeedRate	TP	FMAX
	MaximumProbingRange	TP	DIST
	PrePositioningSafetyDistance	TP	SET_UP
	SelectionCollisionReaction	TP	REACTION
	TrackingCalibrationAngleActive	TP	TRACK
	CalibrationSpindleAngle	TP	CAL_ANG
	CalibrationPrincipalAxisOffset	TP	CAL_OF1
	CalibrationSecondaryAxisOffset	TP	CAL_OF2
Manufacturer	AttributeForPocket[4)]	T	PTYP
	PlcStatus	T	PLC

1) Tool and pocket table identifiers (with default source file name)
   - T: Tool table tool.t
   - TP: Touch probe table touchprobe.tp
   - TRN: Turning tool table toolturn.trn
   - GRD: Grinding tool table toolgrind.grd
   - DRS: Dressing tool table tooldress.drs
   - TCH: Pocket table tool_p.tch

2) The T column is the primary key of the tables. It links the record in table T to a record with tool type specific data in the other tables.

3) Some of the tool types are internally defined by a main tool type (the category) and a specific subtype information. The *Type* combines the information so that one value identifies the type of a tool uniquely. The unique tool type identifiers are listed in Table 216.

4) The definition and metadata of the data item can be edited by the machine manufacturer. It is done by changes of machine parameter values in the machine's configuration.
The unit, maximum length, allowed characters, minimum, maximum, and default value can differ from the default values configured by HEIDENHAIN.
The OPC UA NC Server provides the values based on the current configuration of the machine. (If the configuration is changed during runtime of the server and after the first instantiation of the nodes, for example, during commissioning of the machine, the server must be restarted to apply the changes.)
In general the machine manufacturer has the possibility to replace parts of the machine's configuration and so to

change the definition of all tool data items. Machine manufacturers can use this additional technique, but must be careful to avoid potentially incompatible changes.

5) Can be enabled by the machine manufacturer at the configuration of the machine.

6) Informational, read-only; automatically handled internally.

7) Tool data items referring to 3D model files; represented by instances of 4.19 "ToolDataItemType" with *FileLocation* component and support of 3D model file validation (see "Validation 3DModels", Page 108)

Table 216: HEIDENHAIN Tool Types

Category	OPC UA Identifier	Description
MillingTools	MILL_MILL	Milling cutter
	MILL_ROUGH	Roughing cutter
	MILL_FINISH	Finishing cutter
	MILL_FACE	Face milling cutter
	MILL_BALL	Ball-nose cutter
	MILL_TORUS	Toroid cutter
	MILL_CHAMFER	Chamfer mill
	MILL_SIDE	Side milling cutter
	MILL_THREAD	Thread mill
	MILL_THREAD_CSINK	Thread mill with countersink
	MILL_THREAD_SINGLE	Single form thread mill
	MILL_THREAD_MCUT	Thread mill with multiple cuts
	MILL_THREAD_CIRC	Circular thread mill
DrillingTools	DRILL_DRILL	Drill
	DRILL_TAP	Tap
	DRILL_CENTER	Spotting drill
	DRILL_REAM	Reamer
	DRILL_CSINK	Countersink
	DRILL_TSINK	Counterbore for through holes
	DRILL_BORE	Boring tool
	DRILL_BCKBORE	Back boring tool
	DRILL_THREAD_MILL	Drill thread milling cutter
TurningTools	TURN_ROUGH	Roughing tool
	TURN_FINISH	Finishing tool
	TURN_THREAD	Threading tool
	TURN_RECESS	Recessing tool
	TURN_BUTTON	Button tool
	TURN_RECESS_TURN	Recess-turning tool
Touchprobes	TOUCH_PROBE	Touch probe
GrindingTools1)	GRIND_PIN	Grinding pin (cylindrical)
	GRIND_CONE	Grinding pin (conical)
	GRIND_CUP	Cup wheel
	GRIND_CYLINDER	Straight wheel

Category	OPC UA Identifier	Description
	GRIND_ANGULAR	Slant wheel
	GRIND_FACE	Facing wheel
DressingTools[1)]	DRESS_FIX_RADIUS	Stationary dresser (with radius)
	DRESS_ROT_RADIUS	Rotating dresser (with radius)
	DRESS_FIX_FLAT	Stationary dresser (flat)
	DRESS_ROT_FLAT	Rotating dresser (flat)
	UNDEF	Undefined

1) The OPC UA NC Server in NC software version 18 does not support grinding and dressing tool types yet.

3D Model Files for Tools: Validation Issues List

General information about usage of 3D models for tools and tool carriers (e.g., supported formats) is given in the Technical Manual of the respective control as well as its Setup, Testing and Running NC Programs User's Manual or Setup and Program Run User's Manual.

The *3DModel* component of 4.16 "ToolDataManagementType" provides a method to check 3D model files to be used for tool data items *CarrierKinematics* and *ToolShape*. Table 217 lists the possible 3D model file validation issues as of NC software version 18. (The list might be extended in future versions.)

Table 217: Validate3DModelFile Issues EnumValues

Value	DisplayName	Description
1	FormatUnknown	The file format is not a supported model format.
2	FileTooLarge	The file is too large to load.
3	FileInaccessible	Cannot open file.
4	ErrorOnRead	An error occurred while reading the file.
5	EmptyModel	The model is empty.
6	MeshNotWatertight	The model must have a closed shell, but it doesn't.
7	TooManyTriangles	The mesh has more triangles than are supported.
8	NotSuitableForItem	The file is not suitable for the specified use case.
9	FileNameInvalid	The file name contains incompatible characters.
10	FileNameTooLong	The file name is longer than is allowed for the data item.
11	FormatNotSuitable	The file format does not match the use case.
12	WrongLocation	The file location does not match the use case.
13	OverriddenFile	There is another file with this name in a location of higher priority.
14	MeshTypeMissing	The file requires a mesh type, but none is specified.
999	UnknownIssue	An unknown issue was found during validation.

Tool Data of Machine Manufacturers

Machine manufacturers can define machine-specific tool data. They can add more columns to the internal tool data tables.

With NC software version 18 the OPC UA NC Server identifies the additional data items by the internal table identifier followed by a double underscore and the internal column identifier (e.g., "T__EXAMPLEDATA"). The data items are grouped into a *ManufacturerExtension* tool data category.

Machine manufacturers manage the data items individually. It is possible that two machines of different manufacturers provide a data item with the same name but different meaning and usage.

With NC software version 18 all machine-specific tool data items are taken as potentially relevant for all tool types depending on the internal data source table.

HEIDENHAIN

DR. JOHANNES HEIDENHAIN GmbH
Dr.-Johannes-Heidenhain-Straße 5
83301 Traunreut, Germany
☎ +49 8669 31-0
FAX +49 8669 32-5061
info@heidenhain.de

Technical support	FAX	+49 8669 32-1000
Measuring systems	☎	+49 8669 31-3104
service.ms-support@heidenhain.de		
NC support	☎	+49 8669 31-3101
service.nc-support@heidenhain.de		
NC programming	☎	+49 8669 31-3103
service.nc-pgm@heidenhain.de		
PLC programming	☎	+49 8669 31-3102
service.plc@heidenhain.de		
APP programming	☎	+49 8669 31-3106
service.app@heidenhain.de		

www.heidenhain.com

Original document
1309365 · Ver06 · 10/2023 · PDF