Historical Data Access from Industrial Systems Specification

This OMG document replaces the draft adopted specification and original submission (mantis/02-10-03). It is an OMG Final Adopted Specification, which has been approved by the OMG board and technical plenaries, and is currently in the finalization phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by October 27, 2003.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*; however, at the time of this writing there were no pending issues.

The FTF Recommendation and Report for this specification will be published on November 28, 2003. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

OMG Adopted Specification
dtc/03-02-01
February 2003

Historical Data Access from Industrial Systems Specification

Final Adopted Specification
February 2003
dtc/03-02-01

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The company listed above has granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. The copyright holder listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

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

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.
IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE

BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents & Specifications, Report a Bug/Issue.

Contents

Preface

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 600 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The mission of The Open Group is to drive the creation of boundaryless information flow achieved by:

- Working with customers to capture, understand and address current and emerging requirements, establish policies, and share best practices;
- Working with suppliers, consortia and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies;

- Offering a comprehensive set of services to enhance the operational efficiency of consortia; and
- Developing and operating the industry's premier certification service and encouraging procurement of certified products.

The Open Group has over 15 years experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes tests for CORBA, the Single UNIX Specification, CDE, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at http://www.opengroup.org/ .

OMG Documents

The OMG Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

The OMG documentation is organized as follows:

OMG Modeling Specifications

Includes the UML, MOF, XMI, and CWM specifications.

OMG Middleware Specifications

Includes CORBA/IIOP, IDL/Language Mappings, Specialized CORBA specifications, and CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

Includes CORBAservices, CORBAfacilities, OMG Domain specifications, OMG Embedded Intelligence specifications, and OMG Security specifications.

Obtaining OMG Documents

The OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.)

OMG formal documents are available from our web site in PostScript and PDF format. Contact the Object Management Group, Inc. at:

OMG Headquarters
250 First Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English.

Helvetica bold - OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier bold` - Programming language elements.

Helvetica - Exceptions

Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Acknowledgments

The following company submitted this specification:

- ABB Utility Automation

Conformance to HDAIS

HDAIS specifies conformance points according to the table below.

Conformance Points

	A **Synchronous access**	**B** **Asynchronous access**
1 **Value retrieve &** **Discovery of time series**	ValueIO::SyncRead Node::Home Item::Home Aggregate::Home	ValueIO::AsyncRead Node::Home Item::Home Aggregate::Home

Conformance Points

2 **Value recording**	ValueIO::SyncUpdate ModifiedValueIO::Sync	ValueIO::AsyncUpdate ValueIO::Callback ModifiedValueIO::Async ModifiedValueIO::Callback
3 **Value attributes**	ItemAttributeIO::Sync ItemAttribute::Home	ItemAttributeIO::Async ItemAttributeIO::Callback ItemAttribute::Home
4 **Annotations**	AnnotationIO::Sync	AnnotationIO::Async AnnotationIO::Callback
5 **Play back**	N/A	ValueIO::Playback ValueIO::PlaybackCallback
6 **Discovery of schema**	DAIS::Type::Home DAIS::Property::Home	DAIS::Type::Home DAIS::Property::Home
7 **Discovery of objects for past times**	IDL Attribute Browse::Home::browse_base_time	IDL Attribute Browse::Home::browse_base_time

This table is a matrix where the left column and top row defines the conformance points, e.g. row 1 and column 1 defines the conformance point A1. The matrix cells define the interfaces that shall be conformed to, e.g. cell (conformance point) A1 defines the following interfaces to conform to ValueIO::SyncRead, Node::Home, Item::Home and Aggregate::Home.

An implementation shall obey the following:

- An implementation shall conform to 1.
- A 1 implementation may also conform to 2, 3, 6 and 7 in any combination (just 2, just 3, just 6, both 2 and 3, etc).
- A 2 implementation may also conform to 4.
- An implementation shall conform to one of A or B.
- An implementation may conform to both A and B.
- A B1 implementation may also conform to B5.

Note – Any other combinations than the above are non-conformant.

Overview 1

1.1 Introduction

Recording and archival of time series data in industrial control systems is made for the following purposes:

- verification of the actual system state before and during a disturbance,
- dispute resolution,
- basis for simulation,
- analysis of relations between data, and
- system performance analysis.

Time series data in this context mainly means measured or calculated values representing state variables within the industrial process. Measured data might be telemetered (i.e., collected from remote units). Parameter values (e.g., alarm limits, amplifier gains etc.), control commands and operator actions are also included.

This specification defines a number of interfaces for a time series data management facility. The motivation is to enable integration between the various existing or emerging products that produce or consume historical data, such as:

- Remote terminal units, process control units, SCADA (Supervisory Control and Data Acquisition) systems that produce historical data to be recorded.
- Maintenance and persistent storage systems that consume historical data (a historical data management facility).
- Clients retrieving data for discovery, presentation and update.
- Clients retrieving data for analysis and or input to calculations.
- Bridges to and from existing OPC Historical data services.

Figure 1-1 gives an architecture overview of server and client components that use the interfaces.

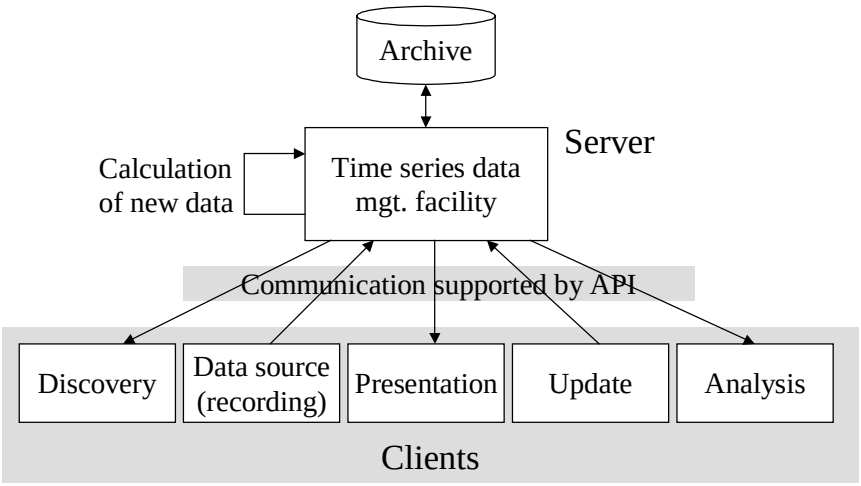

Figure 1-1 Client and server components using the HDAIS API

Figure 1-1 describes the data traffic crossing the interface between the server and its clients. The arrow directions indicate the major data flow direction. The arrow marked "calculation of new data" is outside the scope but indicates that new data can be created by calculations running within the server.

The data source components from Figure 1-1 and their relationships to major industrial control system components are shown in Figure 1-2.

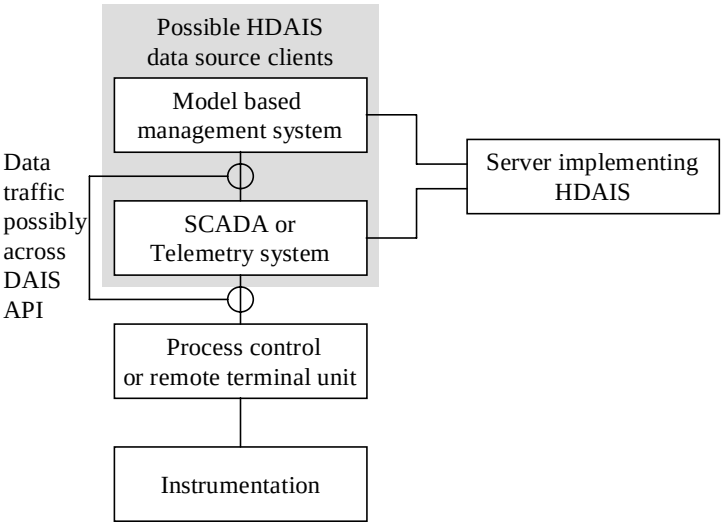

Figure 1-2 Relationships between major industrial control system components and HDAIS

Connection to the process instrumentation (sensors and actuators) is usually through process control or remote terminal units. The remote units are connected to a SCADA (Supervisory Control and Data Acquisition) or telemetry system, possibly using DAIS. A SCADA system makes the telemetry data available to operators or other systems including model based management systems. In the utility context, the model-based management system is called an Energy Management Systems (EMS). Control room operators monitor and control a power system using an EMS. It runs various network related power applications utilizing telemetry (state) and parameter data.

The SCADA/telemetry system or the model based management system typically uses the HDAIS server to record historical data. The recording data source from Figure 1-1 is then part of SCADA/telemetry or model based management system.

Telemetering is defined (from IEEE Std 1000 [1972]) as “measurement with the aid of intermediate means that permit the measurement to be interpreted at a distance from the primary detector.”

1.2 Problems Being Addressed

This specification includes interfaces to record and retrieve time series data. It is aimed to support domains as

- power systems, and
- industrial processes.

It may also be used in the control system domains for:

- space,
- rail way systems,
- air traffic control systems,
- manufacturing systems, and
- health monitoring systems.

Sources of time series data in these domains include:

- Process control units (e.g., interlocking equipment, protection equipment, process regulation equipment, etc.).
- RTUs or remote utility control centers connected via inter center protocols (e.g., ICCP or ELCOM).
- Applications producing calculated data (e.g., custom SCADA calculations, model-based calculations as State Estimator or Power Flows, etc.).
- Operators providing manually maintained data.

Time series data can describe any kind of state variables, control variables, or parameters existing in a control system. For example:

- State variables (also called measurements); e.g., analog values as voltages, pressure, fluid flows, electrical current, power flow, level or discrete values as on, off, tripped or blob type values as sound clips, video images, etc.

- Control variables; e.g., analog controls as set points, pulsed controls as raise/lower or discrete controls as on, off etc.
- Parameters; e.g., limit values, ratings, amplifier gains, filter parameters, etc.

At any given time data is available only for the objects that existed at that time. Moreover, when an object is deleted, its history for past times is not deleted. This is important for use in legal cases or post mortem reviews.

Clients usually don't know in advance the organization of data and the schema (information model) describing the exposed data. Hence the specification includes interfaces where clients can explore both data and schema.

1.2.1 Data Semantics

Time series data is hierarchically organized in trees of Nodes and Items where the Items are leafs. The Nodes in the hierarchy have a Type (e.g., substation, pump, breaker). An Item is an instance of a Property and the Property belongs to a Type.

An Item has a time series of ItemValues. An ItemValue consists of one or more triples consisting of

- value,
- time stamp, and
- quality code.

The value can be of multiple data types, for example:

- various numeric data types
- text
- blob type of data as sound clips, images, control programs etc.

1.2.2 Data Access

This specification provides interfaces for data access including:

- Discovery of Nodes and Items available in a server.
- Discovery of the information model supported by a server (e.g., available Types and their Properties).
- Synchronous and asynchronous read or write of ItemValues.
- Client side subscription callback interfaces for event driven transfer of ItemValues.

1.2.3 Concurrency Control

There are no explicit means to synchronize clients. Time stamping of data is provided so that clients can judge the age.

1.3 Problems Not Being Addressed

The following items are outside the scope of this specification:

- Configuration of the historical data management systems itself and its population of objects. An industrial process evolves over time and this is reflected by changes in configuration. Changes include addition and/or deletion of Nodes and Items; specialized data maintenance tools make this. Such tools are outside the scope of this specification.
- Specification of what time domain calculations that are available in the historical data management systems. This is regarded to be a server configuration issue.
- The actual collection and recording of ItemValues. This is regarded to be a client responsibility.
- User interface for presentation of discovered or recorded data. This is regarded to be a client responsibility.
- Tools for analysis or calculations on data. This is regarded to be a client responsibility.
- Security and encryption/decryption of data.
- Complex relations between resources (e.g., references between objects).

1.4 Design Rationale

Besides meeting the requirements spelled out in the RFP there are a number of design goals that have shaped solutions.

1.4.1 Adherence to OPC

OPC Historical Data Access (HDA) [7] and other OPC specifications has been in use for a number of years today and this specification leverage on the experience gained by OPC. There are a large number of OPC based products in the market place and cases where HDAIS and OPC will be bridged are likely. Adherence to OPC is important to facilitate simple bridging and porting HDAIS software to/from OPC HDA.

1.4.2 Adherence to OMG Data Acquisition from Industrial Systems (DAIS)

The OMG DAIS specification [2] is based on the OPC specifications Data Access [5] and Alarms & Events [6]. HDAIS is an extension of DAIS and is aimed to be fully compatible with DAIS and build on the same basis; i.e., OPC.

1.4.3 Simplicity and Uniformity

Some design principles used when creating OPC were:

- Method behavior is sometimes controlled by an input parameter.
- Related data is transferred in multiple parallel vectors.

- Outputs are always returned in one or more output parameters.

To simplify and get a more uniform interface these principles have been replaced by the following:

- A method has one single behavior resulting in some OPC methods being replaced by more than one DAIS method.
- Related data is kept together in structs resulting in reduction of the number parameters compared to OPC.
- Outputs are returned as method return results resulting in the OPC HRESULT parameter being replaced by exceptions and reduced number of output parameters compared to OPC.

1.4.4 High Performance Implementations

An HDAIS server is a real-time system required to deliver data in high rates and volumes. The performance requirements mean that a typical HDAIS server does not use a relational database management system for on-line operation but some kind of real-time database. The HDAIS API efficiently encapsulates such real-time databases from clients.

To effectively deliver data the HDAIS interface itself must not introduce performance bottlenecks. This has influenced the design in several ways, listed below.

1.4.4.1 Subscription

The subscription mechanism consists of two phases. In the first the client negotiates with the server on what data items to subscribe for and in the second the actual data transfer takes place. This minimizes the amount of transferred data between the server and the client during on-line operation.

1.4.4.2 Sequences

The HDAIS interface supports using sequences of data in calls rather than having calls requiring single valued parameters. This allows clients to ask for processing of multiple data in a single call rather than making multiple calls thus reducing the number of LAN round trips.

1.4.4.3 Iterators

Large volumes of data are not efficiently transferred in one method call. For this reason many methods return an iterator that is used to transfer optimal volumes of data in each call.

1.4.4.4 Data Value Representation

The basic unit of data is a union type: SimpleValue. SimpleValue exploits our knowledge of the basic data types needed and eliminates CORBA any from the highest bandwidth part of the interface. This can make a significant impact on performance when accumulated across large amounts of data.

Relations to Other Standards 2

2.1 Overview

An overview of relations between standards is shown below.

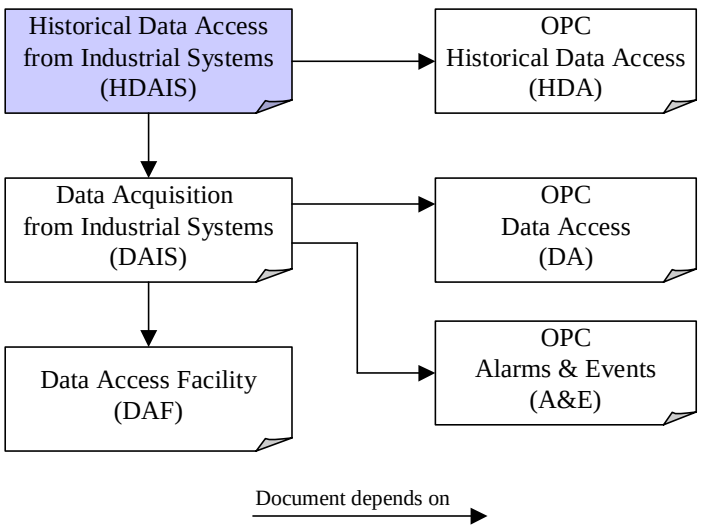

Figure 2-1 Overview relations to other standards

2.2 DAIS

The DAIS [2] specification describes an interface for Data Access and Alarms & Events from industrial control systems. HDAIS extends these interfaces with access to historical data. The basic data types from DAIS [2] and DAF [3] is reused in this specification as well as the call back pattern for event driven transfer of data from a server to its clients.

DAIS Alarms & Events primarily deals with the current alarm condition for objects. However DAIS Alarms & Events also has an interface that supports access of the event history. An event is not only a recording of a value or quality change but also includes information as

- the reason why the recording was made.
- the severity of the alarm condition.
- additional parameters that describe the event.

HDAIS supports recording of quality-coded values where the values have the simple structure as described in Section 1.2.1, “Data Semantics,” on pag e1-4 and do not support the above described more complex events.

2.3 OLE for Process Control (OPC)

OPC consists of a suite of specifications where the ones of interest for HDAIS and DAIS are shown to the right in Figure 2-1. HDAIS is a recast of OPC HDA [7] into OMG IDL following the same principles as DAIS recast OPC DA [5] and OPC A&E [6]. For a discussion of these principles refer to DAIS [2].

2.4 Data Access Facility (DAF)

The basic data types defined in the DAF [3] specification are used in the HDAIS specification. For a discussion of the data types used by HDAIS and DAIS refer to the DAIS [2] specification.

2.5 IEC 61346-1, Structuring and Naming

61346-1 [10] is a standard for hierarchical naming of objects. HDAIS and DAIS both support hierarchical naming according to 61346-1. Refer to DAIS [2] for more information.

2.6 IEC 61970

The IEC 61970-301 standard [9], also named CIM [8], describes a specific organization of power system objects in a hierarchical structure.

IEC 61970-301 specifies the two classes CurveSchedule and CurveShedData that can be used also for time series data. The correspondence between HDAIS and these classes are listed in Table 2-1.

Table 2-1 Correspondence between HDAIS and classes

IEC 61970-301 attribute	HDAIS property
CurveSchedule.pathName	Item.pathname
CurveShedData.xAxisData	ItemValue.time_stamp
CurveShedData.y1AxisData	ItemValue.value

The CurveShedData contains an alternate value CurveShedData.y2AxisData. If this value is present, it shall be mapped to a second Item. For a description of the HDAIS information model refer to Chapter 3.

HDAIS Informational Model 3

3.1 Overview

This section describes the data model seen through the HDAIS interface. The following classes describe the data model

- Node
- Item
- ItemValue
- ModifiedItemValue
- Annotation
- ItemAttribute
- ItemAttributeValue
- ItemAttributeDefinition
- Type
- Property
- AggregateDefinition

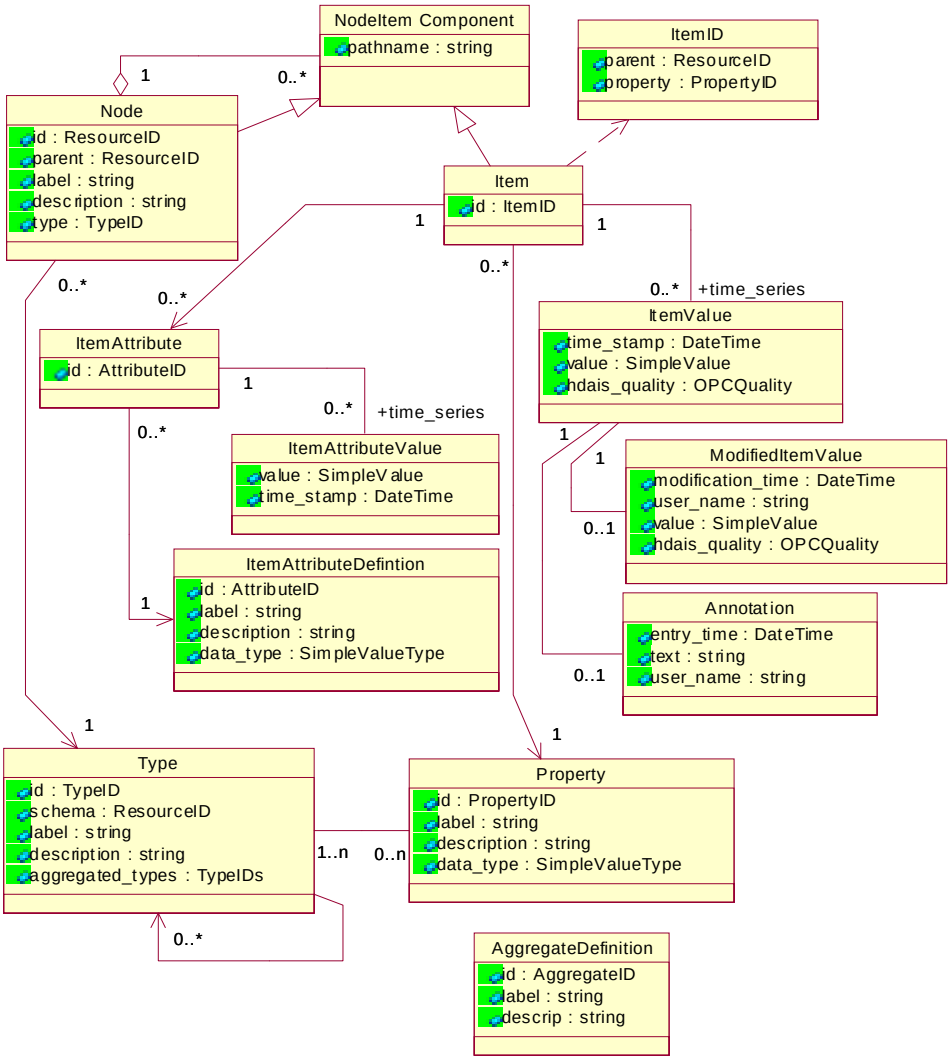

Figure 3-1 HDAIS Information Model

A Node represents objects that are hierarchically organized in a Node tree. A Node typically represents a real world object (e.g., Tank_200, Transformer_3, Measurement_100, etc.). A Node has a Type and the Type tells what kind of object a Node represents (e.g., a Tank, Transformer, Measurement, etc.). A Node can have a number of property values called Items. The Items are leafs in the Node tree. For a Measurement Node typical Items are measured value, limit values, etc. The Type also describes what Properties a Node has and the allowed Child Node Types. An Item is described by a Property. This model is identical to the DAIS Data Access (DA) model.

An Item represents a time series of Property values called ItemValues. HDAIS differs from DAIS DA in that an Item has a time series of ItemValues instead of just a single value. Each ItemValue is time stamped and has a quality code.

A time series has a start ItemValue and an end ItemValue. These ItemValues are called the bounding values. The time for the start ItemValue is called the start time and for the end ItemValue the end time. The start time and the end time forms a time interval.

An ItemValue may be modified (e.g., due to correction of an erroneous recording) and an optional ModifiedItemValue describes the modification.

An ItemValue may also be annotated and an optional Annotation describes the annotation.

Each HDAIS Item may have a number of ItemAttributes describing the treatment of the ItemValues (e.g., if ItemValues are being recorded, corresponding DAIS DA Item, etc.). As an ItemAttribute may change over time it has a time series consisting of ItemAttributeValues. Each ItemAttributeValue consists of a value and a time stamp.

An ItemAttributeDefinition describes each ItemAttribute. For a server there is one common set of ItemAttributeDefinitions for all Properties. This means that all Items always have ItemAttributes for all defined ItemAttributeDefinitions. ItemAttributes at an Item may however not have any ItemAttributeValues.

An AggregateDefinition describes a calculation that can be performed on a time series (e.g., max value, mean value, etc.).

The Node attributes parent, label, and description shall also be represented as Items such that their historical values can be recorded and kept. This allows having the history recorded for:

- Node.parent as a result of changing the Node position in the Node tree.
- Node.labe as a result of renaming.
- Node.description as a result of changing the description.

The Node.id and Node.type are both expected to be unchanged during the Nodes lifetime. The Item.id is also expected to be unchanged during the Items (and Nodes) lifetime.

Common Declarations 4

This section lists the HDAIS declarations common with the DAIS [2] and DAF [3] specifications.

4.1 Character Encoding

Refer to DAIS [2] specification.

4.2 DAFIdentifiers IDL

Refer to the DAF [3] specification.

4.3 DAFDescriptions IDL

This IDL is the same as DAF [3] with the difference that the PropertyID type now is included in the SimpleValue union.

```
//File: DAFDescriptions.idl
#ifndef _DAF_DESCRIPTIONS_IDL_
#define _DAF_DESCRIPTIONS_IDL_
#include <DAFIdentifiers.idl>
#include <TimeBase.idl>
#pragma prefix "omg.org"
module DAFDescriptions
{
    //++
    // Simple Types used as property values.
    //--
    // imported from identifiers module.
    typedef DAFIdentifiers::ResourceID ResourceID;
    typedef DAFIdentifiers::URI URI;

    // absolute time stamps in 100 nanosecond units
```

```
// base time is 15 October 1582 00:00 UTC
// as per Time Service specification
typedef TimeBase::TimeT DateTime;

// a complex number
struct Complex
{
      double real;
      double imaginary;
};

// a blob
typedef string                FileExtension;

struct Blob
{
      any                     blob_data;
      FileExtension           blob_data_type;
};

//++
// Resource Descriptions
//--
// properties are represented by their resource identifiers
typedef ResourceID PropertyID;

// SimpleValue's can take on the following types.
typedef short SimpleValueType;
const SimpleValueType RESOURCE_TYPE = 1;
const SimpleValueType URI_TYPE = 2;
const SimpleValueType STRING_TYPE = 3;
const SimpleValueType BOOLEAN_TYPE = 4;
const SimpleValueType INT_TYPE = 5;
const SimpleValueType UNSIGNED_TYPE = 6;
const SimpleValueType DOUBLE_TYPE = 7;
const SimpleValueType COMPLEX_TYPE = 8;
const SimpleValueType DATE_TIME_TYPE = 9;
const SimpleValueType ULONG_LONG_TYPE = 10;
const SimpleValueType BLOB_TYPE = 11;
const SimpleValueType PROPERTYID_TYPE = 12;
const SimpleValueType RESOURCES_TYPE = 101;
const SimpleValueType URIS_TYPE = 102;
const SimpleValueType STRINGS_TYPE = 103;
const SimpleValueType BOOLEANS_TYPE = 104;
const SimpleValueType INTS_TYPE = 105;
const SimpleValueType UNSIGNEDS_TYPE = 106;
const SimpleValueType DOUBLES_TYPE = 107;
const SimpleValueType COMPLEXES_TYPE = 108;
const SimpleValueType DATE_TIMES_TYPE = 109;
const SimpleValueType ULONG_LONGS_TYPE = 110;
const SimpleValueType BLOBS_TYPE = 111;
const SimpleValueType PROPERTYIDS_TYPE = 112;

// a SimpleValue is the object of a resource description.
union SimpleValue switch( SimpleValueType )
```

```
    {
        case RESOURCE_TYPE          : ResourceID resource_value;
        case URI_TYPE               : URI uri_value;
        case STRING_TYPE            : string string_value;
        case BOOLEAN_TYPE           : boolean boolean_value;
        case INT_TYPE               : long int_value;
        case UNSIGNED_TYPE          : unsigned long unsigned_value;
        case DOUBLE_TYPE            : double double_value;
        case COMPLEX_TYPE           : Complex complex_value;
        case DATE_TIME_TYPE         : DateTime date_time_value;
        case ULONG_LONG_TYPE        : unsigned long long ulong_long_value;
        case BLOB_TYPE              : Blob blob_value;
        case PROPERTYID_TYPE        : PropertyID propertyid_value;
        case RESOURCES_TYPE         : sequence<ResourceID> resource_values;
        case URIS_TYPE              : sequence<URI> uri_values;
        case STRINGS_TYPE           : sequence<string> string_values;
        case BOOLEANS_TYPE          : sequence<boolean> boolean_values;
        case INTS_TYPE              : sequence<long> int_values;
        case UNSIGNEDS_TYPE         : sequence<unsigned long> unsigned_values;
        case DOUBLES_TYPE           : sequence<double> double_values;
        case COMPLEXES_TYPE         : sequence<Complex> complex_values;
        case DATE_TIMES_TYPE        : sequence<DateTime> date_time_values;
        case ULONG_LONGS_TYPE       : sequence<unsigned long long> ulong_long_values;
        case PROPERTYIDS_TYPE       : sequence<PropertyID> propertyid_values;
    };

    // predicate and object for a resource description
    struct PropertyValue
    {
        PropertyID property;
        SimpleValue value;
    };
    typedef sequence<PropertyValue> PropertyValueSequence;

    // resource description with one subject, multiple predicates
    struct ResourceDescription
    {
        ResourceID id;
        PropertyValueSequence values;
    };
    typedef sequence<ResourceDescription> ResourceDescriptionSequence;

    // iterator for handling large numbers of resource descriptions
    interface ResourceDescriptionIterator
    {
        unsigned long max_left();
        boolean next_n(
        in unsigned long n,
        out ResourceDescriptionSequence descriptions );
        void destroy();
    };
};
#endif // _DAF_DESCRIPTIONS_IDL_
```

4.4 DAISCommon IDL

Refer to DAIS [2] specification.

4.5 Iterator Methods

Refer to DAIS [2] specification.

4.6 DAISNode IDL

Refer to DAIS [2] specification.

4.7 DAISType IDL

Refer to DAIS [2] specification.

4.8 DAISProperty IDL

Refer to DAIS [2] specification.

4.9 DAISSession IDL

Refer to DAIS [2] specification.

4.10 DAISServer IDL

Refer to DAIS [2] specification.

HDAIS API 5

5.1 Overview

HDAIS extends the DAIS with functionality for management of time series data. Figure 5-1 shows the interface objects for dealing with the data described in section 0.

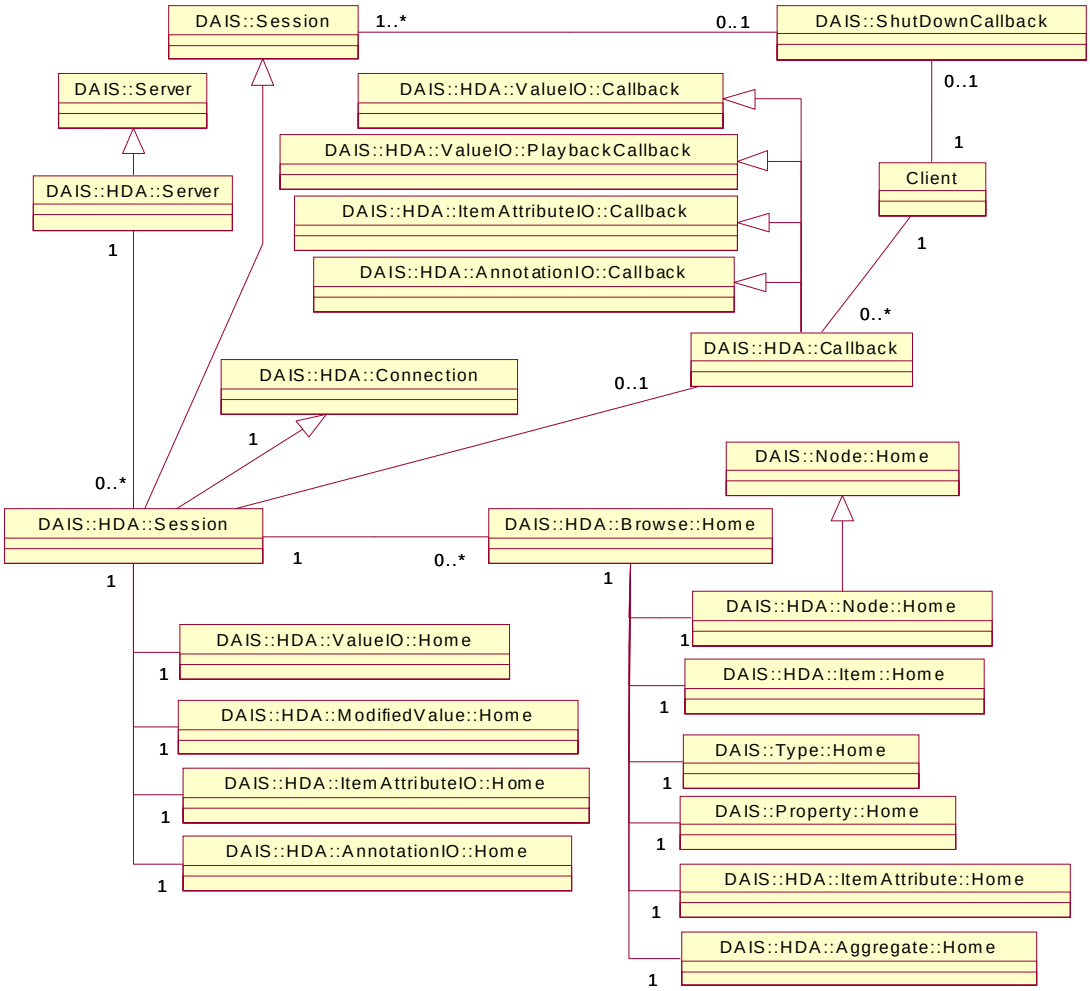

Figure 5-1 HDAIS Objects

Interfaces or objects belong to one of the four main categories:

- server and client objects
- connection interfaces
- browse objects
- data access (IO) objects

The server and client objects are: DAIS::HDA::Server, DAIS::HDA::Session, and DAIS::HDA::Callback. An HDAIS Server extends DAIS functionality by inheriting DAIS::Server hence an HDAIS DAIS::HDA::Server can also be a DAIS DA or A&E

Server. An HDAIS specific DAIS::HDA::Session is created by inheriting DAIS::Session. The Client may connect the HDAIS DAIS::HDA::Session object to a DAIS::HDA::Callback object that it implements. A Client may create several DAIS::HDA::Session objects and each DAIS::HDA::Session object shall have its own DAIS::HDA::Callback object if any. The DAIS::HDA::Callback object implements a number of callback interfaces that are defined with the data access objects.

The DAIS::HDA::Connection interface is implemented by the DAIS::HDA::Session object.

The data type specific browse objects are collected as a number of singleton objects at the DAIS::HDA::Browse::Home object. The DAIS::HDA::Browse::Home is an object at the DAIS::HDA::Session object. Each DAIS::HDA::Browse::Home has a base time at which the browsing for the data by type specific browse objects is made.

The following data type specific browse objects are defined:

- DAIS::HDA::Node::Home for Nodes.
- DAIS::HDA::Item::Home for HDAIS Items.
- DAIS::Type::Home for Types as defined in the DAIS specification.
- DAIS::Property::Home for Properties as defined in the DAIS specification.
- DAIS::HDA::ItemAttribute::Home for ItemAttributes.
- DAIS::HDA::Aggregate::Home for aggregate calculations.

The following type specific data access (IO) objects are defined:

- DAIS::HDA::ValueIO::Home for ItemValue access.
- DAIS::HDA::ModifiedValue::Home for access of ItemValues that has been modified.
- DAIS::HDA::ItemAttributeIO::Home for access of ItemAttributeValues.
- DAIS::HDA::AnnotationIO::Home for access of Annotations.

Each data type has its own name space where it's corresponding interfaces are defined; the following name spaces are defined:

- DAIS::HDA for HDAIS common data.
- DAIS::HDA::Node for Node browse data.
- DAIS::HDA::Item for Item browse data.
- DAIS::HDA::ItemAttribute for ItemAttribute browse data.
- DAIS::HDA::Aggregate for AggregateDefinition browse data.
- DAIS::HDA::AnnotationIO for Annotation data.
- DAIS::HDA::ValueIO for ItemValue data.
- DAIS::HDA::ModifiedValueIO for ModifiedItemValue data.
- DAIS::HDA::ItemAttributeIO for ItemAttributeValue data

5.2 HDAIS Common IDL

```
// File: HDAISCommon.idl
#ifndef __HDAIS_COMMON_IDL
#define __HDAIS_COMMON_IDL
```

```
#include <DAISCommon.idl>

module DAIS {
module HDA {

typedef unsigned long        AggregateID;

const   Error   ERROR_AGGREGATE_NOT_AVAILABLE        = 0x0100;
const   Error   ERROR_DATA_ALREADY_EXIST             = 0x0200;
const   Error   ERROR_DATA_DOES_NOT_EXIST            = 0x0300;
const   Error   WARNING_MORE_DATA_THAN_REQUESTED     = 0x1000;
const   Error   WARNING_NO_DATA                      = 0x2000;
const   Error   WARNING_MORE_EXTREEM_VALUES          = 0x3000;
const   Error   RESULT_DATA_INSERTED                 = 0x8000;
const   Error   RESULT_DATA_REPLACED                 = 0x9000;

const   OPCQuality OPCHDA_EXTRADATA          = 0x00010000;
const   OPCQuality OPCHDA_INTERPOLATED       = 0x00020000;
const   OPCQuality OPCHDA_RAW                = 0x00040000;
const   OPCQuality OPCHDA_CALCULATED         = 0x00080000;
const   OPCQuality OPCHDA_NOBOUND            = 0x00100000;
const   OPCQuality OPCHDA_NODATA             = 0x00200000;
const   OPCQuality OPCHDA_DATALOST           = 0x00400000;
const   OPCQuality OPCHDA_CONVERSION         = 0x00800000;

typedef unsigned short       CancelID;
typedef unsigned short       TransactionID;

typedef     unsigned long                   AttributeID;
typedef     sequence<AttributeID>           AttributeIDs;

struct TimeInterval {
    DateTime                start;
    DateTime                end;};

struct Value {
    SimpleValue             val;
    DateTime                time_stamp;
    OPCQuality              quality;};
};
};
#endif // __HDAIS_COMMON_IDL
```

Error

Error is an enumeration error code defined in the DAIS specification in the IDL DAISCommon. This specification extends that enumeration as given below.

EnumValue	Description
ERROR_AGGREGATE_NOT_AVAILABLE	The requested aggregate is not available. The corresponding OPC code is OPC_E_NOT_AVAIL
ERROR_DATA_ALREADY_EXIST	The inserted data already exists. The corresponding OPC code is OPC_E_DATAEXISTS.
ERROR_DATA_DOES_NOT_EXIST	The updated data does not exist. The corresponding OPC code is OPC_E_NODATAEXISTS.
WARNING_MORE_DATA_THAN_ REQUESTED	More data is available in the time range beyond the number of values requested. This return code is only valid in a response to read_raw method calls. The client may continue to call read_raw to get the remaining data. The corresponding OPC code is OPC_S_MOREDATA.
WARNING_NO_DATA	No ItemValues were found to delete. The corresponding OPC codes are OPC_S_NODATA (for ItemValues) and OPC_S_CURRENTVALUE (for ItemAttributeValues).
WARNING_MORE_EXTREEM_VALUES	Several identical max or min ItemValues were found at different times in the interval. The corresponding OPC code is OPC_S_EXTRADATA.
RESULT_DATA_INSERTED	ItemValue updates in an insert/update operation that was inserted. The corresponding OPC code is OPC_S_INSERTED.
RESULT_DATA_REPLACED	ItemValue updates in an insert/update operation that was replaced. The corresponding OPC code is OPC_S_REPLACED.

AggregateID

The AggregateID is an enumeration that identifies the various aggregate calculations. The definition of the enumeration values can be found in Section 5.6.4, “HDAISAnnotationIO,” on page 5-61.

OPCQuality

The OPCQuality is a flag word giving the OPC quality as defined in the DAIS specification in the IDL DAISCommon. HDAIS extends the flags as listed below.

Flag	Description
OPCHDA_EXTRADATA	More than one piece of data that may be hidden exists at same timestamp. This is the case when a value has been modified and both an ItemValue and a ModifiedItemValue exist.
OPCHDA_INTERPOLATED	Interpolated data value.
OPCHDA_RAW	The ItemValue is returned without any processing made.

OPCHDA_CALCULATED	Calculated data value, as would be returned from a read_processed call.
OPCHDA_NOBOUND	No data found for upper or lower bound values. For each missing bound value and empty ItemValue with this code is returned.
OPCHDA_NODATA	No data collected. Archiving not active (for item or all items).
OPCHDA_DATALOST	Collection started / stopped / lost.
OPCHDA_CONVERSION	Scaling / conversion error.

CancelID

CancelID is a server generated handle generated for each asynchronous call made by a client. A client can use the CancelID to abort an asynchronous call that hasn't finished.

TransactionID

A TransactionID is a handle created by clients and used in asynchronous calls. A server sends the TransactionID back to a client in the callback corresponding to the original asynchronous call from the client.

AttributeID

The AttributeID identifies an ItemAttribute.

TimeInterval

TimeInterval defines a time interval. It is used to specify start and stop times when accessing data. If for read operations no ItemValues exist matching the start or stop times, an empty ItemValue with the OPCQuality set to OPCHDA_NOBOUND shall be returned.

Member	**Description**
start	The interval start time.
end	The interval end time.

Value

Value is a struct that holds the ItemValue data.

Member	**Description**
val	The actual value.
time_stamp	The time stamp when the value was originally recorded.
quality	The quality of the value.

5.3 Server and Client Objects

This section describes the server and client objects defined in the following IDLs

- HDAISServer
- HDAISSession
- HDAISClient

5.3.1 HDAISServer

5.3.1.1 HDAISServer Overview

The fundamental HDAIS service object from which session objects can be obtained. The DAIS::HDA::Server object is normally persistent and is accessed via a naming or trader service. From the DAIS::HDA::Server object it is possible to create session objects for access of

- time series data (HDAIS)
- data (DAIS DA)
- alarms and events (DAIS A&E)

Sessions can be created for a view as described in DAIS [2] section 3.2.2.1.

The DAIS::HDA::Server object corresponds to the IOPCHDA_Server object.

5.3.1.2 HDAISServer IDL

```
//File: HDAISServer.idl
#ifndef _HDAIS_SERVER_IDL
#define _HDAIS_SERVER_IDL

#include <DAISServer.idl>
#include <HDAISSession.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {

interface Server : DAIS::Server
{
    exception InternalServerProblem{string reason;};

    Session create_historical_data_access_session(
        in string                   session_name)
        raises (InternalServerProblem, DuplicateName);

    Session create_historical_data_access_session_for_view(
        in string                   session_name,
        in string                   view_name)
        raises (InternalServerProblem, DuplicateName, InvalidView);
```

```
        readonly attribute unsigned longmax_returned_values;

};};};
#endif // _HDAIS_SERVER_IDL
```

InternalServerProblem

InternalServerProblem is an exception telling that the server cannot respond to the call because of some internal problem. The server might not have all functions up and running or its internal configuration might be erroneous.

For the other exceptions refer to the DAISServer IDL.

Server

An object that implements the HDAIS server and inherits the DAIS::Server. The DAIS::HDA::Server object supports views in the same way as the DAIS::Server object.

create_historical_data_access_session ()

create_historical_data_access_session() is a method that creates a time series data access session object for the default view.

Parameter	Description
session_name	The name of the session. If an empty name is supplied, the server will create a name for the session. If a duplicate name is supplied, no session is generated.
return	A reference to the created DAIS::HDA::Session object.

create_historical_data_access_session_for_view ()

create_historical_data_access_session_for_view() is a method that creates a time series data access session object for a specified view.

Parameter	Description
session_name	The name of the session. If an empty name is supplied, the server will create a name for the session. If a duplicate name is supplied, no session is generated.
view_name	The name of the view to open.
return	A reference to the created DAIS::HDA::Session object.

max_returned_values

max_returned_values is the maximum number of values that can be returned by the server on a per Item basis. A value of 0 indicates that the server forces no limit on the number of values it can return.

5.3.2 HDAISSession

5.3.2.1 HDAISSession Overview

The DAIS::HDA::Session object implements the data access service on a per client basis. A historical data access session object has a number of services provided by one singleton object each. Each singleton object provides methods for manipulation of a specific data type.

The DAIS::HDA::Session object implement functions from the DAIS::Session interface and the DAIS::HDA::Connection interface.

Each client may instantiate one or more DAIS::HDA::Sessions. If callbacks are used, each session object shall have an associated DAIS::HDA::Callback object.

The DAIS::HDA::Session object corresponds to the IOPCHDA_Server object.

5.3.2.2 HDAISSession IDL

```
//File: HDAISSession.idl
#ifndef _HDAIS_SESSION_IDL
#define _HDAIS_SESSION_IDL

#include <DAISSession.idl>
#include <HDAISBrowse.idl>
#include <HDAISConnection.idl>
#include <HDAISValueIO.idl>
#include <HDAISModifiedValueIO.idl>
#include <HDAISItemAttributeIO.idl>
#include <HDAISAnnotationIO.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {

interface Session : DAIS::Session, Connection
{
    readonly attribute ValueIO::Home            item_value_home;

    readonly attribute ModifiedValueIO::Homemodified_item_value_home;

    readonly attribute ItemAttributeIO::Home  item_attribute_home;

    readonly attribute AnnotationIO::Home     annotation_home;

    Browse::Home create_browser (
        in   DateTime                           browse_base_time);
};
};};
#endif // _HDAIS_SESSION_IDL
```

Session

Sesson is an object implementing the data access functions. It inherits common functionality as shut down callbacks and session status from DAIS::Session. It also implements the Connection interface, see Section 5.4, “Connection Interfaces,” on page 5-11.

item_value_home

item_value_home is an attribute holding the ItemValue data access singleton object.

modified_item_value_home

modified_item_value_home is an attribute holding the ModifiedItemValue data access singleton object.

item_attribute_home

item_attribute_home is an attribute holding the ItemAttributeValue data access singleton object.

annotation_home

annotation_home is an attribute holding the Annotation data access singleton object.

create_browser()

create_browser() is a method that creates a browse object.

Parameter	Description
browse_base_time	The base time for the browsing. A zero value means use current time. A client may check the time actually used by the server in the Browse::Home object attribute browse_base_time.
return	The browse object.

5.3.3 HDAISClient

5.3.3.1 HDAISClient overview

The DAIS::HDA::Callback interface shall be implemented by a client for it to receive callbacks from the HDAIS server. If a client only uses synchronous HDAIS interfaces, this object is not needed. For each DAIS::HDA::Session object there shall be one DAIS::HDA::Callback object if callbacks from the server are used.

The DAIS::HDA::Callback interface corresponds to the OPC interface IOPCHDA_DataCallback.

5.3.3.2 HDAISClient IDL

```
//File HDAISClient.idl
#ifndef _HDAIS_CLIENT_IDL
#define _HDAIS_CLIENT_IDL

#include <HDAISAsyncIO.idl>
#include <HDAISPlayback.idl>
#include <HDAISAnnotationIO.idl>
#include <HDAISItemAttributeIO.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {

interface Callback :   ValueIO::Callback,
                       ValueIO::PlaybackCallback,
                       AnnotationIO::Callback,
                       ItemAttributeIO::Callback
{

     void on_cancel_complete (
          in    CancelID                cancel_id);
};
};};
#endif // _HDAIS_CLIENT_IDL
```

Callback

Callback is an object that the client shall implement to be able to receive callbacks from the server.

on_cancel_complete ()

on_cancel_complete() is a method that notifies a client that the cancellation request has been serviced.

Parameter	Description
cancel_id	The cancellation number from the server for the original asynchronous operation and used by the client to request the cancellation.

5.4 Connection Interfaces

In contrary to DAIS there is no group objects defined in HDAIS. Instead connections for a time series (Items) are established by exchanging handles between the client and the server. The interface supporting this is DAIS::HDA::Connection defined in the HDAISConnection IDL.

5.4.1 HDAISConnection Overview

The DAIS::HDA::Connection interface is used to establish and manage bilateral association between Item handles in the server and the client.

The DAIS::HDA::Connection interface is implemented by the DAIS::HDA::Session object. DAIS::HDA::Connection interface has the corresponding OPC methods

- create IOPCHDA_Server::GetItemHandles
- remove IOPCHDA_Server::ReleaseItemHandles
- validate IOPCHDA_Server::ValidateItemIDs

5.4.2 HDAISConnection IDL

```
//File: HDAISConnection.idl
#ifndef __HDAIS_CONNECTION_IDL
#define __HDAIS_CONNECTION_IDL
#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {

struct Description {
    ServerItemHandle            server_handle;
    ClientItemHandle            client_handle;};

typedef sequence<Description>   Descriptions;

struct SetUp {
    ServerItemIdentification    server_id;
    ClientItemHandle            client_handle;};

typedef sequence<SetUp>         SetUps;

struct ValidateSetUp {
    ServerItemIdentification    server_id;
    ServerItemHandle            server_handle;
    ClientItemHandle            client_handle;};

typedef sequence<ValidateSetUp> ValidateSetUps;

interface Home
{
    Descriptions create (
        in  SetUps              connection_setups,
        out ItemErrors          errors);

    void remove (
        in  ServerItemHandles   server_handles,
        out ItemErrors          errors);

    ItemErrors validate (
```

```
            in    ValidateSetUps              validate_setups);
};
};};
#endif // __HDAIS_CONNECTION_IDL
```

Description

Description is a struct that contains the association between the server and client generated handles.

Member	Description
server_handle	Server handle as defined in the DAISCommon IDL [2].
client_handle	Client handle as defined in the DAISCommon IDL [2].

SetUp

SetUp is a struct used by a client to create an association between server and client handles.

Member	Description
server_id	The identification of an Item as defined in the DAISCommon IDL [2].
client_handle	Client handle as defined in the DAISCommon IDL [2].

ValidateSetUp

ValidateSetUp is a struct that contains the information about an association between client handles, server handles and Item identifications.

Member	Description
server_id	The identification of an Item as defined in the DAISCommon IDL [2].
client_handle	Client handle as defined in the DAISCommon IDL [2].
client_handle	Client handle as defined in the DAISCommon IDL [2].

Home

Home is an interface that has the operations to manage associations between server and client handles.

create ()

create() is a method for creation of server and client handle associations.

Parameter	Description
connection_setups	A sequence specifying the wanted associations between client handles and Item identifications.
errors	A sequence reporting the items where no associations were created due to an error. Reported errors are: -- ERROR_BAD_RIGHTS -- ERROR_UNKNOWN_PATHNAME -- ERROR_UNKNOWN_ITEMID -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.
return	A sequence with the successfully created associations.

remove ()

remove() is a method for removal of server and client associations.

Parameter	Description
server_handles	The server handles for which associations shall be removed.
errors	A sequence reporting the items where no associations were created due to an error. Reported errors are: -- ERROR_INVALID_DAIS_HANDLE For the error codes refer to the DAISCommon IDL [2].

validate ()

validate() is a method for verifying that associations still are valid.

Parameter	Description
validate_setups	A sequence of associations to be verified as seen by the client.
return	A sequence reporting the items that no longer has an association at the server. The reported errors are: -- ERROR_BAD_RIGHTS -- ERROR_UNKNOWN_PATHNAME -- ERROR_UNKNOWN_ITEMID -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

5.5 Browse Interfaces

The browse interfaces consist of one IDL definition for each data type specific browse object. The following IDLs are defined:

- HDAISBrowse for the DAIS::HDA::Browse::Home object.
- HDAISNode for the DAIS::HDA::Node::Home object.
- HDAISItem for the DAIS::HDA::Item::Home object.
- HDAISAggregate for the DAIS::HDA::Aggregate::Home object.
- HDAISItemAttribute for the DAIS::HDA::ItemAttribute::Home object.

5.5.1 Mapping to OPC HDA

Mapping of the browse interface to OPC is not straightforward. The two main differences are that HDAIS

- does not require client browse information at the server side
- separate OPC branch and leaf nodes into the HDAIS types Node and Item.

This means that the HDAIS Node and Item browse interfaces behave differently from OPC but have the same functionality. The table below shows how the OPC HDA browse methods map to the Node and Item browse methods.

OPC methods	**OPC parameters**	**HDAIS methods**	**HDAIS parameters**
IOPCHDA_Server::CreateBrowse()	Supply filter parameters once when the browse object is created.	Node::find_by_parent() Node::find_by_type() Item::find_by_parent() Item::find_by_type()	Supply the filter parameters at each call.
IOPCHDA_Browser::GetEnum()	dwBrowseType set to OPCHDA_BRANCH	Node::find_by_parent()	
IOPCHDA_Browser::GetEnum()	dwBrowseType set to OPCHDA_ITEMS or OPCHDA_LEAF	Item::find_by_parent()	
IOPCHDA_Browser::GetEnum()	dwBrowseType set to OPCHDA_FLAT	Item::find_by_type()	
N/A as OPC does not distinguish Nodes as own types.		Node::find_by_type()	
IOPCHDA_Browser:: ChangeBrowsePosition()		N/A as no server side cursor exists	
IOPCHDA_Browser::GetItemID		Node::get_pathnames() Item::get_pathnames()	
IOPCHDA_Browser:: GetBranchPosition()		N/A as no server side cursor exists	

5.5.2 HDAISBrowse

5.5.2.1 HDAISBrowse overview

The Browse::Home object is an object at a Session. It is a container of the data type specific browse objects as shown in Figure 5-1 on page 5-2.

A Browse::Home object can be created for a specific time. The browser will then expose objects (Nodes, Items etc.) as they existed at this time. This makes it possible to browse for disappeared objects. This requires that the server keep a history of all objects that have existed.

The Browse::Home object does not correspond directly to an OPC interface as the OPC browse interfaces are scattered. In OPC browse objects have local states initialized by clients. This is not possible in HDAIS. Clients must maintain such states at the client side and send it to the server in each call.

5.5.2.2 HDAISBrowse IDL

```
//File: HDAISBrowse.idl
#ifndef _HDAIS_BROWSE_IDL
#define _HDAIS_BROWSE_IDL

#include <HDAISNode.idl>
#include <HDAISItem.idl>
#include <DAISProperty.idl>
#include <DAISType.idl>
#include <HDAISAggregate.idl>
#include <HDAISItemAttribute.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module Browse {

interface Home
{
    readonly attribute Node::Home           node_home;

    readonly attribute Item::Home           item_home;

    readonly attribute Property::Home       property_home;

    readonly attribute Type::Home           type_home;

    readonly attribute Aggregate::Home      aggregate_home;

    readonly attribute ItemAttribute::Home  item_attribute_home;

    readonly attribute DateTime             browse_base_time;
};
};};};
```

#endif // _HDAIS_BROWSE_IDL

Home

Home is a container object for all data type specific browse objects.

node_home

node_home is an attribute holding the Node browse singleton object.

item_home

item_ home is an attribute holding the Item browse singleton object.

property_home

property_home is an attribute holding the Property browse singleton object. For a description of the Property::Home object refer to the DAISProperty IDL [2].

type_home

type_home is an attribute holding the Type browse singleton object. For a description of the Type::Home object refer to the DAISType IDL [2].

aggregate_home

aggregate_home is an attribute holding the AggregateDefinition browse singleton object.

item_attribute_home

item_attribute_home is an attribute holding the ItemAttribute browse singleton object.

browse_base_time

browse_base_time is an attribute holding a base time for the browsing. The browser will present the objects that existed at the time specified in browse_base_time. If this time is zero, current time is used. A server that doesn't support base times for browsing will always set this value to zero.

5.5.3 HDAISNode

5.5.3.1 HDAISNode overview

The interface is used to browse Nodes, refer to Chapter 3 for the information model.

For a discussion on Node browsing refer to the DAISNode IDL [2].

Refer to Section 5.5.1, "Mapping to OPC HDA," on page 5-15 for a description of the mapping to OPC HDA.

5.5.3.2 HDAISNode IDL

```
//File: HDAISNode.idl
#ifndef _HDAIS_NODE_IDL
#define _HDAIS_NODE_IDL
#pragma prefix "omg.org"
#include <DAISNode.idl>

module DAIS {
module HDA {
module Node {

interface Home : DAIS::Node::Home
{
    ResourceID          get_root();
};
};};};
#endif // _HDAIS_NODE_IDL
```

Home

Home is an object used for browsing nodes. Most functionality is inherited from the DAIS::Node::Home interface, refer to the DAISNode IDL [2].

get_root()

get_root() is a method to get the root Node of the Node tree.

Parameter	Description
return	The root node identification to be used in further calls.

5.5.4 HDAISItem

5.5.4.1 HDAISItem overview

The interface is used to browse Items, refer to Chapter 3 for the information model.

For a discussion on Item browsing refer to the DAISItem IDL [2]. HDAIS Items differ from DAIS DA Items in that they don't have the six fixed attributes (value, time_stamp, etc). Instead the attributes are divided between the ItemValue (value, time_stamp, and quality) and the ItemAttributeValue. The ItemAtributeValue types may vary dependent on the implementation hence there is a browse interface (ItemAttribute::Home) also for the ItemAttributeValue types (i.e., the ItemAttribute interface).

Refer to Section 5.5.1, "Mapping to OPC HDA," on page 5-15 for a description of the mapping to OPC HDA.

5.5.4.2 HDAISItem IDL

```
//File: HDAISItem.idl
#ifndef __HDAIS_ITEM_IDL
#define __HDAIS_ITEM_IDL
#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module Item {

struct Description {
    ItemID                  id;
    string                  label;};

typedef sequence<Description>       Descriptions;

typedef unsigned short      OPCOperator;
const OPCOperator           OPCHDA_EQUAL            =1;
const OPCOperator           OPCHDA_LESS             =2;
const OPCOperator           OPCHDA_LESSEQUAL        =3;
const OPCOperator           OPCHDA_GREATER          =4;
const OPCOperator           OPCHDA_GREATEREQUAL     =5;
const OPCOperator           OPCHDA_NOTEQUAL         =6;

struct AttributeFilter {
    AttributeID             attribute_id;
    OPCOperator             operator;
    SimpleValue             filter_value;
};
typedef sequence<AttributeFilter>   AttributeFilters;

interface Iterator
{
    boolean next_n (
        in   unsigned long          n,
        out  Descriptions           items);

    void reset();

    Iterator clone();

    void destroy();
};

interface Home
{
    exception UnknownResourceID {string reason;};
    exception UnknownItemID {string reason;};
    exception InvalidFilter {string reason;};

    Description find (
        in   ItemID                         item)
        raises (UnknownItemID);
```

```
        Descriptions find_each(
            in   ItemIDs                    items)
            raises (UnknownItemID);

        Iterator find_by_parent (
            in   ResourceID                 node,
            in   string                     pathname_criteria,
            in   SimpleValueType            data_type_filter,
            in   AttributeFilter            attribute_filter)
            raises (UnknownResourceID, InvalidFilter);

        Iterator find_by_type (
            in   ResourceID                 node,
            in   string                     pathname_criteria,
            in   SimpleValueType            data_type_filter,
            in   TypeIDs                    type_filter,
            in   AttributeFilter            attribute_filter)
            raises (UnknownResourceID, InvalidFilter);

        Strings get_pathnames (
            in   ItemIDs                    items);

        ItemIDs get_ids (
            in   Strings                    pathnames);
};
};};};
#endif // _HDAIS_ITEM_IDL
```

Description

Description is a struct that identifies an Item.

Member	Description
id	The identification of this item.
label	The unique name of the Item within its parent Node. The label is the same as for the Items corresponding Property.

OPCOperator

OPCOperator is an enumeration of comparison operators used when filtering Items on ItemAttributeValues.

AttributeFilter

AttributeFilter is a struct used to filter Items based on a filter criteria specified by the members.

Member	Description
attribute_id	The identification of the ItemAttribute to filter on.

operator	The operator to use according to OPCOperator.
filter_value	A value constant to compare with.

Iterator

The same iterator methods as in DAIS is used, refer to the DAIS specification section “Iterator Methods IDL.”

Home

Home is an object used for browsing Items.

find ()

find() is a method that returns the Description of a known Item.

This method does not have a corresponding OPC method.

Parameter	Description
item	The identification of the Item.
return	The description of the Item.

find_each ()

find_each() is a method that returns the descriptions for a sequence of known Items.

This method does not have a corresponding OPC method.

Parameter	Description
items	A sequence of item identifications.
return	An iterator holding the item descriptions.

find_by_parent ()

find_by_parent() is a method that for a given parent Node returns the child items of that Node. To reach all items in the parent Node sub-tree use this method repeatedly for each Node level in the sub-tree. To reach the child Nodes in the parent Node sub-tree use the method Node::Home::find_by_parent().

find_by_parent() has three filter input parameters. All filters must be fulfilled for an item for it to be selected.

Parameter	Description
node	The parent node identification.

pathname_criteria	The filter selects items with pathnames matching the pathname_criteria. For a description of the filter refer to the section on filter definitions in the DAIS specification [2].
data_type_filter	Select items having the specified canonical data type.
attribute_filters	An ItemAttribute filter specification. If more than one filter entry is specified, all must be fulfilled for an item to be candidate for selection. Note that the OPC specification has ItemAttributes defined both for the pathname and the data type. In HDAIS they are not defined as ItemAttributes as the -- pathname is an attribute of Node and Item. -- data type is an attribute of Property.
return	An iterator holding the Item descriptions.

find_by_type ()

find_by_type() is a method that for a given sub-tree parent Node finds all child Items matching the filter criteria. This will return all items under the given sub-tree root Node. This will make the items in the sub-tree to appear flattened out. This corresponds to the OPC method IOPCHDA::GetEnum() with the parameter dwBrowseType set to OPCHDA_FLAT.

find_by_parent() has four filter input parameters. All filters must be fulfilled for an item for it to be selected.

Parameter	**Description**
node	The parent node identification.
pathname_criteria	The filter selects items with pathnames matching the pathname_criteria. For a description of the filter refer to the section on filter definitions in the DAIS specification [2].
data_type_filter	Selects items having the specified canonical data type.
type_filter	Selects Items belonging to Nodes that have a Type that matches one of the Types in the type_filter.
attribute_filters	An ItemAttribute filter specification. If more than one filter entry is specified, all must be fulfilled for an item to be candidate for selection. Note that the OPC specification has ItemAttributes defined both for the pathname and the data type. In HDAIS they are not defined as ItemAttributes as the -- pathname is an attribute of Node and Item. -- data type is an attribute of Property.
return	An iterator holding the Item descriptions.

get_pathnames()

get_pathnames() is a method to translate a sequence of item identifications to the corresponding sequence of pathnames. If an item fails to translate to a pathname (due to an unknown identification), the corresponding pathname is an empty string.

Parameter	Description
items	A sequence of Items identifications.
return	The corresponding sequence of pathnames.

get_ids()

get_ids() is a method that translates a sequence of pathnames to the corresponding sequence of node identifications. If a pathname fails to translate to node identification (due to an unrecognized pathname), the corresponding node identification is NULL.

get_ids() has no corresponding OPC method.

Parameter	Description
pathnames	A sequence of pathnames.
return	The corresponding sequence of Item identifications.

5.5.5 HDAISItemAttribute

5.5.5.1 HDAISItemAttribute overview

The interface is used to browse the ItemAtributeDefintions, refer to Chapter 3 for the information model.

5.5.5.2 HDAISItemAttributes IDL

```
//File: HDAISItemAttribute.idl
#ifndef _HDAIS_ITEM_ATTRIBUTE_IDL
#define _HDAIS_ITEM_ATTRIBUTE_IDL
#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ItemAttribute {

struct Description {
    AttributeID         id;
    string              label;
    string              descrip;
    SimpleValueType     data_type;};
```

```
typedef sequence<Description>          Descriptions;

//const AttributeID OPCHDA_DATA_TYPE          = 0x0001;
//const AttributeID OPCHDA_DESCRIPTION        = 0x0002;
const AttributeID OPCHDA_ENG_UNITS            = 0x0003;
const AttributeID OPCHDA_STEPPED              = 0x0004;
const AttributeID OPCHDA_ARCHIVING            = 0x0005;
const AttributeID OPCHDA_DERIVE_EQUATION      = 0x0006;
//const AttributeID OPCHDA_NODE_NAME          = 0x0007;
//const AttributeID OPCHDA_PROCESS_NAME       = 0x0008;
const AttributeID OPCHDA_SOURCE_NAME          = 0x0009;
const AttributeID OPCHDA_SOURCE_TYPE          = 0x000a;
const AttributeID OPCHDA_NORMAL_MAXIMUM       = 0x000b;
const AttributeID OPCHDA_NORMAL_MINIMUM       = 0x000c;
//const AttributeID OPCHDA_ITEMID             = 0x000d;
const AttributeID HDAIS_SOURCE_RESOURCE       = 0x0100;
const AttributeID HDAIS_SOURCE_PROPERTY       = 0x0101;
const AttributeID HDAIS_ITEM_PATH_NAME        = 0x0102;
const AttributeID HDAIS_PARENT_NAME           = 0x0103;
const AttributeID HDAIS_INSERT_TIMES          = 0x0104;

interface Home
{
    exception UnknownID {string reason;};

    Description find (
        in   AttributeID                id)
        raises (UnknownID);

    Descriptions find_all();
};
};};};
#endif // _HDAIS_ITEM_ATTRIBUTE_IDL
```

Description

Description is a struct describing an ItemAttributeDefinition.

Member	Description
id	The identification of an ItemAttributeDefinition.
label	The label of an ItemAttributeDefinition.
descrip	The description of an ItemAttributeDefinition.
type	The data type of an ItemAttributeDefinition.

A number of ItemAttributes are defined and listed in the AttributeID section below.

AttributeID

AttributeID is defined in the HDAISCommon IDL refer to Section 5.2, “HDAIS Common IDL,” on page 5-3. The table below lists the ItemAttributeDefinitions and their Description member values.

EnumValue (id)	label	description	data_type
OPCHDA_ENG_UNITS	engineeringUnit	The engineering unit for the Item.	string
OPCHDA_STEPPED	stepped	Tells if the data shall be stepped and not interpolated (0 means interpolated).	boolean
OPCHDA_ARCHIVING	archiving	Tells if the server records data (0 means no recording).	boolean
OPCHDA_DERIVE_EQUATION	equation	Gives the equation used to calculate a derived item value. This attribute only used by servers that are capable of calculating Item values. The format of the string is server specific and intended for human readability only.	string
OPCHDA_SOURCE_NAME	sourceName	Gives the pathname for the corresponding DAIS DA Item.	string
OPCHDA_SOURCE_TYPE	sourceType	Gives what sort of source produces the data for the item. For an OPC DA server, this would be “OPC.” For non-OPC sources, the meaning of this field is server-specific.	string
OPCHDA_NORMAL_MAXIMUM	max	Gives the upper limit for the normal value range for the Item.	double or long
OPCHDA_NORMAL_MINIMUM	min	Gives the lower limit for the normal value range for the Item.	double or long
HDAIS_SOURCE_RESOURCE	sourceResource	Gives the ReourceID for the DAIS DA Item. Can together with the HDAIS_SOURCE_PROPERTY be used to create the source ItemID.	ResourceID
HDAIS_SOURCE_PROPERTY	sourceProperty	Gives the PropertyID for the DAIS DA Item. Can together with the HDAIS_SOURCE_RESOURCE be used to create the source ItemID.	PropertyID
HDAIS_ITEM_PATH_NAME	pathName	The pathname for the Item. An object may be moved in the naming hierarchy and the parent Node name might be changed. Hence the pathname for an object may change over time. This attribute corresponds to the OPCHDA_ITEMID.	string
HDAIS_PARENT_NAME	parentName	The parent Node name for the Item. This name may change over time if the parent Node is renamed.	string

HDAIS_INSERT_TIMES	insertTimes	The attribute contains the time_stamps when ItemValues was inserted or replaced. Several updates can happen at the same insert/update operation. The time when the operation was made is recorded in ItemAttributeIO::Value.time_stamp.	DateTimes

The following OPC constants are not defined in HDAIS for reasons listed below.

EnumValue	Description
OPCHDA_DATA_TYPE	The data type of an Item. The data type is given by a Property and is accessed via the Property::Home browse object. Methods that use the data type for filtering have a specific input parameter for this.
OPCHDA_DESCRIPTION	The description of an Item. The description is given by a Property and is accessed via the Property::Home browse object.
OPCHDA_NODE_NAME	The name of the computer hosting the OPC DA server (e.g., IP address). This information is supposed to be serviced by a naming service.
OPCHDA_PROCESS_NAME	The name of the process hosting the OPC DA server. This information is supposed to be serviced by a naming service.
OPCHDA_ITEMID	The pathname of the Item. The pathname is accessed via the Node::Home and Item::Home browse objects. Methods that use the pathname for filtering have a specific input parameter for this.

Home

Home is an object for browsing ItemAttributeDefinitions.

UnknownID

UnknownID is an exception telling that the provided AttributeID is unknown.

find ()

find() is a method that returns the Description from a known ItemAttributeDefinition.

The find() method has no corresponding OPC method.

Parameter	Description
item	The identification of the ItemAttributeDefinition (i.e., the AttributeID).
return	The Description for the ItemAttributeDefinitions.

find_all ()

find_all() is a method that returns all the Description for all ItemAttributeDefinition.

The find_all method corresponds to the OPC method IOPCHDA_Server::GetItemAttributes().

Parameter	Description
return	The Descriptions for all ItemAttributeDefinitions.

5.5.6 HDAISAggregate

5.5.6.1 HDAISAggregate overview

The interface is used to browse the AggregateDefinitions, refer to Chapter 3 for the information model.

5.5.6.2 HDAISAggregate IDL

```
//File HDAISAggregate.idl
#ifndef __HDAIS_AGGREGATE_IDL
#define __HDAIS_AGGREGATE_IDL
#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module Aggregate {

struct Description {
    AggregateID             id;
    string                  label;
    string                  descrip;};

typedef sequence<Description>   Descriptions;

const   AggregateIDOPCHDA_NOAGGREGATE           = 0x0000;
const   AggregateIDOPCHDA_INTERPOLATIVE         = 0x0001;
const   AggregateIDOPCHDA_TOTAL                 = 0x0002;
const   AggregateIDOPCHDA_AVERAGE               = 0x0003;
const   AggregateIDOPCHDA_TIMEAVERAGE           = 0x0004;
const   AggregateIDOPCHDA_COUNT                 = 0x0005;
const   AggregateIDOPCHDA_STDEV                 = 0x0006;
const   AggregateIDOPCHDA_MINIMUMACTUALTIME     = 0x0007;
const   AggregateIDOPCHDA_MINIMUM               = 0x0008;
const   AggregateIDOPCHDA_MAXIMUMACTUALTIME     = 0x0009;
const   AggregateIDOPCHDA_MAXIMUM               = 0x0010;
const   AggregateIDOPCHDA_START                 = 0x0011;
const   AggregateIDOPCHDA_END                   = 0x0012;
const   AggregateIDOPCHDA_DELTA                 = 0x0013;
const   AggregateIDOPCHDA_REGSLOPE              = 0x0014;
const   AggregateIDOPCHDA_REGCONST              = 0x0015;
const   AggregateIDOPCHDA_REGDEV                = 0x0016;
const   AggregateIDOPCHDA_VARIANCE              = 0x0017;
```

```
const    AggregateIDOPCHDA_RANGE                = 0x0018;
const    AggregateIDOPCHDA_DURATIONGOOD         = 0x0019;
const    AggregateIDOPCHDA_DURATIONBAD          = 0x0020;
const    AggregateIDOPCHDA_PERCENTGOOD          = 0x0021;
const    AggregateIDOPCHDA_PERCENTBAD           = 0x0022;
//const  AggregateIDOPCHDA_WORSTQUALITY         = 0x0023;
const    AggregateIDOPCHDA_ANNOTATIONS          = 0x0024;

interface Home
{
     exception UnknownID {string reason;};

     Description find (
          in   AggregateID                    id)
          raises (UnknownID);

     Descriptions find_all ();
};
};};};
#endif // __HDAIS_AGGREGATE_IDL
```

Description

Description is a struct describing an AggregateDefinition.

Member	Description
id	The identification of an AggregateDefinition.
label	The label of an AggregateDefinition.
descrip	The description of an AggregateDefinition.

AggregateID

AggregateID is defined in the HDAISCommon IDL (refer to Section 5.2, “HDAIS Common IDL,” on page 5-3). The table below lists the AggregateDefinitions and their Description member values.

EnumValue (id)	label	description
OPCHDA_NOAGGREGATE	noAggregate	Do not use any aggregate calculations.
OPCHDA_INTERPOLATIVE	interpolate	Interpolate the values with the given interval.
OPCHDA_TOTAL	total	Sum the values over the sample interval.
OPCHDA_AVERAGE	average	Calculate the average over the sample interval.
OPCHDA_TIMEAVERAGE	weightedAverage	Calculate the weighted average over the sample interval.
OPCHDA_COUNT	count	Calculate the number of values in the sample interval.

OPCHDA_STDEV	standardDeviation	Calculate the standard deviation over the sample interval.
OPCHDA_MINIMUMACTUALTIME	minWithTime	Retrieve the minimum value in the sample interval and the timestamp of the minimum value.
OPCHDA_MINIMUM	min	Retrieve the minimum value in the sample interval.
OPCHDA_MAXIMUMACTUALTIME	maxWithTime	Retrieve the maximum value in the sample interval and the timestamp of the maximum value.
OPCHDA_MAXIMUM	max	Retrieve the maximum value in the sample interval.
OPCHDA_START	start	Retrieve the value at the beginning of the sample interval.
OPCHDA_END	end	Retrieve the value at the end of the sample interval.
OPCHDA_DELTA	delta	Calculate the delta (difference) between the sample interval start and the end values.
OPCHDA_REGSLOPE	regressionSlope	Calculate the slope of the regression line for the sample interval.
OPCHDA_REGCONST	regressionConst	Calculate the regression constant (i.e., the value for the regression line at the start value).
OPCHDA_REGDEV	regressionDev	Calculate the standard deviation for the regression line over the sample interval.
OPCHDA_VARIANCE	variance	Calculate the variance over the sample interval.
OPCHDA_RANGE	range	Calculate the difference between the maximum and minimum values over the sample interval.
OPCHDA_DURATIONGOOD	durationGood	Calculate the time in seconds that the value had quality good within the sample interval.
OPCHDA_DURATIONBAD	durationBad	Calculate the time in seconds that the value had quality bad within the sample interval.
OPCHDA_PERCENTGOOD	durationGood%	Calculate the percentage of the time that the value had quality good within the sample interval.
OPCHDA_PERCENTBAD	durationBad%	Calculate the percentage of the time that the value had quality bad within the sample interval.
OPCHDA_ANNOTATIONS	countAnnotations	Retrieve the number of annotations in the sample interval.

find ()

find() is a method that returns the Description from a known AggregateDefinition.

The find() method has no corresponding OPC method.

Parameter	Description
item	The identification of the AggregateDefinition (i.e., the AggregateID).
return	The Description for the AggregateDefinition.

find_all ()

find_all() is a method that returns all the Description for all AggregateDefinitions.

The find_all() method corresponds to the OPC method IOPCHDA_Server::GetAggregates.

Parameter	Description
return	The Descriptions for all AggregateDefinitions.

5.6 Data Access (IO) Interfaces

This section describes the data access interfaces defined in the following IDLs:

- HDAISValueIO for ItemValues
- HDAISModifiedValueIO for ModifiedItemValues
- HDAISItemAttributesIO for ItemAttributeValues
- HDAISAnnotationIO for Annotations

5.6.1 HDAISValueIO

5.6.1.1 HDAISValueIO overview

The DAIS::HDA::ValueIO defines the object for access of ItemValue time series data. The interface is large and hence divided in several interfaces. Figure 5-2 shows the interfaces and their relation.

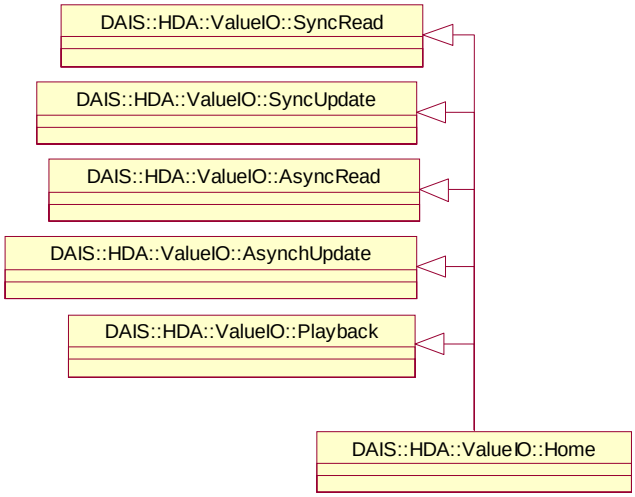

Figure 5-2 ValueIO Interfaces

The DAIS::HDA::ValueIO::Home object implements all interfaces described in this section as a singleton object.

The interfaces use handles for identification of Items and require that the interface DAIS::HDA::Connection has been used to establish associations between server and client handles.

DAIS::HDA::ValueIO::Home interface supports four different ways to read data and two ways to update data.

Ways to read data:

- Synchronous read where the data is received at return from the read method.
- Asynchronous read returning execution immediately to the client. The data is returned at the callback object.
- Subscription where updated data is sent spontaneously by the server through the callback object.
- Playback where data is sent through the callback object in pace with a simulated time.

Ways to update data:

- Synchronous update returning execution to the client once all data has been recorded.
- Asynchronous update returning execution immediately to the client. Once the updates are recorded the client gets a notification through the callback object.

The read operations can deliver data processed in the following ways:

- Raw data as recorded.

- Data processed according to an aggregate calculation. For a description of the aggregate calculations refer to Section 5.5.5, “HDAISItemAttribute,” on page 5-23.

Data updates can be made in the following ways:

- Insert where new ItemValues are inserted. If an ItemValue already exists (i.e., the same time stamp already exists) no insert is made.
- Replace where update ItemValues replace existing ItemValues. If no existing ItemValues correspond to the update ItemValues, the update ItemValues are not processed. The DAIS::HDA::ModifiedValueIO::Home interface can be used to specifically read the replaced ItemValues values.
- Insertreplace where update ItemValues replace existing ItemValues. If no existing ItemValues correspond to the update ItemValues, the update ItemValues are inserted as new.

The following IDLs define the interfaces:

- HDAISValueIOCommon defines common data declarations used by all interfaces.
- HDAISValueIO defines the object dealing with ItemValue access.
- HDAISSyncIO defines the interfaces for synchronous ItemValue access.
- HDAISAsyncIO defines the interfaces for asynchronous ItemValue access.
- HDAISPlayback defines the interfaces for playback of ItemValues.

The table below shows in what IDL the interfaces they define.

IDL	Interface
HDAISValueIO	DAIS::HDA::ValueIO::Home
HDAISSyncIO	DAIS::HDA::ValueIO::SyncRead
	DAIS::HDA::ValueIO::SyncUpdate
HDAISAsyncIO	DAIS::HDA::ValueIO::AsyncRead
	DAIS::HDA::ValueIO::AsyncUpdate
HDAISPlayback	DAIS::HDA::ValueIO::Playback

5.6.1.2 *HDAISValueIOCommon IDL*

```
// File: HDAISValueIOCommon.idl
#ifndef _HDAIS_VALUE_IO_COMMON_IDL
#define _HDAIS_VALUE_IO_COMMON_IDL

#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ValueIO {
```

```
struct TimeSerie {
      ClientItemHandle              client_handle;
      AggregateID                   aggregate_id;
      sequence<Value>               item_values;};

typedef sequence<TimeSerie>     TimeSeries;

struct Update {
      ServerItemHandle              server_handle;
      Value                         item_value;};

typedef sequence<Update>        Updates;

struct ProcessedRef {
      ServerItemHandle              server_handle;
      AggregateID                   aggregate_id;};

typedef sequence<ProcessedRef>      ProcessedRefs;

struct ItemValueRef {
      ServerItemHandle              server_handle;
      DateTime                      time_stamp;};

typedef sequence<ItemValueRef>      ItemValueRefs;

typedef ItemErrors                  UpdateResults;

};};};
#endif // _HDAIS_VALUE_IO_COMMON_IDL
```

TimeSerie

TimeSerie is a struct that describes a sequence of ItemValues for a particular Item.

Member	Description
client_handle	The client side handle that identifies the Item.
aggregate_id	The aggregate that was used when the data was retrieved. The methods that return raw data set the id to OPCHDA_NOAGGREGATE.
item_values	A sequence of Values that holds ItemValue data.

Update

Update is a struct that describes updates that shall be made for ItemValues at a particular Item.

Member	Description
server_handle	The server side handle that identifies the Item.

item_value	The value that shall be used to update the ItemValue. The time_stamp in the item_value is used to find the ItemValue to update.

ProcessedRef

ProcessedRef is a struct that references the Item and the aggregate calculation to use for retrieval of ItemValues.

Member	Description
server_handle	The server side handle that identifies the Item.
aggregate_id	The id of the aggregate calculation to use.

ItemValueRef

ItemValueRef is a struct that references the ItemValues to access.

Member	Description
server_handle	The server side handle that identifies the Item.
time_stamp	The time stamp for the ItemValue to access.

UpdateResults

UpdateResults is a definition of the result returned from insert/update operations. The actual operation performed (insert or update) is returned in the result together with Items that failed due to an error.

5.6.1.3 HDAISValueIO IDL

```
//File: HDAISValueIO.idl
#ifndef _HDAIS_VALUE_IO_IDL
#define _HDAIS_VALUE_IO_IDL
#include <HDAISSyncIO.idl>
#include <HDAISAsyncIO.idl>
#include <HDAISPlayback.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ValueIO {

typedef unsigned short                UpdateCapabilities;
const UpdateCapabilities OPCHDA_INSERTCAP          = 0x0001;
const UpdateCapabilities OPCHDA_REPLACECAP         = 0x0002;
//const UpdateCapabilities OPCHDA_INSERTREPLACECAP = 0x0004;
const UpdateCapabilities OPCHDA_DELETERAWCAP       = 0x0008;
const UpdateCapabilities OPCHDA_DELETEATTIMECAP    = 0x0010;
```

```
interface Home :
                SyncRead
                ,SyncUpdate
                ,AsyncRead
                ,AsyncUpdate
                ,Playback
{
    readonly attribute UpdateCapabilities       capabilities;
};
};};};
#endif // _HDAIS_VALUE_IO_IDL
```

UpdateCapabilities

UpdateCapabilities is a flag word that describes the update capabilities the server supports.

Flag	Description
OPCHDA_INSERTCAP	The server support insertion of new ItemValues.
OPCHDA_REPLACECAP	The server support replacement of ItemValues.
OPCHDA_DELETERAWCAP	The server support deletion of ItemValues in a time interval.
OPCHDA_DELETEATTIMECAP	The server support deletion of specified ItemValues.

Home

Home is an object that implements all the ItemValue access interfaces.

capabilities

capabilities is an attribute that tells the client what update capabilities the server supports.

The OPC corresponding OPC methods are IOPCHDA_SyncUpdate::QueryCapabilities() and IOPCHDA_AsyncUpdate::QueryCapabilities().

5.6.1.4 HDAISSyncIO IDL

```
//File: HDAISSyncIO.idl
#ifndef _HDAIS_SYNC_IO_IDL
#define _HDAIS_SYNC_IO_IDL
#include <HDAISValueIOCommon.idl>
#include <HDAISModifiedValueIO.idl>
#include <HDAISItemAttributeIO.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ValueIO {
```

```
interface SyncRead
{
    exception MaximumNumberOfValuesExceeded {string reason;};

    TimeSeries sync_read_raw (
        in   TimeInterval              interval,
        in   unsigned long             max_number_of_values,
        in   boolean                   bounds,
        in   ServerItemHandles         server_handles,
        out  ItemErrors                item_errors)
        raises (MaximumNumberOfValuesExceeded);

    TimeSeries sync_read_processed (
        in   TimeInterval              interval,
        in   DateTime                  sample_interval,
        in   ProcessedRefs             item_refs,
        out  ItemErrors                item_errors)
        raises (MaximumNumberOfValuesExceeded);

    TimeSeries sync_read_at_time (
        in   DateTimes                 time_stamps,
        in   ServerItemHandles         server_handles
        out  ItemErrors                item_errors);
};

interface SyncUpdate
{
    ItemErrors sync_insert (
        in   Updates                   item_values);

    ItemErrors sync_replace (
        in   Updates                   item_values);

    UpdateResults sync_insert_replace (
        in   Updates                   item_values);

    ItemErrors sync_delete_raw (
        in   TimeInterval              interval,
        in   ServerItemHandles         server_handles);

    ItemErrors sync_delete_at_time (
        in   DateTimes                 time_stamps,
        in   ServerItemHandles         server_handles);
};
};};};
#endif // _HDAIS_SYNC_IO_IDL
```

SyncRead

SyncRead is an interface for synchronous read of ItemValues.

MaximumNumberOfValuesExceeded

MaximumNumberOfValuesExceeded is an exception that tells the number of ItemValues requested by the client is larger than the server can handle.

sync_read_raw ()

sync_read_raw() is a method for synchronous read of the raw ItemValues.

The corresponding OPC method is IOPCHDA_SyncRead::ReadRaw().

Parameter	Description
interval	interval specifies the time interval for which to read ItemValues.
max_number_of_values	The maximum number of ItemValues to return for an Item.
bounds	If true, the bounding ItemValues shall be returned for the start and end times for the specified intervals. If a bounding ItemValue doesn't exist (no matching time stamp), an empty ItemValues is returned having the quality OPCHDA_NOBOUND.
server_handles	The server handles that identify the Items.
item_errors	A sequence reporting the items that was not read due to an error. Reported errors are: -- WARNING_MORE_DATA_THAN_REQUESTED -- WARNING_NO_DATA (in the interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.
return	TimeSeries for the found ItemValues are returned.

sync_read_processed ()

sync_read_processed() is a method for synchronous read of ItemValues with the returned data processed by an aggregate calculation.

The corresponding OPC method is IOPCHDA_SyncRead::ReadProcessed().

Parameter	Description
interval	interval specifies the time interval for which to read ItemValues.
sample_interval	The time interval where to pick ItemValues to use in the calculation. A calculated result is returned for each sample interval.
item_refs	The server handles and AggregateIDs to use as reference to Items and aggregate calculations.

item_errors	A sequence reporting the items that was not read due to an error. Reported errors are: -- WARNING_NO_DATA (in sample_interval) -- WARNING_MORE_EXTREEM_VALUES -- ERROR_AGGREGATE_NOT_AVAILABLE -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.
return	TimeSeries for the calculated ItemValues are returned.

sync_read_at_time ()

sync_read_at_time() is a method for synchronous read of ItemValues at specified times.

The corresponding OPC method is IOPCHDA_SyncRead::ReadAtTime().

Parameter	Description
time_stamps	The times for the ItemValues to read.
server_handles	The server handles that identifies the Items.
item_errors	A sequence reporting the Items that was not read due to an error. Reported errors are; -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.
return	TimeSeries for the found ItemValues are returned.

SyncUpdate

SyncUpdate is an interface for synchronous update of ItemValues.

sync_insert ()

sync_insert() is a method for synchronous insertion of new ItemValues.

The corresponding OPC method is IOPCHDA_SyncUpdate::Insert().

Parameter	Description
item_values	The descriptions of the new ItemValues to insert.

return	A sequence reporting the Items that was not updated due to an error. Reported errors are: -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_DATA_ALREADY_EXIST -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

sync_replace ()

sync_replace() is a method for synchronous replacement of already existing ItemValues.

The corresponding OPC method is IOPCHDA_ SyncUpdate::Replace().

Parameter	Description
item_values	The descriptions of the updates to make for the existing ItemValues.
return	A sequence reporting the Items that was not updated due to an error. Reported errors are: -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_DATA_DOES_NOT_EXIST -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

sync_insert_replace ()

sync_insert_replace() is a method for synchronous insertion of new or replacement of existing ItemValues.

The corresponding OPC method is IOPCHDA_ SyncUpdate::InsertReplace().

Parameter	Description
item_values	The descriptions of the updates to make for the existing ItemValues.
return	A sequence reporting the kind of updates made for successful ItemValues as well as errors for failed ItemValues. The reports are; -- RESULT_DATA_INSERTED -- RESULT_DATA_REPLACED -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

sync_delete_raw ()

sync_delete_raw() is a method for synchronous deletion of all ItemValues in a time interval.

The corresponding OPC method is IOPCHDA_ SyncUpdate::DeleteRaw().

Parameter	Description
interval	interval specifies the time interval for which to delete ItemValues.
server_handles	server_handles specifies the Items that shall be searched for ItemValues to delete.
item_refs	The server handles and AggregateIDs to use as reference to Items and aggregate calculations.
return	A sequence reporting the items that was not read due to an error. Reported errors are: -- WARNING_NO_DATA (in interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

sync_delete_at_time ()

sync_delete_at_time() is a method for synchronous delete of specified ItemValues.

The corresponding OPC method is IOPCHDA_ SyncUpdate::DeleteAtTime().

Parameter	Description
time_stamps	The times for the ItemValues to delete.
server_handles	The server handles that identify the Items.
return	A sequence reporting the Items that was not read due to an error. Reported errors are: -- WARNING_NO_DATA (for the specified times) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

5.6.1.5 HDAISAsyncIO IDL

```
//File: HDAISAsyncIO.idl
#ifndef _HDAIS_ASYNC_IO_IDL
#define _HDAIS_ASYNC_IO_IDL
#include <HDAISValueIOCommon.idl>
#include <HDAISModifiedValueIO.idl>
```

```
#include <HDAISItemAttributeIO.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ValueIO {

interface AsyncRead
{
    exception MaximumNumberOfValuesExceeded {string reason;};

    CancelID async_read_raw (
        in  TransactionID           transaction_id,
        in  TimeInterval            interval,
        in  unsigned long           max_number_of_values,
        in  boolean                 bounds,
        in  ServerItemHandles       server_handles)
        raises (MaximumNumberOfValuesExceeded);

    CancelID subscribe_raw (
        in  TransactionID           transaction_id,
        in  DateTime                start_time,
        in  DateTime                value_return_interval,
        in  ServerItemHandles       server_handles);

    CancelID async_read_processed (
        in  TransactionID           transaction_id,
        in  TimeInterval            interval,
        in  DateTime                sample_interval,
        in  ProcessedRefs           item_refs)
        raises (MaximumNumberOfValuesExceeded);

    CancelID subscribe_processed (
        in  TransactionID           transaction_id,
        in  DateTime                start_time,
        in  DateTime                sample_interval,
        in  ProcessedRefs           item_refs,
        in  unsigned long           no_samples_per_callback);

    CancelID async_read_at_time (
        in  TransactionID           transaction_id,
        in  DateTimes               time_stamps,
        in  ServerItemHandles       server_handles);

    void cancel (
        in  CancelID                cancel_id);
};

interface AsyncUpdate
{
    CancelID async_insert (
        in  TransactionID           transaction_id,
        in  Updates                 item_values);

    CancelID async_replace (
```

```
            in   TransactionID              transaction_id,
            in   Updates                    item_values);

        CancelID async_insert_replace (
            in   TransactionID              transaction_id,
            in   Updates                    item_values);

        CancelID async_delete_raw (
            in   TransactionID              transaction_id,
            in   TimeInterval               interval,
            in   ServerItemHandles          server_handles);

        CancelID async_delete_at_time (
            in   TransactionID              transaction_id,
            in   DateTimes                  time_stamps,
            in   ServerItemHandles          server_handles);
};

interface Callback
{
        void on_data_change (
            in   TransactionID              transaction_id,
            in   boolean                    all_quality_good,
            in   TimeSeries                 time_series,
            in   ItemErrors                 item_errors);

        void on_read_complete (
            in   TransactionID              transaction_id,
            in   boolean                    all_quality_good,
            in   TimeSeries                 time_series,
            in   ItemErrors                 item_errors);

        void on_update_complete (
            in   TransactionID              transaction_id,
            in   ClientItemHandles          client_handles,
            in   ItemErrors                 item_errors);
};
};};};
#endif // _HDAIS_ASYNC_IO_IDL
```

AsyncRead

AsyncRead is an interface for asynchronous read of ItemValues.

async_read_raw ()

async_read_raw() is a method for asynchronous read of the raw ItemValues. The result is returned by the server on the on_read_complete() method.

The corresponding OPC method is IOPCHDA_AsyncRead::ReadRaw().

Parameter	Description
transaction_id	A client assigned handle for the read operation.

interval	interval specifies the time interval for which to read ItemValues.
max_number_of_values	The maximum number of ItemValues to return for an Item.
bounds	If true, the bounding ItemValues shall be returned for the start and end times for the specified intervals. If a bounding ItemValue doesn't exist (no matching time stamp), an empty ItemValues is returned having the quality OPCHDA_NOBOUND.
server_handles	The server handles that identify the Items.
return	A cancellation id that the client may use to cancel the operation.

subscribe_raw ()

subscribe_raw() is a method to read existing ItemValues from a specified start time and continue to feed the client with new values that becomes available after transmission of the initially existing values. The result is returned by the server on the on_data_change() method.

The corresponding OPC method is IOPCHDA_AsyncRead::AdviseRaw().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
start_time	The time where to start the read operation.
value_return_interval	The time interval to return ItemValues. If no ItemValue has a matching time stamp, the nearest is picked.
server_handles	The server handles that identify the Items.
return	A cancellation id that the client may use to cancel the operation.

async_read_processed ()

async_read_processed() is a method for asynchronous read of ItemValues with the returned data processed by an aggregate calculation. The result is returned by the server on the on_read_complete() method.

The corresponding OPC method is IOPCHDA_AsyncRead::ReadProcessed().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
interval	interval specifies the time interval for which to read ItemValues.

sample_interval	The time interval where to pick ItemValues to use in the calculation. A calculated result is returned for each sample interval.
item_refs	The server handles and AggregateIDs to use as reference to Items and aggregate calculations.
return	A cancellation id that the client may use to cancel the operation.

subscribe_processed ()

subscribe_processed() is a method to read existing ItemValues from a specified start time and continue to feed the client with new values that become available after transmission of the initially existing values. The ItemValues are processed with an aggregate calculation before transmission. The result is returned by the server on the on_data_change() method.

The corresponding OPC method is IOPCHDA_AsyncRead::AdviseProcessed().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
start_time	The time where to start the read operation.
sample_interval	The time interval where to pick ItemValues to use in the calculation. A calculated result is returned for each sample interval.
item_refs	The server handles and AggregateIDs to use as reference to Items and aggregate calculations.
return	A cancellation id that the client may use to cancel the operation.

async_read_at_time ()

async_read_at_time() is a method for asynchronous read of ItemValues at specified times. The result is returned by the server on the on_read_complete() method.

The corresponding OPC method is IOPCHDA_AsyncRead::ReadAtTime().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
time_stamps	The times for the ItemValues to read.
server_handles	The server handles that identify the Items.
return	A cancellation id that the client may use to cancel the operation.

cancel ()

cancel() is a method to cancel ongoing asynchronous operations.

The corresponding OPC methods are IOPCHDA_AsyncRead::Cancel() and IOPCHDA_AsyncUpdate::Cancel().

Parameter	Description
cancel_id	The cancellation number from the server for the original asynchronous operation and used by the client to request the cancellation.

AsyncUpdate

AsyncUpdate is an interface for asynchronous update of ItemValues.

async_insert ()

async_insert() is a method for asynchronous insertion of new ItemValues. The result is returned by the server on the on_update_complete() method.

The corresponding OPC method is IOPCHDA_AsyncUpdate::Insert().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
item_values	The descriptions of the new ItemValues to insert.
return	A server assigned cancellation handle.

async_replace ()

async_replace() is a method for asynchronous replacement of already existing ItemValues. The result is returned by the server on the on_update_complete() method.

The corresponding OPC method is IOPCHDA_AsyncUpdate::Replace().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
item_values	The descriptions of the updates to make for the existing ItemValues.
return	A server assigned cancellation handle.

async_insert_replace ()

async_insert_replace() is a method for asynchronous insertion of new or replacement of existing ItemValues. The result is returned by the server on the on_update_complete() method.

The corresponding OPC method is IOPCHDA_AsyncUpdate::InsertReplace().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
item_values	The descriptions of the updates to make for the existing ItemValues.
return	A server assigned cancellation handle.

async_delete_raw ()

async_delete_raw() is a method for asynchronous deletion of all ItemValues in a time interval. The result is returned by the server on the on_update_complete() method.

The corresponding OPC method is IOPCHDA_AsyncUpdate::DeleteRaw().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
interval	interval specifies the time interval for which to delete ItemValues.
server_handles	server_handles specifies the Items that shall be searched for ItemValues to delete.
item_refs	The server handles and AggregateIDs to use as reference to Items and aggregate calculations.
return	A server assigned cancellation handle.

async_delete_at_time ()

async_delete_at_time() is a method for asynchronous delete of specified ItemValues. The result is returned by the server on the on_update_complete() method.

The corresponding OPC method is IOPCHDA_AsyncUpdate::DeleteAtTime().

Parameter	Description
time_stamps	The times for the ItemValues to delete.
server_handles	The server handles that identify the Items.
return	A server assigned cancellation handle.

Callback

Callback is an interface to be implemented by the client for the server to transmit responses to asynchronous calls from the client.

on_data_change ()

on_data_change() is a method the server will use to transmit responses to the asynchronous subscribe calls

- subscribe_raw
- subscribe processed

The corresponding OPC method is IOPCHDA_DataCallback::OnDataChange().

Parameter	Description
transaction_id	The client assigned handle for the subscribe operation returned by the server.
all_quality_good	All ItemValue qualities are good.
time_series	The TimeSeries for initially read or updated ItemValues.
item_errors	A sequence reporting the items that was not read and will not be further reported due to an error. Reported errors for all subscribe operations are: - -ERROR_BAD_RIGHTS - -ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER Reported errors specifically for subscribe_processed are: - -WARNING_MORE_EXTREEM_VALUES -- ERROR_AGGREGATE_NOT_AVAILABLE For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

on_read_complete ()

on_read_complete() is a method the server will use to transmit responses to the asynchronous read calls

- async_read_raw()
- async_read_processed()
- async_read_at_time()

The corresponding OPC method is IOPCHDA_DataCallback::OnReadComplete().

Parameter	Description
transaction_id	The client assigned handle for the read operation returned by the server.
all_quality_good	All ItemValue qualities are good.
time_series	The TimeSeries for initially read or updated ItemValues.

item_errors	A sequence reporting the items that was not read due to an error. Reported errors for all read operations are: -- WARNING_NO_DATA (in sample_interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER Reported error specifically for read_raw is: -- WARNING_MORE_DATA_THAN_REQUESTED Reported errors specifically for read_processed are: -- WARNING_MORE_EXTREEM_VALUES -- ERROR_AGGREGATE_NOT_AVAILABLE For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

on_update_complete ()

on_update_complete() is a method the server will use to transmit responses to the asynchronous update calls

- async_insert()
- async_replace()
- async_insert_replace()
- async_delete_raw()
- async_delete_at_time()

The corresponding OPC method is IOPCHDA_DataCallback::OnUpdateComplete().

Parameter	Description
transaction_id	The client assigned handle for the update operation returned by the server.
client_handles	The client handles for the successfully updated ItemValues.

item_errors	A sequence reporting the items that was not updated due to an error. Reported errors for all subscribe operations are: -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER Reported error specifically for insert is: -- ERROR_DATA_ALREADY_EXIST Reported error specifically for replace is: -- ERROR_DATA_DOES_NOT_EXIST Reported errors specifically for insert_replace are: -- RESULT_DATA_INSERTED -- RESULT_DATA_REPLACED Reported error specifically for delete_raw is: -- WARNING_NO_DATA (in interval) Reported error specifically for delete_at_time is: -- WARNING_NO_DATA (for the specified times) For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

5.6.1.6 *HDAISPlayback IDL*

```
//File: HDAISPlayback.idl
#ifndef _HDAIS_PLAYBACK_IDL
#define _HDAIS_PLAYBACK_IDL
#include <HDAISValueIOCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ValueIO {

interface Playback
{
    exception MaximumNumberOfValuesExceeded {string reason;};

    CancelID playback_raw_with_update (
        in   TransactionID          transaction_id,
        in   TimeInterval           initalization_interval,
        in   unsigned long          max_number_of_values,
        in   DateTime               duration,
        in   DateTime               playback_interval,
        in   ServerItemHandles      server_handles)
        raises (MaximumNumberOfValuesExceeded);

    CancelID playback_processed_with_update (
        in   TransactionID          transaction_id,
```

```
            in   TimeInterval              initalization_interval,
            in   DateTime                  sample_interval,
            in   unsigned long             number_of_sample_intervals,
            in   DateTime                  playback_interval,
            in   ProcessedRefs             item_refs)
            raises (MaximumNumberOfValuesExceeded);
};

interface PlaybackCallback
{
     void on_playback (
            in   TransactionID             transaction_id,
            in   boolean                   all_quality_good,
            in   TimeSeries                time_series,
            in   ItemErrors                item_errors);
};
};};};
#endif // _HDAIS_PLAYBACK_IDL
```

Playback

Playback is an interface to start play back of ItemValues from the server.

MaximumNumberOfValuesExceeded

MaximumNumberOfValuesExceeded is an exception that tells the number of ItemValues requested by the client is larger than the server can handle.

playback_raw_with_update ()

playback_raw_with_update() is a method that initially transmits ItemValues according to the requested initialization interval. After initialization ItemValues are transmitted according to a simulated time. ItemValues are transmitted as they change at the simulated time. The corresponding OPC method is IOPCHDA_Playback::ReadRawWithUpdate().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
initialization_interval	initialization_interval specifies the time interval for which to read ItemValues for initialization.
max_number_of_values	The maximum number of ItemValues to return for an Item.
duration	The duration time for the simulated time. This period starts after the interval end time.
playback_interval	The time between transmissions of updated ItemValues.
server_handles	server_handles specifies the Items that shall be played back.
return	A server assigned cancellation handle.

playback_processed_with_update ()

playback_processed_with_update() is a method that initially transmits ItemValues according to the requested initialization interval. After initialization ItemValues are transmitted according to a simulated time. ItemValues are transmitted as they change at the simulated time. The speed of the simulated time can be controlled.

The corresponding OPC method is IOPCHDA_Playback::ReadProcessedWithUpdate().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
initialization_interval	initialization_interval specifies the time interval for which to read ItemValues for initialization.
sample_interval	The time interval where to pick ItemValues to use in the calculation. A calculated result is created for each sample interval.
number_of_sample_intervals	The number of sample intervals from which to collect changed ItemValues for transmission to the client.
playback_interval	The time between transmissions of updated ItemValues at real time.
item_refs	The server handles and AggregateIDs to use as reference to Items and aggregate calculations.
return	A server assigned cancellation handle.

By increasing the number_of_sample_intervals the play back speed will increase as ItemValues for more recorded times will be transmitted in the same playback_interval. Figure 5-3 shows how the play back speed can be controlled by changing the number_of_sample_intervals.

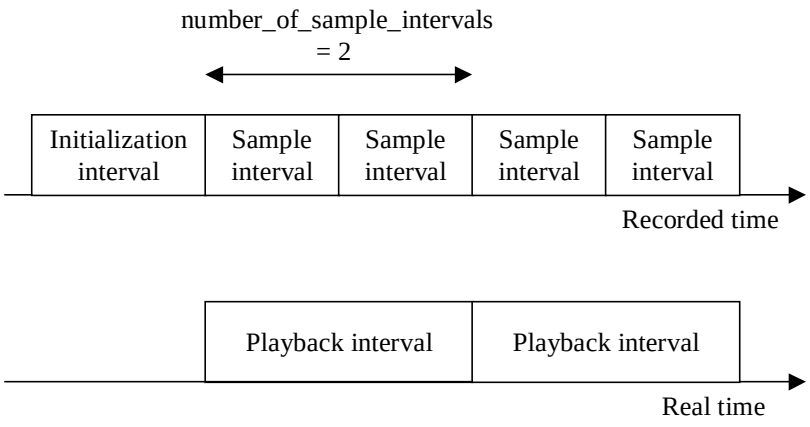

Figure 5-3 Play Back Example

In Figure 5-3 the sample_interval is set to half of the playback_interval and number_of_sample_intervals is set to two. This will result in a play back speed equal to real time. If number_of_sample_intervals is set to one, the play back speed will be half of real time speed and if it is set to four it will be twice the real time speed.

PlaybackCallback

PlaybackCallback is an interface to be implemented by the client for the server to transmit play back data to the client.

on_playback ()

on_playback() is a method the server will use to transmit responses to the asynchronous play back calls

- playback_raw_with_update()
- playback_processed_with_update()

The corresponding OPC method is IOPCHDA_DataCallback::OnPlayback().

Parameter	Description
transaction_id	The client assigned handle for the play back operation returned by the server.
all_quality_good	All ItemValue qualities are good.
time_series	The TimeSeries for initially read or as play back proceed updated ItemValues.
item_errors	A sequence reporting the items that were not updated due to an error. Reported errors for all subscribe operations are: -- WARNING_NO_DATA (in initialization interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER Reported error specifically for playback_processed_with_update is: -- ERROR_AGGREGATE_NOT_AVAILABLE For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

5.6.2 HDAISModifiedValueIO

5.6.2.1 HDAISModifiedValueIO Overview

The DAIS::HDA::ModifiedValueIO::Home interface has methods for reading of ItemValues as they appeared before the modification was made. It is also possible to read deleted ItemValues. Note that modified values are accessed via the DAIS::HDA::ValueIO interfaces except deleted values.

DAIS::HDA::ModifiedValueIO::Home is implemented as a singleton object.

The interfaces use handles for identification of Items and require that the interface DAIS::HDA::Connection has been used to establish associations between server and client handles.

5.6.2.2 HDAISModifiedValueIO IDL

```
//File: HDAISModifiedValueIO.idl
#ifndef _HDAIS_MODIFIED_VALUE_IO_IDL
#define _HDAIS_MODIFIED_VALUE_IO_IDL
#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ModifiedValueIO {

typedef unsigned short                  ModificationType;
const ModificationType  OPCHDA_INSERT          = 1;
const ModificationType  OPCHDA_REPLACE         = 2;
const ModificationType  OPCHDA_INSERTREPLACE   = 3;
const ModificationType  OPCHDA_DELETE          = 4;

struct Modification {
    Value                       item_value;
    DateTime                    modification_time;
    ModificationType            modification_type;
    string                      user_name;};

struct TimeSerie {
    ClientItemHandle            client_handle;
    AggregateID                 aggregate_id;
    sequence<Modification>      modifications;};

typedef sequence<TimeSerie>     TimeSeries;

interface Sync
{
    TimeSeries sync_read_modified (
        in   TimeInterval               interval,
        in   unsigned long              max_number_of_values,
        in   ServerItemHandles          server_handles,
        out  ItemErrors                 item_errors);
```

```
};

interface Async
{
    CancelID async_read_modified (
        in   TransactionID          transaction_id,
        in   TimeInterval           interval,
        in   unsigned long          max_number_of_values,
        in   ServerItemHandles      server_handles);

    void cancel (
        in   CancelID               cancel_id);
};

interface Home : Sync, Async
{};

interface Callback
{
    void on_read_modified_complete (
        in   TransactionID          transaction_id,
        in   boolean                all_quality_good,
        in   TimeSeries             time_series,
        in   ItemErrors             item_errors);
};
};};};
#endif // _HDAIS_MODIFIED_VALUE_IO_IDL
```

ModificationType

ModificationType is an enumeration used to indicate the type of modification made.

EnumValue	Description
OPCHDA_INSERT	The update operation was an insert.
OPCHDA_REPLACE	The update operation was a replace.
OPCHDA_INSERTREPLACE	The update operation was an insert_replace.
OPCHDA_DELETE	The update operation was a delete.

Modification

Modification is a struct describing a modification.

Member	Description
item_value	The old ItemValue as it appeared before the modification. The following cases exist: -- OPCHDA_INSERT; no old item_value. -- OPCHDA_REPLACE; old item_value exists. -- OPCHDA_INSERTREPLACE old item_value may or may not exist -- OPCHDA_DELETE; old item_value exists.

modification_time	The time when the modification was made.
modification_type	The type of modification made.
user_name	The name of the person that made the modification.

TimeSerie

TimeSerie is a struct that describes a sequence of Modifications made to ItemValues for a particular Item.

Member	Description
client_handle	The client side handle that identifies the Item.
aggregate_id	The aggregate that was used when the data was retrieved. The methods that return raw data set the id to OPCHDA_NOAGGREGATE.
modifications	A sequence of Modifications.

Sync

Sync is an interface for synchronous read operations.

sync_read_modified ()

sync_read_modified() is a method for synchronous read of modified ItemValues.

The corresponding OPC method is IOPCHDA_SyncRead::ReadModified().

Parameter	Description
interval	interval specifies the time interval for which to read ItemValues.
max_number_of_values	The maximum number of ItemValues to return for an Item.
server_handles	The server handles that identify the Items.
item_errors	A sequence reporting the items that were not read due to an error. Reported errors are: -- WARNING_MORE_DATA_THAN_REQUESTED -- WARNING_NO_DATA (in the interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.
return	TimeSeries for the found modified ItemValues are returned.

Async

Async is an interface for asynchronous read operations.

async_read_modified ()

async_read_modified() is a method for asynchronous read of modified ItemValues. The server returns the result on the on_read_modified_complete () method.

The corresponding OPC method is IOPCHDA_AsyncRead::ReadModified().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
interval	interval specifies the time interval for which to read ItemValues.
max_number_of_values	The maximum number of ItemValues to return for an Item.
server_handles	The server handles that identifies the Items.
return	A server assigned cancellation handle.

cancel ()

cancel() is a method to cancel ongoing asynchronous read operations for modified values.

The corresponding OPC method is IOPCHDA_AsyncRead::Cancel().

Parameter	Description
cancel_id	The cancellation number from the server for the original asynchronous operation and used by the client to request the cancellation.

Home

Home is a singleton object for reading of modified ItemValues.

Callback

Callback is an interface to be implemented by the client for the server to transmit responses to the asynchronous read operations from the client.

on_read_modified_complete ()

on_read_modified_complete() is a method the server will use to transmit responses to async_read_modified() calls.

The corresponding OPC method is IOPCHDA_DataCallback::OnReadModifiedComplete().

Parameter	Description
transaction_id	The client assigned handle for the read operation returned by the server.
all_quality_good	All ItemValue qualities are good.
time_series	TimeSeries for the found modified ItemValues.
item_errors	A sequence reporting the items that were not read due to an error. Reported errors are: -- WARNING_MORE_DATA_THAN_REQUESTED -- WARNING_NO_DATA (in the interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

5.6.3 HDAISItemAttributeIO

5.6.3.1 HDAISItemAttributeIO overview

The DAIS::HDA::ItemAttributeIO::Home interface has methods for transfer of ItemAttributeValue time series data and is implemented as a singleton object.

The interfaces use handles for identification of Items and require that the interface DAIS::HDA::Connection has been used to establish associations between server and client handles.

5.6.3.2 HDAISItemAttributeIO IDL

```
//File: HDAISItemAttributeIO.idl
#ifndef _HDAIS_ITEM_ATTRIBUTE_IO_IDL
#define _HDAIS_ITEM_ATTRIBUTE_IO_IDL
#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module ItemAttributeIO {

typedef short SimpleValueType;
const SimpleValueType DAIS_SIMPLE_VALUE_TYPE   = 1;
const SimpleValueType ITEMID_VALUE_TYPE        = 2;

union SimpleValue switch( SimpleValueType )
{
    case DAIS_SIMPLE_VALUE_TYPE: DAIS::SimpleValue simple_value;
```

```
        case ITEMID_VALUE_TYPE: ItemID item_id_value;};

struct Value {
        SimpleValue                     simple_value;
        DateTime                        time_stamp;};

typedef sequence<Value>                 Values;

struct TimeSerie {
        AttributeID                     attribute_id;
        ClientItemHandle                client_handle;
        Values                          attribute_values;};

typedef sequence<TimeSerie>             TimeSeries;

interface Sync
{
        TimeSeries sync_read_attribute (
                in   TimeInterval       interval,
                in   ServerItemHandle   server_handle,
                in   AttributeIDs       attribute_ids,
                out  ItemErrors         item_errors);};

interface Async
{
        CancelID async_read_attribute (
                in   TransactionID      transaction_id,
                in   TimeInterval       interval,
                in   ServerItemHandle   server_handle,
                in   AttributeIDs       attribute_ids);

        void cancel (
                in   CancelID           cancel_id);};

interface Home : Sync, Async
{};

interface Callback
{
        void on_read_attributes_complete (
                in   TransactionID      transaction_id,
                in   TimeSeries         time_series,
                in   ItemErrors         item_errors);};
};};};
#endif // _HDAIS_ITEM_ATTRIBUTE_IO_IDL
```

SimpleValueType

SimpleValueType is an extension of the SimpleValueType as defined in the DAFDescriptions IDL. The following values are defined.

Enum	Description
DAIS_SIMPLE_VALUE_TYPE	The SimpleValue as used by DAIS and DAF.
ITEMID_VALUE_TYPE	The extension of SimpleValue with the ItemID.

SimpleValue

SimpleValue is a union that includes the original DAIS/DAF simple value extended with ItemID and ItemIDs.

Value

Value is a struct that holds the ItemAttributeValue data.

Member	Description
simple_value	The attribute value.
time_stamp	The time stamp when the attribute value was inserted or updated.

TimeSerie

TimeSerie is a struct that describes a sequence of ItemAttributeValues for a particular ItemAttribute and Item.

Member	Description
attribute_id	The identification of the ItemAttribute.
client_handle	The client side handle that identifies the Item.
attribute_values	A sequence of ItemAttribute values.

Sync

Sync is an interface for synchronous read operations.

sync_read_attribute ()

sync_read_attribute() is a method for synchronous read of ItemAttributeValues.

The corresponding OPC method is IOPCHDA_SyncRead::ReadAttribute ().

Parameter	Description
interval	interval specifies the time interval for which to read ItemAttributeValues.
server_handle	The server handle that identifies the Item.

attribute_ids	The identifications of ItemAttributeValues to read.
item_errors	A sequence reporting the items that were not read due to an error. Reported errors are: -- WARNING_NO_DATA (in the interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.
return	TimeSeries for the found ItemAttributeValues are returned.

Async

Async is an interface for asynchronous read operations.

async_read_attribute ()

async_read_attribute() is a method for asynchronous read of ItemAttributeValues.

The corresponding OPC method is IOPCHDA_AsyncRead::ReadAttribute().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
interval	interval specifies the time interval for which to read ItemAttributeValues.
server_handle	The server handle that identifies the Item.
attribute_ids	The identifications of ItemAttributeValues to read.
return	A server assigned cancellation handle.

cancel ()

cancel() is a method to cancel ongoing asynchronous read operations.

The corresponding OPC method is IOPCHDA_AsyncRead::Cancel().

Parameter	Description
cancel_id	The cancellation number from the server for the original asynchronous operation and used by the client to request the cancellation.

Home

Home is a singleton object for reading of ItemAttributeValues.

Callback

Callback is an interface to be implemented by the client for the server to transmit responses to the asynchronous read operations from the client.

on_read_attributes_complete ()

on_read_attributes_complete () is a method the server will use to transmit responses to async_read_attribute () calls.

The corresponding OPC method is IOPCHDA_DataCallback::OnReadAttributesComplete().

Parameter	Description
transaction_id	The client assigned handle for the read operation returned by the server.
time_series	TimeSeries for the found ItemAttributeValues.
item_errors	A sequence reporting the items that were not read due to an error. Reported errors are: -- WARNING_NO_DATA (in the interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

5.6.4 HDAISAnnotationIO

5.6.4.1 HDAISAnnotationIO Overview

The DAIS::HDA::ItemAnnotationIO::Home interface has methods for transfer of Annotations time series data and is implemented as a singleton object.

The interfaces use handles for identification of Items and require that the interface DAIS::HDA::Connection has been used to establish associations between server and client handles.

5.6.4.2 HDAISAnnotationIO IDL

```
//File: HDAISAnnotation.idl
#ifndef _HDAIS_ANNOTATION_IDL
#define _HDAIS_ANNOTATION_IDL
#include <HDAISCommon.idl>
#pragma prefix "omg.org"

module DAIS {
module HDA {
module AnnotationIO {
```

```
typedef unsigned short                          AnnotCapabilities;
const AnnotCapabilities OPCHDA_READANNOTATIONCAP      = 0x0001;
const AnnotCapabilities OPCHDA_INSERTANNOTATIONCAP    = 0x0002;

struct Description {
    DateTime                    time_stamp;
    DateTime                    entry_time;
    string                      text;
    DAFDescriptions::Blob       a_blob;
    string                      user_name;};

struct TimeSerie {
    ClientItemHandle            client_handle;
    sequence<Description>       annotations;};

typedef sequence<TimeSerie>     TimeSeries;

struct Update {
    ServerItemHandle            server_handle;
    sequence<Description>       annotations;};

interface Sync
{
    TimeSeries sync_read (
        in   TimeInterval           interval,
        in   ServerItemHandles      server_handles,
        out  ItemErrors             item_errors);

    ItemErrors sync_insert (
        in   Update                 annotation_update);
};

interface Async
{

    CancelID async_read (
        in   TransactionID          transaction_id,
        in   TimeInterval           interval,
        in   ServerItemHandles      server_handles);

    CancelID async_insert (
        in   TransactionID          transaction_id,
        in   Update                 annotation_update);

    void cancel (
        in   CancelID               cancel_id);
};

interface Home : Sync, Async
{
    readonly attribute AnnotCapabilitiescapabilities;
};

interface Callback
{
```

```
        void on_read_annotation_complete (
            in   TransactionID              transaction_id,
            in   TimeSeries                 time_series,
            in   ItemErrors                 item_errors);

        void on_insert_annotation_complete (
            in   TransactionID              transaction_id,
            in   ClientItemHandles          client_handles,
            in   ItemErrors                 item_errors);
};
};};};
#endif // _HDAIS_ANNOTATION_IDL
```

AnnotCapabilities

AnnotCapabilities is a flag word that describes the annotation capabilities the server supports.

Flag	Description
OPCHDA_READANNOTATIONCAP	The server support read of Annotations.
OPCHDA_INSERTANNOTATIONCAP	The server support insertion of Annotations.

Description

Description is a struct that describes an Annotation.

Member	Description
time_stamp	The time_stamp for the annotated ItemValue.
entry_time	The time when the Annotation was entered.
text	The annotation text.
a_blob	A blob that may contain any kind of data (e.g., a picture, a movie, a sound clip, etc.).
user_name	The name of the person that entered the Annotation.

TimeSerie

TimeSerie is a struct that describes a sequence of Annotations for a particular Item.

Member	Description
client_handle	The client side handle that identifies the Item.
annotations	A sequence of Annotation descriptions.

Update

Update is a struct that describes Annotation updates that shall be made for ItemValues at a particular Item.

Member	Description
server_handle	The server handle that identifies the Item.
annotations	A sequence of Annotation descriptions to be used in the update.

Sync

Sync is an interface for synchronous read or update operations.

sync_read ()

sync_read is a method for synchronous read of Annotations.

The corresponding OPC method is IOPCHDA_SyncAnnotations::Read().

Parameter	Description
interval	interval specifies the time interval for which to read Annotations.
server_handles	The server handles that identify the Items.
item_errors	A sequence reporting the items that were not read due to an error. Reported errors are: -- WARNING_NO_DATA (in the interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.
return	TimeSeries for the found Annotations are returned.

sync_insert ()

sync_insert is a method for synchronous insert of Annotations. If an Annotation already exists, it is replaced.

The corresponding OPC method is IOPCHDA_SyncAnnotations:: Insert().

Parameter	Description
annotation_update	The descriptions of the Annotations to be inserted.

return	A sequence reporting the items that were not read due to an error. Reported errors are: -- WARNING_NO_DATA, see note below -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

Note – WARNING_NO_DATA means that ItemValues to be annotated were not found (i.e., ItemValues corresponding to Description::time_stamps was not found). Other ItemValues for the Item may still have been annotated.

Async

Async is an interface for asynchronous read or update operations.

async_read ()

async_read () is a method for asynchronous read of Annotations.

The corresponding OPC method is IOPCHDA_AsyncAnnotations::Read().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
interval	interval specifies the time interval for which to read Annotations.
server_handles	The server handles that identify the Items.
return	A server assigned cancellation handle.

async_insert ()

async_insert is a method for asynchronous insert of Annotations. If an Annotation already exists, it is replaced.

The corresponding OPC method is IOPCHDA_AsyncAnnotations::Insert().

Parameter	Description
transaction_id	A client assigned handle for the read operation.
annotation_update	The descriptions of the Annotations to be inserted.
return	A server assigned cancellation handle.

cancel ()

cancel() is a method to cancel ongoing asynchronous read or insert operations.

The corresponding OPC method is IOPCHDA_AsyncAnnotations::Cancel().

Parameter	Description
cancel_id	The cancellation number from the server for the original asynchronous operation and used by the client to request the cancellation.

Home

Home is a singleton object for access of Annotations.

capabilities

capabilities is an attribute that tells the client what annotation capabilities the server supports.

The OPC methods that correspond to the flag word are IOPCHDA_SyncAnnotations::QueryCapabilities () and IOPCHDA_ AsyncAnnotations::QueryCapabilities().

Callback

Callback is an interface to be implemented by the client for the server to transmit responses to the asynchronous read or insert operations from the client.

on_read_annotation_complete ()

on_read_annotation_complete() is a method the server will use to transmit responses to async_read() calls.

The corresponding OPC method is IOPCHDA_DataCallback::OnReadAnnotations().

Parameter	Description
transaction_id	The client assigned handle for the read operation returned by the server.
time_series	TimeSeries for the found Annotations.
item_errors	A sequence reporting the items that were not read due to an error. Reported errors are: -- WARNING_NO_DATA (in the interval) -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

on_insert_annotation_complete ()

on_insert_annotation_complete() is a method the server will use to transmit responses to async_insert() calls.

The corresponding OPC method is IOPCHDA_DataCallback::OnInsertAnnotations().

Parameter	Description
transaction_id	The client assigned handle for the read operation returned by the server.
client_handles	The client side handles for the Items that have been successfully updated with Annotations.
item_errors	A sequence reporting the items that were not updated due to an error. Reported errors are: -- WARNING_NO_DATA, see note below -- ERROR_BAD_RIGHTS -- ERROR_INVALID_DAIS_HANDLE -- ERROR_INTERNAL_SERVER For the error codes refer to the DAISCommon IDL [2] and Section 5.2, “HDAIS Common IDL,” on page 5-3.

Note – WARNING_NO_DATA means that ItemValues to be annotated were not found (i.e., ItemValues corresponding to Description::time_stamps was not found). Other ItemValues for the Item may still have been annotated.

5.7 Basic Sequencing

The basic sequencing is shown in Figure 5-4.

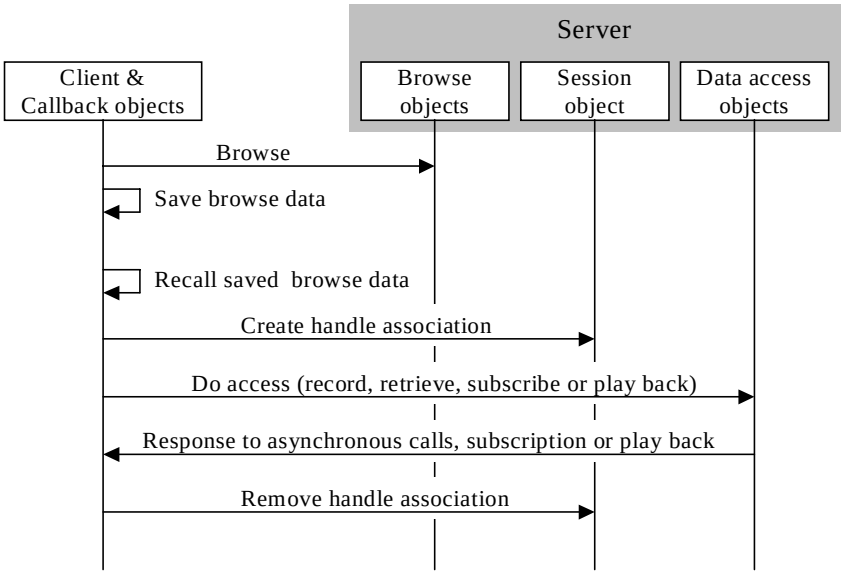

Figure 5-4 Basic Sequencing

Before a client accesses data it must know what Items exist in the server. This knowledge can be acquired by using the browse interfaces (Node::Home, Item::Home etc) or in some other way. Based on the Items configured in the server the client usually saves a collection of Item descriptions for later use in an access. Item collections may be related to a trend displays, report displays, scheduling programs, etc.

At some point in time a client recalls the Item descriptions and uses the Connection interface at the Session object to establish associations between server and client handles.

When associations are successfully created the client uses the server handles in the data access interfaces (e.g., ValueIO::SyncRead, ValueIO::AsyncRead, ValueIO::Playback).

For asynchronous subscription and playback calls the server returns data to the client on the callback interfaces using the client handles as Item identifications.

References A

A.1 List of References

1. OMG HDAIS RFP utility/2002-01-03
2. OMG Data Acquisition from Industrial Systems specification (DAIS) from the DAIS FTF utility/02-05-04.
3. OMG Utility Management Systems Data Access Facility (DAF) formal/01-06-01 and the DAIS FTF revision utility/02-05-03
4. OPC Overview; www.opcfoundation.org.
5. OPC Data access version 2.05; www.opcfoundation.org.
6. OPC Alarm and events 1.03; www.opcfoundation.org.
7. OPC Access to Historical data 1.1; www.opcfoundation.org.
8. Guidelines for Control Center APIs; EPRI TR-106324
9. Energy management system APIs; IEC draft IEC 61970-301.
10. Structuring principles and reference designations; IEC standard IEC 61346-1.

A

OMG IDL *B*

The complete IDL can be found in the zip archive mantis/2002-10-04. The following URL should be used to access this file:

http://www.omg.org/cgi-bin/doc?mantis/02-10-04

As the HDAIS IDL has dependencies both to the DAF and the DAIS IDLs a flat file structure would is unstructured and difficult to navigate. Hence the IDL files have been sorted in a directory structure with a common root for common IDL as DAFIdentifiers IDL and DAFDescriptions IDL. The next level contains DAF and DAIS specific IDLs. DAIS further divided in HDA for HDAIS and DAAE for DAIS DA and A&E. This structure has been included in the zip archive. To simplify compilation of the IDL HDA.bat is included for convenience.

Glossary

DAF - The Utility Management System Data Access Facility.

DAF Client - A program or software entity that uses the DAF interfaces to obtain information. Abbreviated to client in most of this specification.

Data Provider - An implementation of the DAF. That is, a program or software entity that supplies information via the DAF interfaces. Also referred to as a DAF server or just a server.

DMS - A Distribution Management System. This is a UMS for operating an electric power sub-transmission and distribution system.

EMS - An Energy Management System. This is a UMS for operating an electric power main transmission and/or production system.

EPRI - Electric Power Research Institute. A power industry body that is engaged in an effort to define APIs and data models for EMS systems and applications.

EPRI CIM - The EPRI Common Information Model. A data model defined in UML that can be used to describe power systems and related concepts.

OPC - OLE for Process Control.

PLC - Programmed Logic Controller, a device that controls an item or items of equipment. A PLC may transmit data it gathers to a UMS and receive control commands from the UMS. In this case it fills a role similar to an RTU.

Power System - The integrated facilities and resources that produce, transmit and/or distribute electric energy.

RDF - Resource Description Framework. A model of data that has been defined by a W3C recommendation and is used in conjunction with XML notation.

RTU - Remote Terminal Unit, a device located at a (usually) remote site that connects equipment with a central UMS. An RTU gathers data from equipment, and transmits that data back to the UMS. It also receives commands from the UMS and controls the equipment.

SCADA - Supervisory Control and Data Acquisition, a system that gives operators oversight and control of geographically dispersed facilities.

UML - Unified Modeling Language. The OMG standard modeling language, which has been used to define the EPRI Common Information Model.

UMS - Utility Management System, a control system that incorporates simulation and analysis applications used by a water, gas or electric power utility for operations or operational decision support.

WQEMS - A Water Quality and Energy Management System. This is a UMS for operating water supply and/or waste water systems.

XML - Extensible Markup Language. A generic syntax defined by a W3C recommendation that can be used to represent UMS data and schema, among other things.