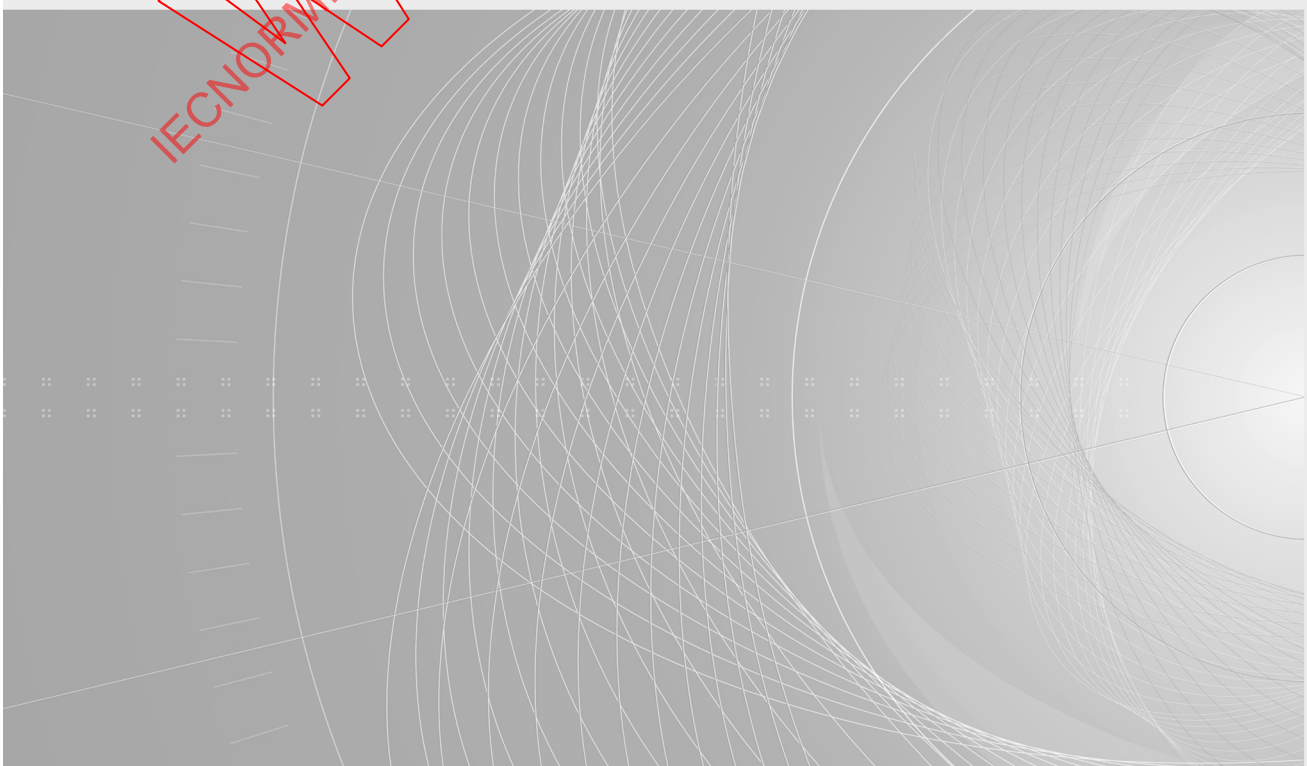
IEC 61970-403

Edition 1.0 2008-06

INTERNATIONAL STANDARD

Energy management system application program interface (EMS-API) – Part 403: Generic data access



IEC 61970-403:2008(E)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 61970-403

Edition 1.0 2008-06

INTERNATIONAL STANDARD

Energy management system application program interface (EMS-API) – Part 403: Generic data access

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE W

ICS 33.200

ISBN 2-8318-9851-X

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ENERGY MANAGEMENT SYSTEM APPLICATION PROGRAM INTERFACE (EMS-API) –

Part 403: Generic data access

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61970-403 has been prepared by IEC technical committee 57: Power systems management and associated information exchange.

The text of this standard is based on the following documents:

FDIS	Report on voting
57/929/FDIS	57/948/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

A list of all parts of the IEC 61970 series, under the general title *Energy Management System Application Program Interface (EMS-API)*, can be found on the IEC website.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

A bilingual version of this publication may be issued at a later date.

INTRODUCTION

This standard is one of the IEC 61970 series parts that define services for utility operational systems. This standard is based upon the work of the Electric Power Research Institute (EPRI) Control Center API (CCAPI) research project (RP-3654-1).

The IEC 61970-4xx series specifies a set of interfaces that a component (or application) should implement to be able to exchange information with other components and/or access publicly available data in a standard way. The 61970-4xx series component interfaces describe the specific event types and message contents that can be used by applications independent of any particular component technology. The implementation of these messages using a particular component technology is described in the 61970-5xx series of documents. Thus, IEC 61970-4xx documents describe a Platform Independent Model (PIM), while IEC 61970-5xx documents describe a Platform Specific Model (PSM).

IEC 61970-403 Generic Data Access (GDA) defines services that are needed to access public entity objects for the power system domain that are defined in the IEC 61970-3xx series: Common Information Model (CIM). GDA permits a client to access data maintained by another component (either an application or database) or system without any knowledge of the logical schema used for internal storage of the data. Knowledge of the existence of the common model is sufficient.

This request and reply oriented service is intended for synchronous, non-real time access of complex data structures as opposed to high-speed data access of SCADA data, for example, which is provided by IEC 61970-404, High Speed Data Access. An example where the GDA would be used is for bulk data access of a persistent store to initialise an analysis application with the current state of a power system network, and then storage of the results with notification.

ENERGY MANAGEMENT SYSTEM APPLICATION PROGRAM INTERFACE (EMS-API) –

Part 403: Generic data access

1 Scope

This International Standard provides a generic request/reply-oriented data access mechanism for applications from independent suppliers to access CIM data in combination with IEC 61970-402: Common Services. An application is expected to use the Generic Data Access (GDA) service as part of an initialisation process or an occasional information synchronization step. GDA is generic in that it can be used by an application to access any CIM data. GDA is also generic in that it also provides a back end storage mechanism independent query capability that can be used to facilitate the creation of CIM data warehouses.

This specification provides a simple, concise service that meets the functionality requirements of current and future applications while:

- avoiding unnecessary complexity;
- not requiring any specific database technology for implementation.

This service is designed to support interaction where the application or system requesting information is developed, supplied, maintained, or operated by a separate agency from the application supplying the data. Furthermore, the update portion of this service assumes that it is undesirable for one system to directly write into another[1)]. To support these objectives, the GDA capabilities are divided into three categories:

a) read access;

b) update access;

c) change notification events.

It should be noted that the update portion of this service does not support unconditional access to critical real-time data. Rather, the update portion allows a requesting application to ask for data to be changed in a service provider, but the service provider is under no obligation to carry out that change at any particular time. Furthermore, a positive response from the update service does not indicate that the update has occurred, but only that the service provider has successfully received the request and that the request is syntactically and semantically correct.

GDA could be classified as an Enterprise Information Integration (EII) technology adapter specialized to the power industry via the assumed use of the CIM. There are a number of EII products currently available on the market, but there is no accepted cross-platform standard for writing connectors for these products and overall these products do not take full advantage of a common semantic model such as the CIM. In recommending GDA, WG 13 is recommending a standard EII connector model in the form of a simpler, less expensive, and more specialized interface.

1) For more information on how 61970 excludes direct control of one application by another, see IEC 61970-402 Annex C: The IEC 61970 services and mapping IEC 61968 verbs.

Though the target of this IEC standard includes the utility control center technical domain, generic data access encompasses a general set of concepts that can be applied to many types of systems. Examples of these systems include:

- Energy and distribution management systems
- Work and asset management systems
- Geographic information systems
- Outage management systems
- Other types of technically oriented operational business systems.

In recognition that the integration between applications in two or more of these systems is often necessary, the intent of this specification is to address general GDA requirements to the extent that they are common to different types of systems while effectively addressing utility operation application specific needs.

2 Normative references

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

IEC 61970-1, *Energy management system application program interface (EMS-API) – Part 1: Guidelines and general requirements*

IEC 61970-2, *Energy management system application program interface (EMS-API) – Part 2: Glossary*

IEC 61970-401, *Energy management system application program interface (EMS-API) – Part 401: Component interface specification (CIS) framework*

IEC 61970-402, *Energy management system application program interface (EMS-API) – Part 402: Component interface specification (CIS) – Common services*

OMG, Utility Management System Data Access Facility, document formal/2002-11-08

3 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 61970-2 apply.

NOTE Refer to International Electrotechnical Vocabulary, IEC 60050, for general glossary definitions.

4 Background

This part of IEC 61970 specifies Component Interface Specifications (CIS) for Energy Management Systems Application Program Interfaces (EMS-API). It specifies the interfaces that a component (or application) shall implement to be able to exchange information with other components (or applications) and/or to access publicly available data in a standard way (see IEC 61970-1 for an overview of these standards). The goal of the creation of this document is to improve the interoperability of utility operational applications and systems. This specification provides a mechanism for applications from independent suppliers to access 61970 Common Information Model (CIM) data using a common service for the purpose of supplementary processing, storage, or display.

In IEC 61970-401, the CIS Framework provides an overview of the CIS documents in the IEC 61970-4xx series. It explains the separation of these specifications into two major groups. One group of standards, IEC 61970-402 to IEC 61970-449[2)], defines a set of generic application independent services that a component shall use for exchanging information with another component or for accessing public data. Added "However, as the generic interfaces do not specify what specific data is exchanged, interoperability between products using IEC 61970 402-449 is not guaranteed." The other group, IEC 61970-450 to IEC 61970-499, defines the information content conveyed using the generic services that a particular component or system exchanges with other components. While IEC 61970-402 to IEC 61970-449 specify application category independent message exchange mechanisms, IEC 61970-450 to IEC 61970-499 specify application category dependent CIM derived message contents.

As explained in IEC 61970-401, a major aspect of the IEC 61970-4xx series is that they take maximum advantage of existing industry standards. Of particular importance are standards developed by the OPC (originally OLE for Process Control) and OMG (Object Management Group). However, these standards are missing a certain functionality considered important for the environment in which the 61970 standards will be applied.

IEC 61970-402 provides the base functionality considered necessary and common that is provided by neither the normative standards incorporated by reference nor the new APIs specified in the IEC 61970-403 to IEC 61970-449 generic interface standards. An application is expected to use the Common Services in conjunction with the generic interfaces. These generic application category independent interfaces include this document as well as:

- IEC 61970-404: High Speed Data Access (HSDA)
- IEC 61970-405: Generic Eventing and Subscription (GES)
- IEC 61970-407: Time Series Data Access (TSDA)

IEC 61970-403 Generic Data Access (GDA) defines services that are needed to access public entity objects for the power system domain that are defined in the IEC 61970-3xx series: Common Information Model (CIM). GDA permits a client to access data maintained by another component (either an application or database) or system without any knowledge of the logical schema used for internal storage of the data. Knowledge of the existence of the common model is sufficient.

5 GDA read access

5.1 General

This specification describes two forms for GDA read access. The first, originally standardized within the OMG as the Utility Management System (UMS) Data Access Facility (DAF), provides clients a basic ability to query for instance data and metadata. The second extends UMS DAF to provide clients a more advanced ability whereby clients can perform more advanced query filtering and joining.

5.2 Read access requirements

GDA read access requirements address the problem of obtaining data from an operational data store on a read-only basis. This includes information describing a real or simulated state of the system together with the system's model data. GDA read access should be sufficient for integrating many applications and systems in a near-real-time or non-real-time mode.

---

2) At this time, only parts 402 to 408 exist. Additional generic services beyond are not yet under consideration.

Target applications expected to use this interface are listed in the draft IEC 61970-401 CIS Framework document. The requirements for the Read Access interface are:

a) Interfaces shall define a standard way to access data appropriate to utility operational systems in general.

b) Interfaces shall support navigation and access to instance data and metadata within a single CIM context from:

- third party near-real-time applications developed independently of the GDA read access provider, including analysis and decision support applications;
- foreign systems that require input from the GDA read access provider on a near-real-time or non-real-time basis, such as other control systems, customer management systems, trading systems and asset management systems.

c) The facility shall provide access to data organized in a complex schema (in particular the CIM) which contains:

- multiple classes of data. Models may contain between one hundred and one thousand classes;
- class attributes belonging to a variety of fundamental types including boolean, integer, floating point, string, and time;
- reference relationships, which may be single or many-valued;
- inheritance relationships.

d) Interfaces shall provide an efficient means to obtain and join large groups of related data, such as selected attribute values for multiple instances. A filtering capability shall be provided to limit the amount of data returned when querying large systems.

e) Interfaces shall not *require* the use of query. It must be possible to implement the facility for a wide range of extent systems, and most of these do not support a standard query language for their real-time data. That is, a GDA provider does not necessarily need to provide a means to query schema to offer a meaningful implementation. For example, applications that are aware of the CIM data model and terminology a priori and use it directly in the API to make direct requests for data do not need a schema query capability.

f) Interfaces shall support a form of read transaction. When a read transaction is used to bracket a series of data access operations, the data obtained are self-consistent and in some sense defined by the underlying system.

g) Interfaces shall take the form of a mapping that can accommodate future change in the CIM. The mapping shall define how classes, attributes and associations in the CIM correspond to constructs or usage patterns in the proposed interfaces.

h) Interfaces shall provide a means to query and join normalized and de-normalized CIM data. That is, it shall be possible to retrieve CIM data where the constructs or usage patterns consist of de-normalized views of CIM data.

i) Interfaces shall define GDA in a manner that supports programming language independence.

j) The GDA shall not include any services that do not provide for implementation independence.

k) It must be possible to access CIM data via the standard mapping in the same way irrespective of the underlying CIM implementation.

l) The interface should be simple and easy to implement to enable standardization and encourage implementation by suppliers and integrators.

5.3 GDA resource query module

5.3.1 General

Structures and interface semantics relevant to basic read access requirements have been substantially covered by the scope of the Object Management Group (OMG) Utility Management System (UMS) Data Access Facility (DAF) standard, to which the read access portion of this standard refers. Specifically, this document normatively includes by reference the UML contained in the UMS DAF resource query interface.

5.3.2 Resource query module description

The DAF resource query module provides UML as shown below. See Figure 1:

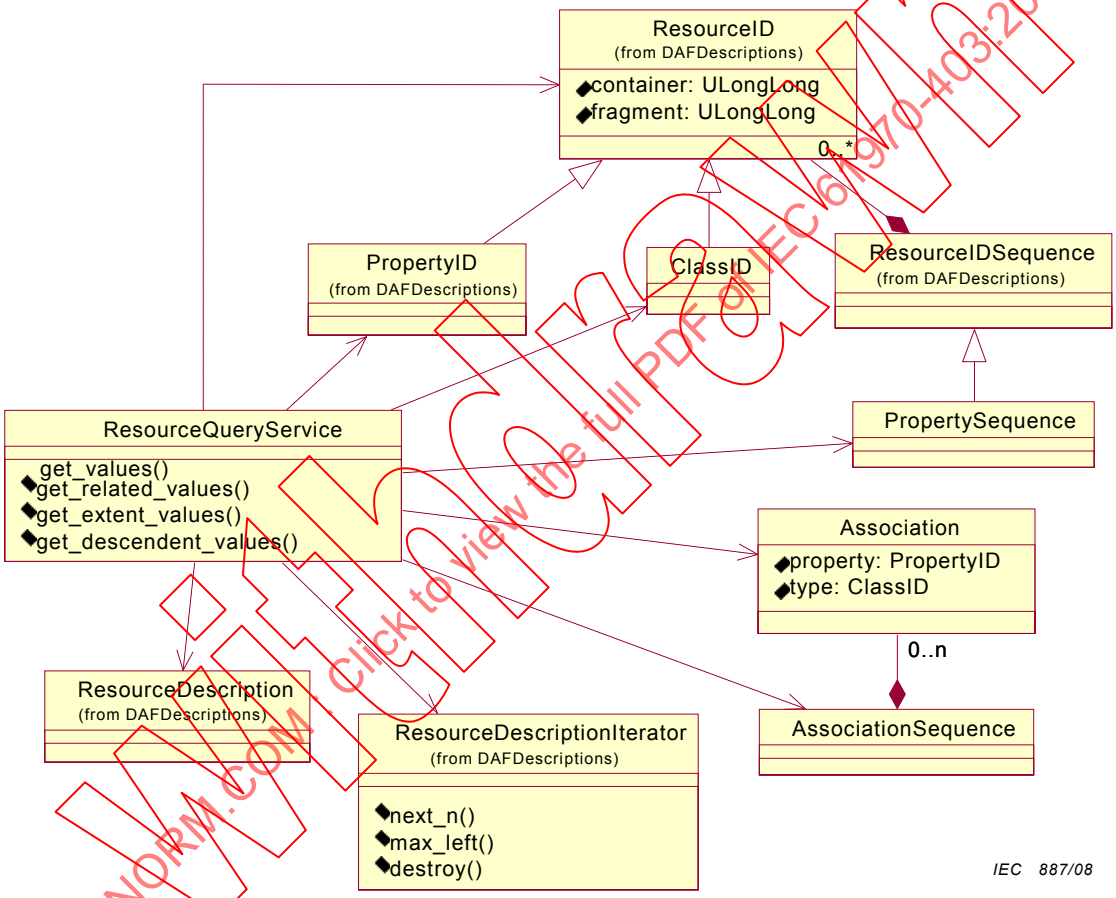


Figure 1 – DAF resource query

The GDA resource query module employs a set of classes for relating and identifying classes, class properties and instances defined in the DAF resource identifiers and resource description modules. For more information about these modules, see the DAF specification and IEC 61970-402: Common services.

5.3.3 GDA resource query service

5.3.3.1 General

Resource descriptions are obtained from operations on the resource query service. The interface provides a family of three base operations and one advanced operation intended to be easy to use. The base operations are: get_values(), get_extent_values(), and get_related_values(). A fourth operation, get_descendent_values(), is a generalization of the other three and is capable of greater optimization. See Table 1.

Table 1 – DAF resource query operations

Operation	Example signature	Throws
get_values	ResourceDescription get_values (ResourceID resource, PropertySequence properties)	UnknownResource, QueryError
get_extent_values	ResourceDescription get_extent_values (PropertySequence properties, ClassID class_id)	UnknownResource, QueryError
get_related_values	ResourceDescription get_related_values (PropertySequence properties, Association association, ResourceID source)	UnknownResource, QueryError
get_descendent_values	ResourceDescription get_descendent_values (PropertySequence properties, AssociationSequence path, ResourceIDSequence sources, AssociationSequence tail)	UnknownResource, QueryError

Each operation on this interface performs a single query. Each resource description returned by a query contains values for a subset of the properties requested. The property values appear in the same order as the properties that were passed to the query, although some may be omitted. A property value is omitted when it is not available from the data provider for the particular resource, or when the property identifier is unrecognised. This behaviour makes it possible to federate multiple query services where each answers part of the query. On the other hand, if the property is recognized but the data provider detects that it is not a member of the resource's class, the QueryError exception is raised. Similarly, QueryError is raised if the data provider determines that a property is many-valued (a resource description cannot represent multiple values for a property).

From a client's perspective, there is always exactly one resource query service in a given context. (Annex A: Use Of GDA Proxies describes how multiple data providers are handled.) A context is a set of applications running to meet a specific business need. For example, contexts might include test, long term planning, or simulation. Context does not appear as a parameter in any GDA service. It is assumed that there can be a separate provider for any one service and context.

5.3.3.2 get_values()

This query requests a resource description for a single resource given by its resource identifier. If the resource identifier is unknown to the data provider, the UnknownResource exception is raised. For more information about these modules, see the DAF specification.

5.3.3.3 get_extent_values()

This query requests a description for each resource of a given class, that is, for each member of the class extent set. The class is given by its ClassID, which is a resource identifier.

- If the resource identifier is unknown to the data provider, the UnknownResource exception is raised.
- If it is recognized but does not represent a class, the QueryError exception is raised.

For more information about these modules, see the DAF Specification.

5.3.3.4 get_related_values()

This query requests a description for each resource associated with a given source resource. The source is specified by a ResourceID, and the association by an association structure (defined below). The data provider evaluates the association for the source resource, which yields zero or more result resources. For each result resource, the data provider evaluates the given properties and generates a resource description, which is returned through the iterator.

- If the source resource identifier is unknown to the data provider, the UnknownResource exception is raised.
- If the data provider does not recognize the property specified in the association, the UnknownAssociation exception is raised.

Since data providers sometimes have only partial information, it is possible that the association is recognized but its value is not available for the given source resource. In that case, the UnknownAssociation exception is also raised. This distinguishes the case where no information is available from the case where the association value is empty. If the data provider detects an error in the association or determines that it is not type compatible with the source resource (as defined below), the QueryError exception is raised. For more information about these modules, see the DAF Specification.

5.3.3.5 get_descendent_values()

This query is a generalization of the foregoing queries and is designed for clients that form queries in a generic manner. It also provides the greatest opportunity for optimisation on the part of the data provider. For more information about these modules, see the DAF Specification.

5.4 GDA filtered query module

5.4.1 General

The GDA filtered query module enhances access to metadata and instance data maintained in a GDA read access server. Specifically, GDA filtered query extends the GDA resource query interface by adding the ability to specify property values that are used to qualify a resource query. In structured query language (SQL) terms, the addition of filters to GDA read access is equivalent to adding a “where” clause. GDA filters allow clients to more precisely define what information they are interested in receiving. For methods in the GDA’s resource query service interface, the GDA’s filtered resource query service interface adds a filter based on a property’s value.

5.4.2 Filtered query module description

5.4.2.1 General

Figure 2 below illustrates the GDA filtered queries model.

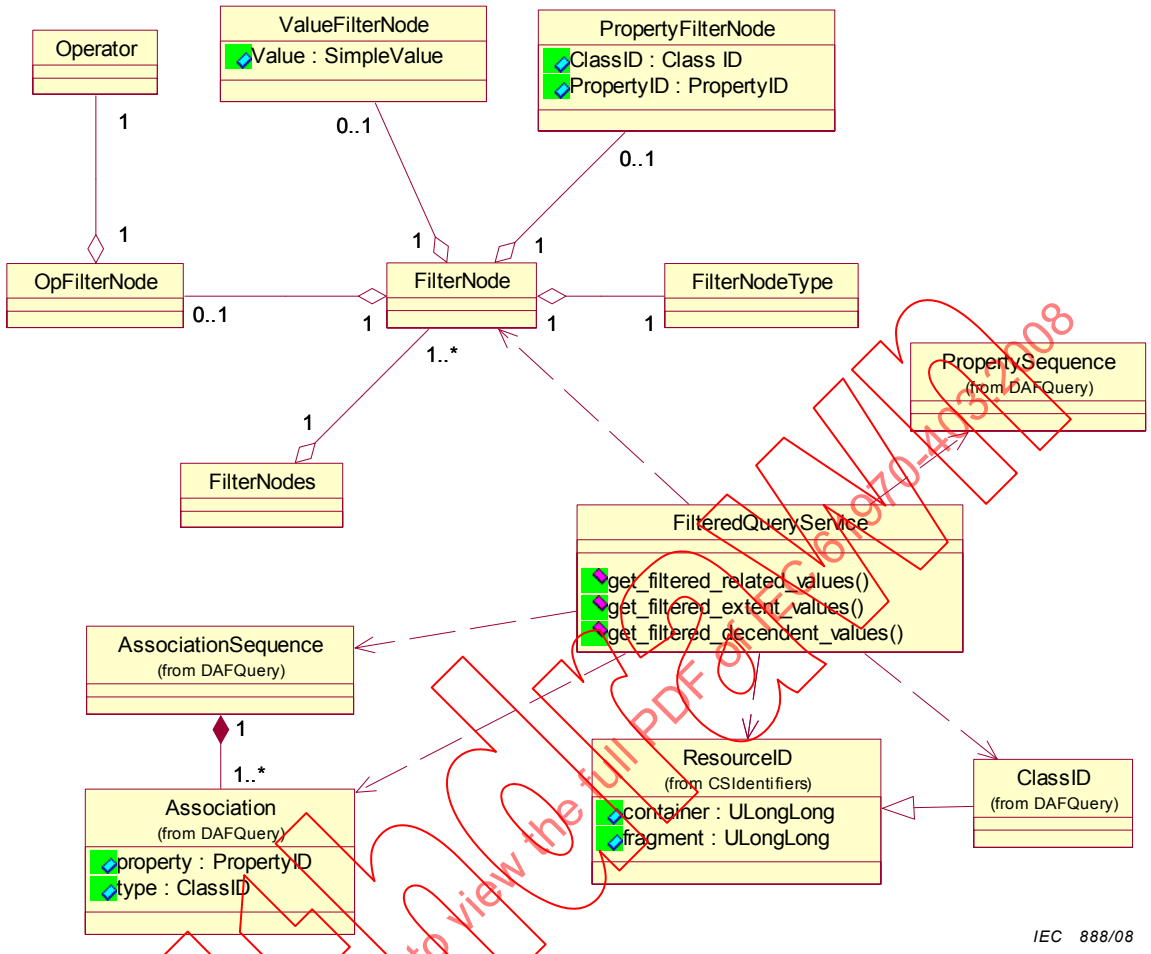


Figure 2 – GDA filtered query

GDA filtering is accomplished by logically AND'ing or OR'ing one or more property value comparisons. Property value comparisons are only made on the returned sequence of resource descriptions of a query. That is, the values of properties returned by a query are used to filter the returned set of resource descriptions. A property value comparison is made by passing in a property ID and a constant value. This constant value is compared against the returned value for the selected property ID. The diagram below (see Figure 3) illustrates how a filter tree can be constructed when using the Filtered Query Module:

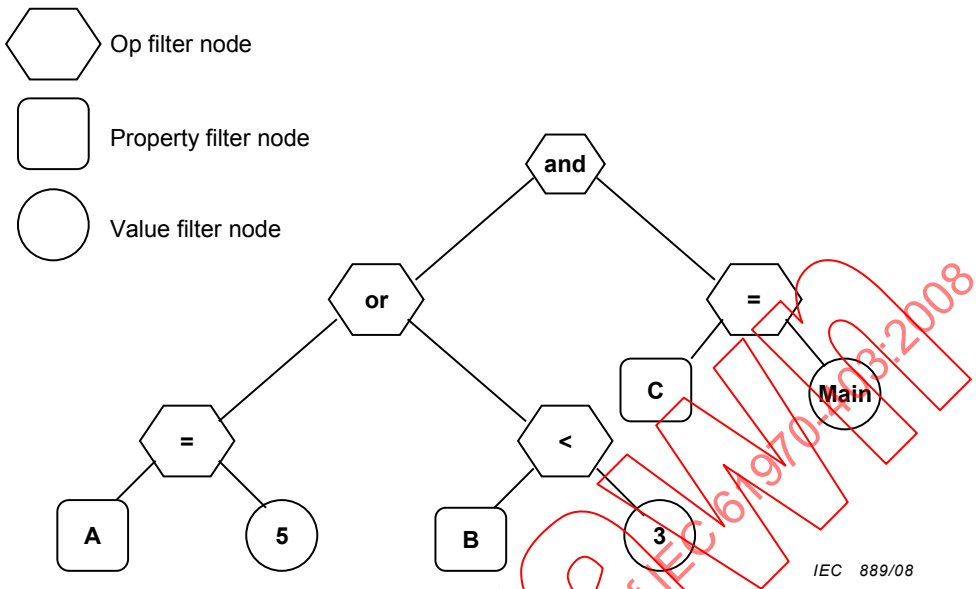


Figure 3 – Example of a complete query filter parse tree

5.4.2.2 FilterNode

A filter is constructed from a triple of filter nodes. A filter node is a generic atomic element that can be used to hold an Op filter node, Property filter node, or a Value filter node. Filter node triples are joined together to create a complete query filter consisting of a sequence of AND'ed or OR'ed filters each of which consists of a triple of filter nodes. Table 2 describes GDA filtered query elements.

Table 2 – GDA filtered query elements

Element name	Description
FilterNodeType	Indicates whether the FilterNode is an OPFilterNode, a ProperyFilterNode, or a ValueFilterNode
OpFilterNode	An optional element containing an OpFilterNode
PropertyFilterNode	An optional element containing a PropertyFilterNode
ValueFilterNode	An optional element containing a ValueFilterNode

5.4.2.3 FilterNodeType

FilterNodeType indicates whether the FilterNode is an OPFilterNode, a ProperyFilterNode, or a ValueFilterNode. Table 3 describes GDA filtered query node types.

Table 3 – GDA filtered query node types

Value	Description
NODE_EMPTY	Used to indicate that the Filter Node is empty
NODE_OP	Used to indicate that the Filter Node is an OPFilterNode
NODE_PROPERTY	Used to indicate that the Filter Node is a PropertyFilterNode
NODE_VALUE	Used to indicate that the Filter Node is a ValueFilterNode

5.4.2.4 FilterNodes

A sequence of FilterNodes is called FilterNodes.

5.4.2.5 Operator

An Operator is a numeric code used to indicate how a property value defined in a property filter node is compared to a constant value contained in a value filter node as well as to indicate how filter node triples are AND'ed or OR'ed together. Table 4 describes GDA filtered query operation types.

Table 4 – GDA filtered query operation types

Value	Description
OP_AND	Used to logically AND filter node triples together
OP_OR	Used to logically OR filter node triples together
OP_EQ	If the property is equal to the value then the property filter triple evaluates to true.
OP_LE	If the property is less than or equals the value then the property filter triple evaluates to true.
OP_GE	If the property is greater than or equals the value then the property filter triple evaluates to true.
OP_LT	If the property is less than the value then the property filter triple evaluates to true.
OP_GT	If the property is greater than the value then the property filter triple evaluates to true.
OP_NE	If the property is not equal to the value then the property filter triple evaluates to true.
OP_LIKE	If the property is a string and the string contains the value then the property filter triple evaluates to true
OP_NOT	If the property is logically not equal to the value then the property filter triple evaluates to true.
OP_ISNULL	If the property is null then the property filter triple evaluates to true.

5.4.2.6 OpFilterNode

An OpFilterNode is used to compare a property filter node with a value filter node as well as to AND or OR filter node triples together. Table 5 describes GDA filtered query OPFilterNode.

Table 5 – GDA filtered query OpFilterNode

Element name	Description
Operator	An Operator as defined in the section 5.4.2.5
FilterNodes	A sequence of Filter Nodes

5.4.2.7 PropertyFilterNode

A PropertyFilterNode is used to specify what property is being compared. Table 6 describes GDA Filtered Query PropertyFilterNode.

Table 6 – GDA filtered query PropertyFilterNode

Element name	Description
ClassID	The Class ID associated with the property being compared
PropertyID	The ID for the property being compared

5.4.2.8 ValueFilterNode

A ValueFilterNode is used to contain the value against which the property is being compared. Table 7 describes GDA filtered query ValueFilterNode.

Table 7 – GDA filtered query ValueFilterNode

Element name	Description
SimpleValue	A discriminated union used to hold the value the property is being compared with

5.4.3 Filtered query service

5.4.3.1 General

Each operation on this interface performs a single query. Each resource description returned by a query contains values for a subset of the properties requested. The property values appear in the same order as the properties that were passed to the query, although some may be omitted. A property value is omitted when it is not available from the data provider for the particular resource, or when the property identifier is unrecognised. This behaviour makes it possible to federate multiple query services where each answers part of the query.

Resource descriptions are filtered using the propertyFilters. That is, a given resource description will not be returned if it contains requested properties whose values do not correspond to the filter property and value. Properties in the filters are not limited to the subset of those in the properties argument of a query.

On the other hand, if the property is recognized but the data provider detects that it is not a member of the resource's class, the QueryError exception is raised. Similarly, QueryError is raised if the data provider determines that a property is many-valued (a resource description cannot represent multiple values for a property). Table 8 describes the GDA filtered query operations.

Table 8 – GDA filtered query operations

Operation	Example signature	Throws
get_filtered_extent_ values	ResourceDescription get_filtered_extent_values (PropertySequence properties, ClassID class_id, FilterNode filter_node)	UnknownResource, QueryError
get_filtered_related_ values	ResourceDescription get_filtered_related_values (PropertySequence properties, Association association, ResourceID source, FilterNode filter_node)	UnknownResource, QueryError
get_filtered_descendent _values	ResourceDescription get_filtered_descendent_values (PropertySequence properties, AssociationSequence path, ResourceIDSequence sources, FilterNode filter_node, AssociationSequence tail)	UnknownResource, QueryError

5.4.3.2 get_filtered_extent_values()

This query requests a description for each resource of a given class, that is, for each member of the class extent set. The class is given by its ClassID, which is a resource identifier. The resource descriptions returned are filtered by propertyFilters. If the resource identifier is unknown to the data provider, the UnknownResource exception is raised. If it is recognized, but does not represent a class, the QueryError exception is raised.

5.4.3.3 get_filtered_related_values()

This query requests a description for each resource associated with a given, parent resource. The parent is specified by a ResourceID, and the association by an association structure. In effect, the data provider evaluates the association property for the parent resource, which yields zero or more child resources. For each child resource, the data provider evaluates the given properties and generates a resource description, which is returned through the iterator. The resource descriptions returned are filtered by propertyFilters.

If the parent resource identifier is unknown to the data provider, the UnknownResource exception is raised. If the data provider does not recognize the association or the value of the association for the parent resource is not available, the UnknownAssociation exception is raised. This distinguishes the case where no information is available from the case where the association value is empty. If the data provider detects an error in the association or determines that its property is not a member of the parent resource, the QueryError exception is raised.

5.4.3.4 get_filtered_descendent_values()

This relatively complex query is designed to support a common use case and allows it to be optimised by the data provider. It is equivalent to repeated application of the get_related_values() query. After each step, the resource descriptions applied to the next step are filtered by propertyFilters. The query requests a description for each resource associated with a given, parent resource via a chain of associations. The parent is specified by a ResourceID, and the chain of associations by an AssociationSequence.

In effect, the data provider evaluates the first association for the parent resource, which yields zero or more child resources subject to the filter provided. For each child resource, the data provider evaluates the second association to yield grandchildren subject to the filter provided. This process continues until the end of the association sequence is reached or an association is encountered which is not recognized or for which no value is available. In this way, the query traverses a tree of resources subject to the filter provided.

If all of the associations are recognized and have values available, the data provider returns descriptions for the leaves of the tree of resources. That is, the resources obtained from the last association in the chain form the results of the query. For each of these, the data provider evaluates the given properties and generates a resource description, which is returned through the iterator. The query also returns an empty association sequence through its tail argument.

If an unknown or unavailable association is reached before the end of the chain, the query returns partial results. The results are formed from the last level of the resource tree that was completely traversed. If the n'th association is unrecognised or fails to yield values for a particular resource, the query returns the resources obtained from the n-1'th association. If the first association fails, the query returns the parent resource. A resource description containing no property values is generated for each of these and is returned through the iterator.

An association sequence representing the failed part of the chain is returned through the tail argument. This tail sequence begins with the failed association and continues with the associations that follow it in the chain. The purpose of the tail is to enable clients (especially proxies) to complete the query by decomposing it and retrying the parts, perhaps through other data providers.

Exceptions are raised in limited circumstances. If the parent resource identifier is unknown to the data provider, the UnknownResource exception is raised. If the data provider detects an error in the association sequence or detects that the first association is not a member of the parent resource, the QueryError exception is raised.

5.5 GDA extended query module

5.5.1 General

GDA's ExtendedQueryService allows the user to join data as opposed to just being able to follow a path as provided by GDA resource query module and GDA filtered query module. In this way, for example, a query can return not only data about breakers from a selected substation, but also return data about the substation in the same query.

5.5.2 Extended resource query module description

5.5.2.1 General

The GDA extended resource query module provides UML that shall be used by the GDA extended query interfaces to accomplish query filtering and joining, as shown in Figure 4.

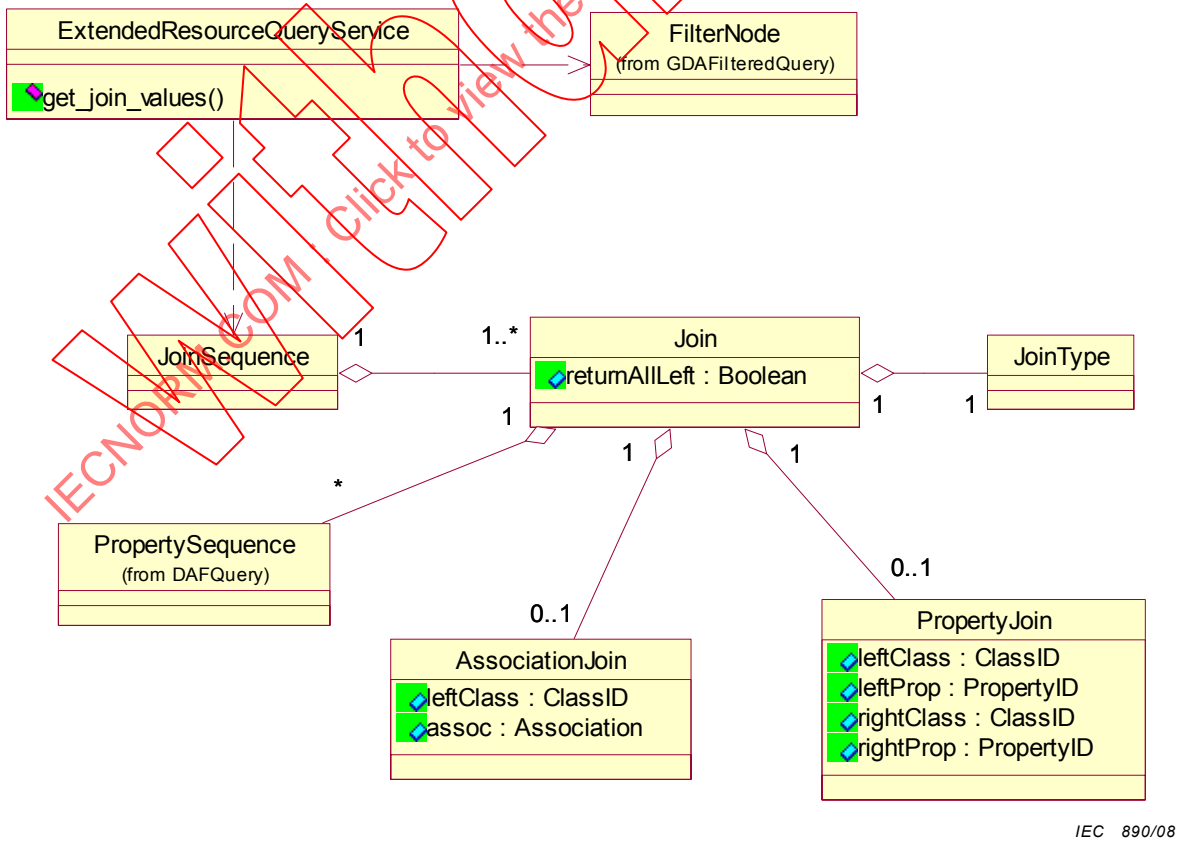


Figure 4 – Extended resource query service UML

As stated in the requirements subclause, GDA needs to be able to retrieve de-normalized views of CIM data. The most frequent application of denormalized views is retrieval from storage of past inter-application messages exchanged via the 61970 Generic Eventing Service (GES). In this case, message payload attributes frequently do not include entire CIM classes and their associations. Instead, the message payload often only includes a de-normalized view of CIM data. After exchange, data related to the de-normalized view is often stored in an application or database. The Extended Resource Query provides the capability to retrieve data from the de-normalized sources as well as the capability to join that data with other normalized CIM data. Consequently, the Extended Resource Query includes the capability to join CIM data on the basis of an association between classes as well as on the basis of shared property value. The diagram below illustrates the use case of joining de-normalized data with normalized data. In this case, the user can employ the Extended Resource Query service to retrieve data joined from the de-normalized message class and the CIM normalized classes using Breaker.Name and Location.Name as the "joining" properties.

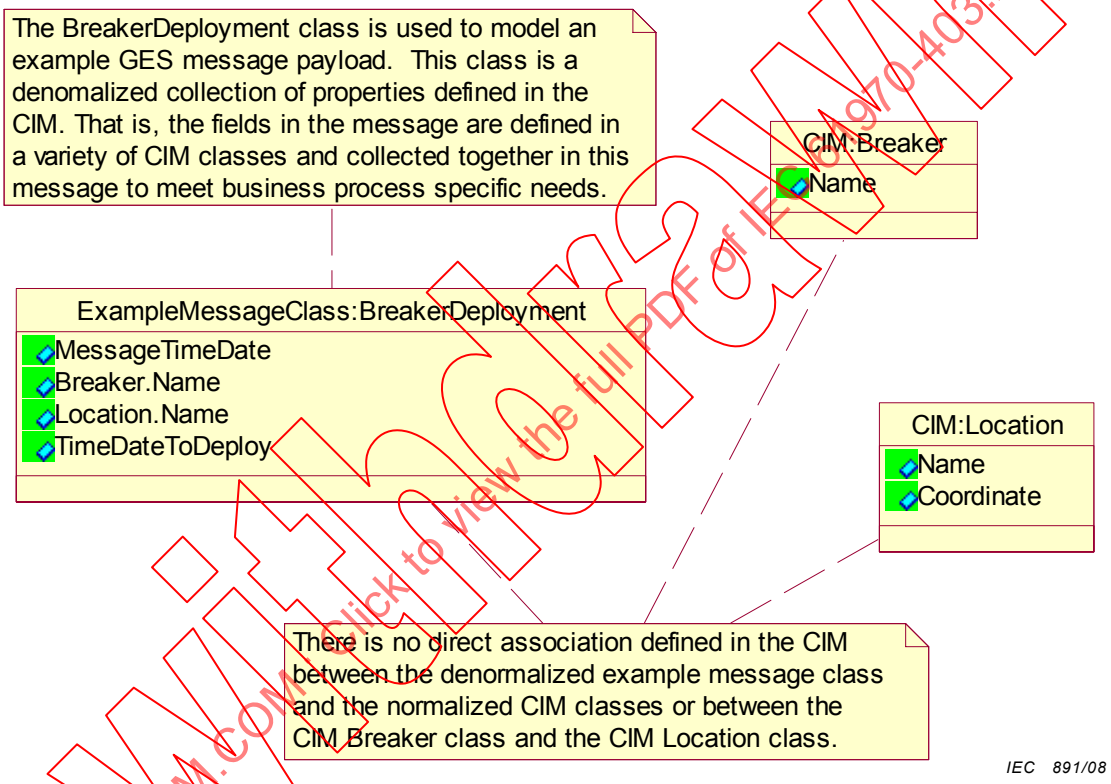


Figure 5 – Example property joining use case

Using extended resource query property joining, it is possible to retrieve data from the three joined classes.

When joining classes together, it is convenient to think of what is being joined as a series of "left" and "right" classes. Using the example illustrated in Figure 5, the first "left/right" pair of classes could be:

- First join:
  - CIM:Breaker is the left class
  - ExampleMessageClass:BreakerDeployment is the right class
- Second join:
  - ExampleMessageClass:BreakerDeployment is the left class
  - CIM:Location is the right class

The diagram below (see Figure 6) illustrates how a parse tree might be created in memory for a query with a property and an association join.

Ex: (ClassA.Property_a = ClassB.Property_b) and ClassB.Assoc_To_ClassC

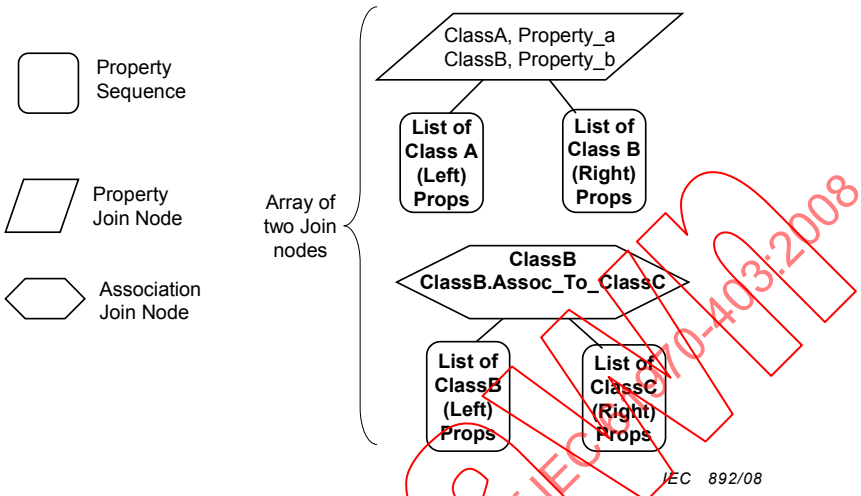


Figure 6 – Join parse tree

Figure 6 depicts two join nodes in a sequence. This sequence is passed in to the get_join_values method along with the array Filter Nodes.

5.5.2.2 Join

A join is constructed from a join node and two property lists. A join node is a generic atomic element that can be used to hold an association join node or a property join node. Join nodes are passed as an array to create a complete query. Table 9 describes GDA filtered query join elements.

Table 9 – GDA filtered query join elements

Element name	Description
joinType	Indicates whether the join is an association join or a property join.
assocJoin	An optional element containing an AssociationJoin
propJoin	An optional element containing a PropertyJoin
leftProperties	A list of properties to return from the “left” class
rightProperties	A list of properties to return from the “right” class
returnAllLeft	If set, an out join is being requested, i.e. all instances of the “left” class are to be returned even if there are no matching instances on the right side of the join.

5.5.2.3 JoinType

JoinType indicates whether the join is an association join or a property join. Table 10 describes GDA filtered query join types.

Table 10 – GDA filtered query join types

Element name	Description
JOIN_ASSOCIATION	Used to indicate that the Join Node is an AssociationJoin Node
JOIN_PROPERTY	Used to indicate that the Join Node is a PropertyJoin Node

5.5.2.4 AssociationJoin node

An AssociationJoin node is used to specify what class and association is being joined on. Table 11 describes GDA filtered query join nodes.

Table 11 – GDA filtered query join nodes

Element name	Description
leftClass	The "left" Class ID for the class being joined
Assoc	The Association ID for the association being joined on

5.5.2.5 PropertyJoin node

A PropertyJoin node is used to specify what classes and properties are being joined. Table 12 describes GDA filtered query property join node.

Table 12 – GDA filtered query property join node

Element name	Description
leftClass	The "left" Class ID for the class being joined
leftProp	The Property ID for the property being joined on
rightClass	The "right" Class ID for the class being joined
rightProp	The Property ID for the property being joined on

5.5.2.6 JoinSequence

A JoinSequence consists of an array of join triples.

5.5.3 Extended resource query service

5.5.3.1 General

Each operation on this interface performs a single query. Each resource description returned by a query contains values for a subset of the properties requested. The property values appear in the same order as the properties that were passed to the query, although some may be omitted. A property value is omitted when it is not available from the data provider for the particular resource, or when the property identifier is unrecognised. This behaviour makes it possible to federate multiple query services where each answers part of the query. Table 13 describes GDA extended resource query operations.

Table 13 – GDA extended resource query operations

Operation	Example signature	Throws
get_join_values	ResourceDescriptionSequenceIterator get_join_values (JoinSequence joins, FilterNode filter_node)	UnknownResource, UnknownAssociation, QueryError

5.5.3.2 get_join_values

This method accepts a sequence of join conditions and returns an iterator that can be used to iterate through the resource description sequences where each resource description sequence represents a single row in the resulting rowset. The caller can also specify a filter to be applied to the resulting rowset. Resource description sequences are obtained via the iterator returned by get_join_values (). Each resource description sequence represents a row in the final rowset.

A resource description sequence is a sequence of resource descriptions. In the context of get_join_values (), resource description sequence has the following semantics:

- The number of ResourceDescriptions in the ResourceDescriptionSequence is equal to or less than the total number of classes being joined.
- Typically, ResourceDescriptions will be returned in the order in which classes are referenced in the JoinSequence, but this is not a requirement.
- For outer joins, if there is not a matching instance, a corresponding ResourceDescription will not be present in the ResourceDescriptionSequence.
- On the client side, typically, ResourceDescriptions will need to be matched back to the original class IDs. To be able to accomplish this, the client should always include “rdf:type” in both *leftProperties* and *rightProperties*. The returned class ID will be the class ID of either the original class or a subclass of the original class.

6 GDA update

6.1 General

The GDA update module extends GDA read access by providing limited write access to metadata and instance data.

6.2 GDA update requirements

The GDA update module is required to provide limited write access to metadata and instance data maintained in a GDA server.

6.3 GDA update module

6.3.1 General

The GDA update interface provides the following functionality.

- Update of objects: In common services terms, this is the ability to update the property values of extant resources.
- Update of multiple objects: Provide the ability to update a collection of objects in one operation.
- Control of object lifecycles: Provide the ability to create and destroy objects accessible through GDA. In Common services terms, this is the ability to create and delete resources.

This also implies associating and disassociating uniform resource identifiers (URI's) with resources[3].

- Update of multiple objects: Provide the ability to update a collection of objects in one operation.
- Update of metadata: Create, delete and update data definitions. In common services terms, this means creating, deleting and updating resources of type rdf:Class and rdf:Property.
- Wrap extant systems: It must be possible to implement the interfaces for a variety of systems including proprietary databases and existing EMS systems.
- Multiple clients, providers and concurrency: Provision must be made for multiple clients and multiple data providers. Where updates are concerned, this requirement is interpreted to imply a form of concurrency control.

6.3.2 Solution approach

6.3.2.1 General

The GDA write access module is shaped by the forgoing requirements and in particular the update concurrency and update constraint issues that arise.

6.3.2.2 Concurrency control solution

The persistent data stores underlying many operational systems in service today do not support transactions. Interface designs that rely on "ACID"[4] transactions cannot be implemented correctly over these systems. Furthermore, many of these systems provide only coarse-grained locking facilities which work at the whole database, table, or physical partition level. Implementations that hold locks of this type on behalf of clients incur severe performance penalties. Deadlocks are likely in an environment with more than a few clients. The interface design should enable simple implementations that are not prone to the foregoing problems.

This specification relies on a system of preconditions to check concurrency conflicts. This solution, similar to some optimistic locking strategies, is scaleable and is simple to implement. Each update operation is accompanied by precondition information that the server uses to detect conflicting concurrent updates. The precondition information represents the client's assumptions about the data, at the time the update was formulated. If these assumptions hold true later, at the time the server applies the update, then no conflicting updates have intervened. Conversely, if the assumptions no longer hold, the updates must be rejected. This is equivalent to a lock conflict. The client controls the scope of the preconditions accompanying an update. This is equivalent to the scope of the read-lock in a locking scheme.

The granularity of the precondition data is at the property level. This is equivalent to the lock granularity. The data provider is required to lock out other sources of updates in the short period while an update operation is executed. However, this period is far shorter than the time taken by the client to formulate the update and does not include any network latency. The data provider is not required to maintain lock state information on behalf of the clients between update operations. There are no abandoned locks to be timed out, nor deadlocks to be detected and cleared.

---

[3] Internet Engineering Task Force: RFC2396, Uniform Resource Identifiers (URI): Generic Syntax; Berners-Lee, Fielding, Masinter, Internet Draft Standard August 1998

[4] ACID is an acronym that stands for "Atomicity, Consistency, Isolation, Durability". These four qualities are the hallmark of transaction models typical supported by a Relational Database Management System such as Oracle or SQL Server.

6.3.2.3 Update constraints solution

The standard database operations are insert, update and delete. Effective application of these operations to a data repository requires that the client be aware of update constraints beyond those expressed in the schema. This is in contrast to read access where the schema is sufficient. Update constraints arise from many implementation-dependent causes. The following examples apply mainly to enterprise-scale relational database systems (even more constraints may be imposed by the proprietary database system used in a control data provider).

a) Primary key, foreign key, index and other constraint definitions

b) Stored procedures that control updates and require certain arguments

c) The effect of trigger definitions

d) The use of separate staging tables for input so that the main tables are not directly updateable

e) The use of views that cannot be directly updated

Implementation choices such as a), b) and c) above tend to impose constraints on the ordering of operations. The exact nature of the constraints depends on the details of the implementation. It is common for certain updates to depend on insertions that in turn depend on still other updates. There is no universal rule for ordering operations (such as all inserts then updates then deletes) that works for all implementations.

Implementation choices such as d) and e) prevent the direct application of any updates. In order for a change to be effected in a visible resource, some sequence of operations must be applied to staging tables or tables hidden behind views.

This specification removes these implementation dependencies by removing the requirement for a client to specify a specific sequence of insertions, updates and deletions. Instead, a transaction is encoded as a model difference that lists the information to be added (forward differences) and removed (reverse differences), but not the operations required nor their order. An implementation is responsible for scheduling a sequence of operations to affect the model difference.

6.3.3 Resource update service module description

6.3.3.1 General

The update functionality is provided by a new service, ResourceUpdateService. This is defined as an interface within the GDAUpdate module. The following class diagram in Figure 7 shows the service and its associated structures.

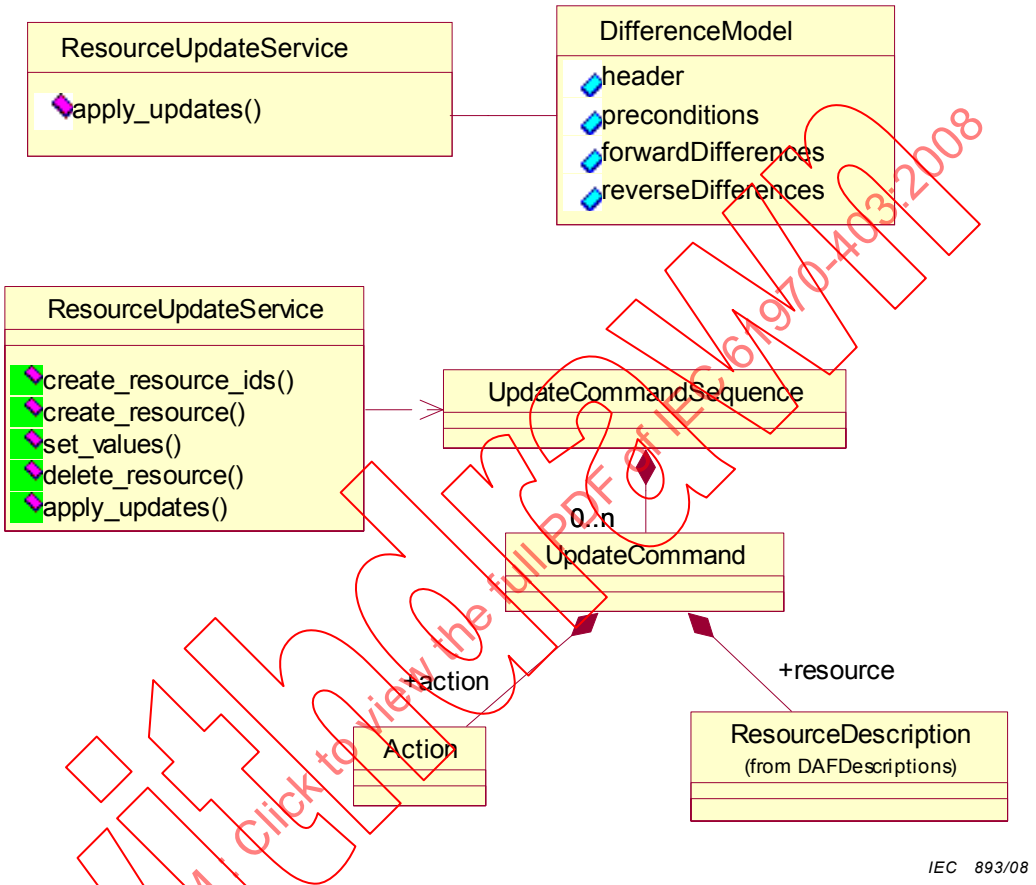


Figure 7 – GDA update module

6.3.3.2 DifferenceModel

A DifferenceModel describes the differences between the model before apply_updates() is invoked and the model after apply_updates() successfully completes. A Difference Model is made up of four groups of statements, each encoded as a sequence of ResourceDescription structures. Any or all of these sets of statements may be empty.

6.3.3.3 headers

This member carries header statements, consisting of statements about the difference model itself. These may indicate authorship, date and purpose. These properties consist of a subset of the CIM document class properties. This member is typically employed when system state is maintained within a document managed by the GDA server (for more information on the use of documents within a GDA server, see Annex C of 61970-402: The IEC 61970 services and mapping IEC 61968 verbs).

Header statements are optional and are provided to assist in tracking and auditing of model updates. If present, they must not affect the model itself.

6.3.3.4 forwardDifferences

This member carries forward difference statements. These statements are found in the updated model, and not found in the old model.

6.3.3.5 reverseDifferences

This member carries reverse difference statements. These statements are found in the old model, and not found in the updated model.

6.3.3.6 preconditions

This member carries precondition statements. These are a subset of the statements found in both the old model and the updated model considered to be dependencies of the difference model in some application defined sense.

Preconditions are the basis of concurrency control when more than one client is performing updates. Each precondition is an assumption on the part of the client about the state of resources prior to update. A precondition failure indicates that another client has completed a conflicting update.

6.3.4 Resource update service

6.3.4.1 General

The ResourceUpdateService interface provides operations to update data that is accessible via GDA Read Access. Table 14 describes GDA resource update service operations.

Table 14 – GDA resource update service operations

Operation	Example signature	Throws
apply_updates	ResourceIDSequence apply_updates (DifferenceModel differences)	PreconditionFailed UpdateError

6.3.4.2 apply_updates()

This operation requests that a series of updates be performed to effect the given model difference as a single operation. The differences argument is a DifferenceModel structure, which is defined in the next section. The operation causes the preconditions to be tested and the model to be updated to reflect the forward and reverse differences.

- The operation will raise a PreconditionFailed exception if any statement in the preconditions or reverse differences of the DifferenceModel is false (not present in the model) prior to updating.

6.3.5 Adding and removing resources

For the purpose of the GDA, adding a resource means adding it to one of the classes in the model. That is done in two steps:

- A ResourceID is obtained for the new resource. This may be obtained from the create_resource_ids() operation or by constructing a new URI Reference and translating it to a ResourceID using the ResourceIDService. Note that this step does not create anything in the model, it merely allocates an identifier.
- The rdf:type property of the new resource is set to the appropriate class. The ResourceID for the rdf:type property and the desired class are both obtained from the ResourceID service. The property is set by adding this information to the forwardDifferences member of a DifferenceModel and passing it to apply_updates().

Conversely, removing a resource means dissociating it from any class in the model. To remove a resource, the statement that would create it is placed in the reverseDifferences member of a DifferenceModel that is then passed to apply_updates().

7 GDA events

7.1 General

GDA eventing augments DAF eventing with a more powerful mechanism. The GDA events package provides the means to notify clients of specific data changes and to ensure consistent data access.

7.2 GDA events module

7.2.1 General

The GDA events module allows the resource ID's for object instances to be embedded in an event as well as a hint to clients as to the nature of the change.

7.2.2 Events module description

7.2.2.1 General

The events module allows a client to receive call back events from a server. Figure 8 shows the UML for the GDA event model.

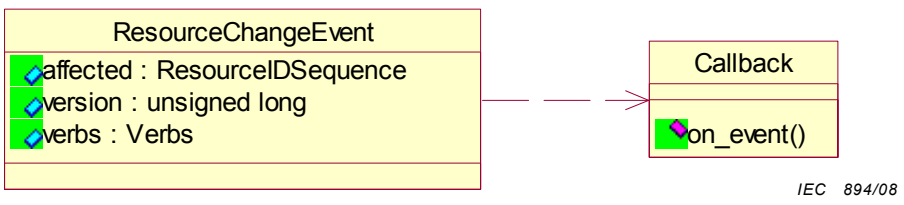


Figure 8 – GDA event model

7.2.2.2 affected

The affected member is a sequence of resource identifiers that indicates what data has changed. If affected is empty, then the client must assume that any or all of data supplied by the data provider may have changed. Otherwise, affected contains one or more class or object identifiers. The client should assume that instances of each class or object have been created or destroyed, or their property values have changed, etc., as described by the verb.

7.2.2.3 verbs

The verbs member is a sequence of numbers that indicates how data has changed. For every member of the affected sequence, there must be a member of the verbs sequence. If affected is empty, then the client must assume that any or all of data supplied by the data provider may have changed. Otherwise, affected contains one or more class or object identifiers. For example, the verbs member could indicate if instances of each class or object have been created or deleted, or their property values have changed.

7.2.2.4 version

The version member is a unique integer assigned to each data change, which may be used to correlate events with changes detected through the DAF current_version() operation. The version number in an event is equal to the value returned by DAF current_version() at the time the event was generated.

7.2.3 Events service

7.2.3.1 General

This interface allows a client to receive updates. Table 15 describes GDA Resource Event Service Operations.

Table 15 – GDA resource event service operations

Operation	Example signature	Throws
on_event	on_event (ResourceChangeEvent event)	

7.2.3.2 ResourceChangeEvent

This structure is passed as an event to indicate a data change. Events of this type are delivered after a data change and indicate that the client may begin reading or re-reading data.

7.2.3.3 Event compression

An implementation is not required to issue an event for every update as it occurs. A data provider may be capable of grouping a series of transactions or simple updates into a larger unit. This series may constitute a logical grouping or a temporal grouping of changes. A single resource change event may be issued after the last change of the series, to cover all of the changes. This is referred to as event compression.

Notwithstanding the use of event compression, the DAF current_version() method must track all of the visible changes as they occur.

7.2.3.4 Event handling guidelines

This specification describes an eventing mechanism that can be seen as a compatible extension of DAF Eventing. When used in combination with DAF Eventing, a client may receive one of three types of events. The most basic type is called a general update event and contains

empty affected and verb sequences. The second most basic event type is called a specific update event and only identifies the changed classes via data and non-empty affected members. The most powerful event type is called an extended update event and identifies the changed classes or objects via data and non-empty affected and verb members. The precision used depends on both the capability of the data provider and the nature of the update. A given data provider may use any of these event types depending on circumstances.

To ensure interoperability, the following guidelines for events should be observed.

- If affected or verb are not empty, then they should identify every class of data changed since the preceding event.
- The data provider should emit exactly one event for an update or series of updates. It should not issue multiple types of update events for the same update.
- Any client that responds to events should respond to general, specific and extended update event types.
- If a client is not capable of interpreting the affected or verb information, it should treat specific update events the same way as general update events.

8 GDA server status and capabilities

8.1 General

The GDA server module extends GDA Read/Write Access by providing information about a GDA Server's status and capabilities.

8.2 GDA server module

8.2.1 General

This GDA server module provides UML that shall be used by the GDA server interfaces to allow a client to determine the status of a server, as shown in Figure 9.

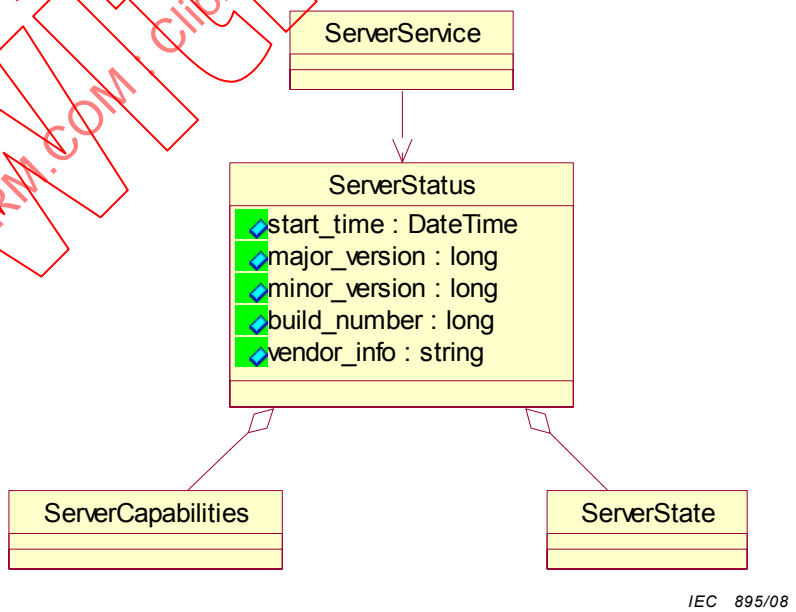


Figure 9 – GDA server model

8.2.2 ServerStatus

This class is returned by the server as a result of invoking the status method. See Table 16.

Table 16 – GDA server status

Element	Description
start_time	Contains the date/time when the server started
Server_state	Contains the current state of the server per Section 7.1.1.2
Major_version	Contains the major version of the server
Minor_version	Contains the minor version of the server
Build_number	Contains the build number of the server
Vendor_info	Contains vender specific information
Server_capabilities	Describes the capabilities of the servers per Section 7.1.1.1

8.2.3 ServerState

An instance of the class ServerState describes the state of a GDA server. This class has a single attribute called ServerState. The value of ServerState shall conform to Table 17:

Table 17 – GDA server state

Value	Description
SERVER_STATE_ RUNNING	Indicates the server is running
SERVER_STATE_ FAILED	Indicates the server has failed
SERVER_STATE_ INDETERMINATE	Indicates the server is in an indeterminate state

8.2.4 ServerCapabilities

An instance of the class ServerCapabilities describes the capabilities of a GDA Server. This class has a single attribute called ServerCapabilities. The value of ServerCapabilities shall conform to Table 18:

Table 18 – GDA server capabilities

Value	Description
SERVER_CAP_QUERY	The server supports the GDA Resource Query Interface.
SERVER_CAP_FILTERED_QUERY	The server supports the GDA Filtered Query Interface.
SERVER_CAP_EXTENDED_QUERY	The server supports the GDA Extended Query Interface.
SERVER_CAP_UPDATE	The server supports the GDA Update Interface.
SERVER_CAP_EVENTS	The server supports the GDA Eventing Interface.
SERVER_CAP_EXTENDED_ID	The server supports the Extended Resource ID Interface.

8.2.5 GDA server module description

This module is used to obtain the status of a server.

8.2.6 Status

This method returns ServerStatus to the caller as described in Table 19.

Table 19 – GDA server status operations

Operation	Example signature	Throws
status	status ()	