



# IST-2000-30082 IDA-STEP

## Integrating Distributed Applications on the Basis of STEP Data Models

Document link to project: Deliverable D08, Part 2

Document title: **SDAI XML query**

Originator: Vaidas Nargėlas, vaidas.nargelas@lksoft.lt

UAB LKSoft Baltic;
Studentų g.65; Kaunas LT-3031
Phone: +370 37 300825; Fax: +370 37 451599

Document history

| *Version* | *Date* | *Status* | *Evolutions* | *Responsible person/organization* |
|---|---|---|---|---|
| 1 | 2003-01-31 | Draft | Created | Co-authors, LKSoft |
| 2 | 2003-07-04 | Draft | Revised for spec. V1.1 | Co-authors, LKSoft |

# Table of Contents

# Introduction

Queries in SDAI were originally defined in *ISO 10303-22::10.4.14 SDAI query* [1]. However requirements of the other deliverables of the project led to new XML based SDAI query specification.

The goal of this work was to identify the ways to overcome limitations of SDAI query [1], make a new extended specification, and create the reference implementation.

# SDAI query requirements

The following list of requirements is based on requirements of IDA-STEP project workpackages 5, 6, and 7 [2]:

1. To support execution both on locally loaded models and database located models
2. To be able to handle big amount of data on the database (beyond the size of available RAM)
3. To be represented in a way convenient for transmission from various kinds of clients to the server
4. To handle querying both AIM and ARM data directly
5. To be suited for querying arbitrary STEP schema without prior knowledge in the query engine
6. To be relatively easy comprehended by human despite the complexity of the queries
7. To have extensible nature for future additions

Simple analysis of SDAI query described in [1] revealed that this kind of query can handle only relatively simple queries. Requirement to handle ARM data in queries led to thinking that queries should support constructs used in mapping reference path syntax [3]. It was obvious that SDAI query can not handle complexity of mapping reference path.

Several alternatives were considered:

1. Using mapping reference path syntax. Advantage of this approach is that mapping reference paths have rich list of querying constructs. However reference path syntax is ill defined and often has ambiguous meaning.
2. Extending SDAI query syntax. The syntax described in *ISO 10303-22::10.4.14* [1] covers only small portion of requirements. It would have to be extended substantially and the original idea of SDAI queries was lost especially because the syntax is not designed for extensibility.
3. Using XPATH based syntax. XPath [4] is used as a query language for XML and possible could be used for querying STEP data. However XPath is well suited for XML tree structures and gets not very useful for non tree structures. Representing arbitrary STEP schema as a tree automatically is not possible therefore XPath looses its advantages and becomes too limited. Upcoming version 2.0 of XPath [5] was not considered because of a lack of well working implementations.
4. Using XML based syntax. A new XML based specification could be developed trying to overcome issues with other approaches. New specification can be specifically designed for querying arbitrary STEP data for using both AIM and ARM. Compared to mapping reference path it would try to escape ambiguities as much as possible. Compared to SDAI query it would not suffer from too short query construct list. Compared to XPath it would not be suited for querying only XML tree oriented data but would be more verbose and possibly less readable.

As a result of considering above mentioned alternatives the last mentioned approach was chosen. Use of the rest of the syntaxes can be achieved by adding intermediate translation level which translates query in the specific syntax to XML based query as a common way to represent a query.

# SDAI XML query specification

The following chapter describes SDAI XML query specification version 1.1. It outdates query specification version 1 described in [6].

## Query namespace

SDAI XML query namespace has the URI `http://www.lksoft.com/SDAI/Query/V1.1`. SDAI XML query engine must use this XML namespace for `query`/`query-lib` top element or their `query-element-prefixes` attribute. Other query elments including `query`/`query-lib` can be in SDAI XML namespace or any of the namespaces listed in `query-element-prefixes`.

## Query element namespaces

SDAI XML query utilizes a set of custom XML namespaces which define semantics of constraints elements. In particularly the namespace defines semantics of `type`, `fwd`, `inv`, and `val` constraints. For the rest of SDAI XML query elements any listed namespace (including SDAI XML query namespace) can be used.

Query top element attribute query-element-prefixes enumerates custom XML namespace prefixes used by the query. The namespace URI defines the semantics of constraint elements in this namespace. URI used by JSDAI have the following format:

```
jsdai:[ schema:SCHEMA_NAME
      | mapping:MAPPING_NAME
      | query-lib: [ QUERY_LIB_ID | #local_query_lib_id ]
      ]
```

The URI in the form `jsdai:schema:` refers to STEP schema to be accessed by a query.

The URI in the form `jsdai:mapping:` refers to ARM mapping to be accessed by a query. Mapping can be used only for local queries.

The URI in the form `jsdai:query-lib:` refers to query library that is available to query engine to be accessed by a query.

## Query element

Query is described by `query` top level element. Inside a query `domain` and `result` elements can be specified. The element may have the attribute `context` which has value `local` or `remote`. If `context` is `local` then query is performed on a locally loaded STEP data and if `context` is `remote` then query is performed on remote database.

## Query-lib element

Query library is described by `query-lib` element. It is used as top level element for standalone queries and as a child of `query` element for local queries. Standalone query library has id specified by mandatory `id` attribute and may have scope specified by scope attribute. The query library can have `transaction`, `session` or `global` scope. The default is `transaction` scope. The previously created query library can be removed by using `idref` attribute to specify id of the query library and remove operation by specifying remove=”yes” attribute.

Query libraries can be used as the way to have macro query definitions for reoccurring portions of constraints. The primary application of query libraries is performing ARM based queries on a STEP database.

The element `query-ent` is the only valid child element of a `query-lib`.

## Domain element

Domain element can be used to specify query domain when the domain is specifically known before the query but is identified by some search criteria. This is especially targeted towards STEP databases with huge amounts of data.

### *Result element*

Result specifies one or more named result sets that can be retrieved when the query is completed. Result can have `name` attribute. Name of a result can be used retrieving resulting data. The result element may contain constraint elements defined in the next section.

### *Query-ent element*

Element `query-ent` defines entity querying semantics for query libraries. It is used used as a child of `query-lib` element only. It has an attribute `name` which defines the name of a query entity.

### *Query-type element*

Element `query-type` defines constraints that are applied when the `type` constraint on the containing query entity is performed.

### *Query-fwd element*

Element `query-fwd` defines constraints that are applied when the `fwd` or `inv` constraints on the containing query entity is performed. Contained constrants are applied in straightforward direction for `fwd` constraint and in inverse direction for `inv` constraint. This element has to have the attribute `attr` and may have the attribute `target`. These attributes define the attribute to apply the `query-fwd` on and have the similar meaning as `fwd` constraint attributes.

### *Query-val element*

Element `query-val` defines constraints that are applied when the `val` constrain on the containing query entity is performed. This element has to have the attribute `attr` and may have the the attribute `select`. These attributes define the attribute to apply the `query-val` on and have the similar meaning as `val` constraint attributes.

### *Constraint elements*

The list of constraint elements follows:

- `type`
- `fwd`
- `inv`
- `val`
- `intersect`
- `union`
- `and`
- `or`
- `not`
- `items`

### *Execution model and constraint elements*

Query execution consists of the following steps:

- Results contained in a query are processed
- For each result the list of constraint elements is traversed.

- Each constraint element accepts input instance set and emits the output instance set. Inside of a query output set becomes the input set of the next constraint. Starting input instance set is defined by query engine at the moment `execute` method is invoked. Output set of `val` constraint can be the set of any simple type values depending on the context.
- The result of an individual `result` is the output set of the last constraint element.
- Each constraint element may contain child constraints. This is to further narrow the parent constraint element. Instance is included in the parent constraint output instance set only if using it as input set of the first child constraint for execution of child constraint list leads to non empty child output instance set.

### *Type constraint element*

The output instance set of the `type` constraint is the subset of the input set containing only instances that of specified type. The entity type is identified by `ent` attribute. If `exact` attribute is `yes` then instances of subtype of specified type don't match the constraint otherwise both instances of the type and its subtypes match the constraint.

### *Fwd constraint element*

The output instance set of the `fwd` constraint is a set of instances that are attribute values of the input instance set. The express attribute containing values is specified by `attr` attribute. The `attr` may be accompanied by `ent` attribute to avoid ambiguity. For the case when express attribute is of aggregation type `aggr` and `aggr-size` attributes can further constraint output instance set in a similar way like mapping reference path uses `[i]` notation [3]. Additionally `target` attribute may specify the output instance entity type.

### *Inv constraint element*

The output instance set of the `inv` constraint is a set of instances that reference the input instance set. Input instance set is references through express attribute specified by `attr` and `ent` attributes. For the case when specified attribute is of aggregation type `aggr` and `aggr-size` attributes can further constraint inverse reference.

### *Val constraint element*

This constraint can result in different output sets depending on the context where `val` constraint is used. If `val` is used as a direct last child element of `items` or `query-val` elements then the output set of this constraints is the values of express attribute specified by `attr` attribute. In all other contexts the output instance set of the `val` constraint is a subset of input instance set containing only instances that have value set for the express attribute specified by `attr` attribute. The `attr` may be accompanied by `ent` attribute to avoid ambiguity. Val constraint can not contain child constrains but can contain comparison elements possibly grouped using `and` or `or` elements. Current specification version defines two comparison elements but this can be extended in future versions:

- equality match `eq` element
- non equality match `neq` element.

When val constraint contains comparison elements output set is constrained to only values/instances which have attribute values that match specified comparison elements.

### *Group element*

Group element can be used as a child of `intersect`, `union`, `and`, or `or` constraints. These elements contain the set of child constraint lists. Each list is contained by `group` element. However if the list consists of only one constraint this constraint can be direct child of parent `intersect`, `union`, `and` or `or` constraint.

### *Intersect constraint element*

The output instance set of `intersect` constraint is intersection of output sets of each child constraint list that are emitted by the last constraint in the list.

### *Union constraint element*

The output instance set of `union` constraint is union of output sets of each child constraint list that are emitted by the last constraint in the list.

### *And constraint element*

The output instance set of `and` constraint is subset of input set containing only instances for which all child constraint lists emit non empty set by the last constraint in the list. There can be cases when `intersect` and `and` constraints have identical semantics.

### *Or constraint element*

The output instance set of `or` constraint is subset of input set containing only instances for which at least one child constraint lists emit non empty set by the last constraint in the list. There can be cases when `union` and `or` constraints have identical semantics.

### *Not constraint element*

The output instance set of `not` constraint is subset of input set containing only instances for which child constraint lists emits empty set by the last constraint in the list.

### *Items constraint element*

Constraint `items` is used only as direct last child element of `result`. It collects the output results of child constraints and supplies them for query result set. This constraint may have the attribute `instances` with value `include` or `exclude`. If `instances` is `include` then the input instances are treated as first item of query result set. If `instances` is `exclude` then the input instances are not inclued in query result set. The default is to include instances.

### *Further possible extensions to specification*

Some possible extensions to specification were considered:

1. Parameter support. Adding parameters would let to execute queries based on previous queries results. Parameters could be used in constraint matching as instance sets or as values.
2. Applying other query results as part of constraints being executes in current query.
3. Extended domain matching. This version has only limited matching possibilities. (They are not described in details in this document.) One possible way would be to describe domain using express and then use above constraints for matching.
4. Add more comparison operators to `val` constraint including regular expressions.

## SDAI XML query engine API

SDAI XML query API includes extensions in `SdaiSession` class and new interfaces `QuerySource`, `SdaiQuery`, and `QueryResultSet`.

### *SdaiSession class extension*

```
package jsdai.lang;
public final class SdaiSession
    extends SdaiCommon implements QuerySource {

    ...

    public SdaiQuery newQuery(org.w3c.dom.Document querySpec)
        throws SdaiException {
        ...
    }

    public SdaiQuery newQuery(org.w3c.dom.Element el)
        throws SdaiException {
        ...
    }
} // SdaiSession
```

Two new overloaded methods `newQuery` were added to `SdaiSession` class in `jsdai.lang` package. For XML queries `SdaiSession` acts as a factory and these methods serve as entry points to creation of queries.

### *QuerySource interface*

```
package jsdai.lang;
public interface QuerySource {

    /**
     * Returns domain in which to perform a search
     */
    ASdaiModel getQuerySourceDomain()
        throws SdaiException;

    /**
     * Returns instances with which to perform a search
     */
    AEntity getQuerySourceInstances()
        throws SdaiException;

    /**
     * SDAI query according to part 22.
     */
    int query(String where, EEntity entity, AEntity result)
        throws SdaiException;

} // QuerySource
```

`QuerySource` interface is part of `jsdai.lang` package. It defines the domain over which an SDAI query operation is executed as defined in *ISO 10303-22::9.3.12 query_source* [1]. Therefore `SdaiModel`, `SdaiRepository`, and `SchemaInstance` classes implement `QuerySource` interface and `Aggregate` interface extends `QuerySource`. `QuerySource` defines `getQuerySourceDomain` and `getQuerySourceInstances` methods which provide domain and initial input instance set for a query. QuerySource also defines `query` method compatible with *ISO 10303-27* [7]. `QuerySource` can be used as input parameter to query execution.

### *SdaiQuery interface*

```
package jsdai.lang;
public interface SdaiQuery {

    public void setQuerySource(QuerySource qs)
        throws SdaiException;

    public void setDomain(ASdaiModel domain)
        throws SdaiException;

    public void execute()
        throws SdaiException;

    public void execute(QuerySource qs)
        throws SdaiException;

    public void execute(QuerySource qs, ASdaiModel domain)
        throws SdaiException;

    public AEntity getResult(String name)
        throws SdaiException;

    public AEntity getResult(int index)
        throws SdaiException;

    public QueryResultSet getResultSet(String name)
        throws SdaiException;

    public QueryResultSet getResultSet(int index)
        throws SdaiException;

    public String[] getResultNames()
        throws SdaiException;

    public void setParameter(String name, Object value)
        throws SdaiException;

    //public void close()
    //    throws SdaiException;

} // SdaiQuery
```

`SdaiQuery` class is part of `jsdai.lang` package. This class represents SDAI XML query. It contains several overloaded `execute` methods. Query can be executed over query source and/or domain passed as parameters or can be executed over the default domain. The default query source and domain can be set with `setQuerySource` and `setDomain`. Method `setParameter` is for future use when parameter support is added to SDAI XML query specification. There are two overloaded `getResult` methods that return result entities as `AEntity` aggregate identified either by name `String` or by 0 based index. However the latter two methods are considered deprecated since support for specification version 1.1. Two new overload methods `getResultSet` were added. There is the version which accepts the `String` and another version which accepts 0 based index. Both methods return `QueryResultSet` which is the way to get query result items. Method `getResultNames` returns result names contained in this query.

### *QueryResultSet interface*

```
package jsdai.lang;
public interface QueryResultSet {

    public boolean next()
        throws SdaiException;

    public Object getItem(int itemPos)
        throws SdaiException;

} // QueryResultSet
```

`QueryResultSet` interface is a mean to get items of a query result. It has method `next` to access the next result. This method also returns `false` in the case results are exhausted. The method `getItem` returns the item identified by 1 based index and listed in `items` constraint in the query definition.

## Anticipated future changes

This API version is considered to be almost stable. However some extension should be added to ensure optimal execution of remote database queries. Below are the classes that will be added as part of query implementation:

- `jsdai.query.EEntityRef`
- `jsdai.query.SchemaInstanceRef`
- `jsdai.query.SdaiModelRef`
- `jsdai.query.SdaiRepositoryRef`

These classes were added as extension to JSDAI implementation to provide the way to transparently refer to Eentity, SchemaInstance, SdaiModel and SdaiRepository object between JSDAI client and SQL Bridge. The remote queries will return the above objects as items in query result.

## Query engine prototype implementation

Query engine prototype implementation was created for use in realization of IDA-STEP project deliverables. The core part of the query engine is located in package `jsdai.query`. The engine has the extensible design which allows both local queries and remote queries to share the same core.

Prototype implementation covers the following:

- Full parsing and interpretation of SDAI XML queries according to specification version 1[6] and 1.1.
- Complete constraint element type support except aggregate handling (`attr` and `attr-size` attributes are ignored).
- Complete querying in locally loaded SdaiModels and partial querying in remoted STEP database.
- Query library support for local and remote queries. It is a shared implementation.

## Query application

SDAI XML queries were extensively used in deliverable D08-1-v1 [8]. Queries will play important role in merging process [9] when updated query engine version is available. Also queries are going to become integral part of STEP-Book [10] as a way to specify STEP data requests by user.

# References


[1] ISO 10303-22: Implementation methods: Standard data access interface specification
[2] IST-2000-30082: Integrating Distributed Applications on the Basis of STEP Data Models. Annex 1 - "Description of Work"
[3] ISO TC 184/SC4 N1190:2001(E): Guidelines for the development of mapping specifications, 2nd edition
[4] XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath
[5] XML Path Language (XPath) 2.0 W3C Working Draft 15 November 2002, http://www.w3.org/TR/2002/WD-xpath20-20021115/
[6] IDA-STEP deliverable D08-2-v1: SDAI XML Query (Version 1)
[7] ISO/TS 10303-27: Product data representation and exchange: Implementation methods: Java™ programming language binding to the standard data access interface with Internet/Intranet extensions
[8] IDA-STEP deliverable D08-1-v1: JSDAI based implementation of "PDTnet"
[9] IDA-STEP deliverable D18-2-v1: Status report for configurable tools to merge STEP exchange files with STEP databases
[10] IDA-STEP deliverable D07-1-v1: Generic STEP-Book Architecture


# Appendix

## *SDAI Query DTD*

DTD specification is only a limited specification since DTD does not have enough features to describe SDAI XML query specification.

```
<?xml version="1.0"?>

<!-- JSDAI Query syntax specification Version 1.1                  -->
<!-- ------------------------------------------                   -->
<!-- Copyright (c) 2003 LKSoftWare GmbH                           -->
<!-- $Id: query-v1.1.dtd,v 1.1 2003/07/03 16:52:50 vaidas Exp $ -->

<!-- *************** Query ***************                       -->
<!--                    query                                     -->
<!--                    The top element of a query document       -->
<!ELEMENT  query        (domain | result | query-lib)+              >

<!--          xmlns      Namespace specification
              context    Execution context (local or remote)
              xmlns:pfx  Sample query element namespace
              query-element-prefixes
                         The prefixes that should be treated as part
                         of query. In example DTD this is "pfx".    -->
<!ATTLIST  query
              xmlns      CDATA
                         #FIXED "http://www.lksoft.com/SDAI/Query/V1.1"
              xmlns:pfx  CDATA                              #IMPLIED
              context    (local|remote)                   "local"
              query-element-prefixes
                         NMTOKENS                          #REQUIRED >

<!-- *************** Query Library ***************               -->
<!--                    query                                     -->
<!--                    The top element of query library or child
                        of query as local query library            -->
<!ELEMENT  query-lib    (query-ent)+                                >
<!--          id        Query library identifier which can be used by
                        another query as part of query-lib URI.
              idref     Query library reference identifier used to
                        reference query defined before when attribute
```

```
                    remove is used.
              scope      The scope of the query library. The default
                         is "transaction". If query-lib is a child of
                         query then scope can not be specified
                         explicitly and "local" scope is assumed.
              remove     The remove operation on a previously defined
                         query library.
              ** The following attributes are only used **
              ** when query-lib is a top element        **
              xmlns      Namespace specification
              context    Execution context (local or remote)
              xmlns:pfx  Sample query element namespace
              query-element-prefixes
                         The prefixes that should be treated as part
                         of query. In example DTD this is "pfx".    -->
<!ATTLIST  query-lib
              id         NMTOKEN                          #IMPLIED
              idref      NMTOKEN                          #IMPLIED
              scope      (transaction|session|global)     #IMPLIED
              remove     (yes)                            #IMPLIED
              xmlns      CDATA
                         #FIXED "http://www.lksoft.com/SDAI/Query/V1.1"
              xmlns:pfx  CDATA                            #IMPLIED
              context    (local|remote)                  "local"
              query-element-prefixes
                         NMTOKENS                         #IMPLIED  >

<!-- *************** Domain ***************                          -->
<!-- Since domain is not supported this part of DTD is outdated    -->
<!--                    domain                                      -->
<!--                    Query domain on which query is
                        performed                                   -->
<!ELEMENT  domain       (repository | model | schema-instance)+      >

<!--                    repository                                  -->
<!--                    Repository where to query                   -->
<!ELEMENT  repository   (name | description | change-date | author |
                         organization | preprocessor-version |
                         originating-system | authorization |
                         default-language | context-identifiers)*    >

<!--                    model                                       -->
<!--                    Model where to query                        -->
<!ELEMENT  model        (name | underlying-schema | change-date |
                         default-language | context-identifiers)*    >

<!--                    schema-instance                             -->
<!--                    Schema instance where to query              -->
<!ELEMENT  schema-instance
                        (name | native-schema | change-date |
                         validation-date | validation-result |
                         validation-level | description | author |
                         organization | preprocessor-version |
                         originating-system | authorization |
                         default-language | context-identifiers)*    >

<!--                    name                                        -->
<!--                    Name of repository, model, or schema
                        instance                                    -->
<!ELEMENT  name         (#PCDATA)                                    >

<!--                    description                                 -->
<!--                    Description of repository or schema
                        instance                                    -->
```

```
<!ELEMENT  description  (#PCDATA)                                    >

<!--                   change-date                                  -->
<!--                   Date of recent modification of
                       repository, model, or schema
                       instance. String in ISO format               -->
<!ELEMENT  change-date  (#PCDATA)                                   >

<!--                   author                                       -->
<!--                   Author of repository or schema instance      -->
<!ELEMENT  author       (#PCDATA)                                   >

<!--                   organization                                 -->
<!--                   Organization of repository or schema
                       instance                                     -->
<!ELEMENT  organization (#PCDATA)                                   >

<!--                   preprocessor-version                         -->
<!--                   Preprocessor version of repository or
                       schema instance                              -->
<!ELEMENT  preprocessor-version
                       (#PCDATA)                                    >

<!--                   originating-system                           -->
<!--                   Originating system of repository or
                       schema instance                              -->
<!ELEMENT  originating-system
                       (#PCDATA)                                    >

<!--                   authorization                                -->
<!--                   Authorization of repository or schema
                       instance                                     -->
<!ELEMENT  authorization
                       (#PCDATA)                                    >


<!--                   default-language                             -->
<!--                   Default language of repository, model, or
                       schema instance                              -->
<!ELEMENT  default-language
                       (#PCDATA)                                    >

<!--                   context-identifiers                          -->
<!--                   Context identifiers of repository, model,
                       or schema instance                           -->
<!ELEMENT  context-identifiers
                       (#PCDATA)                                    >

<!--                   underlying-schema                            -->
<!--                   Underlying schema of the model               -->
<!ELEMENT  underlying-schema
                       (#PCDATA)                                    >

<!--                   native-schema                                -->
<!--                   Native schema of the schema instance         -->
<!ELEMENT  native-schema
                       (#PCDATA)                                    >

<!--                   validation-date                              -->
<!--                   Validation date of the schema
                       instance. String in ISO format               -->
<!ELEMENT  validation-date
                       (#PCDATA)                                    >
```

```
<!--                    validation-result                              -->
<!--                    Validation result of the schema
                        instance. An integer                           -->
<!ELEMENT  validation-result
                        (#PCDATA)                                       >

<!--                    validation-level                               -->
<!--                    Validation level of the schema
                        instance. An integer                           -->
<!ELEMENT  validation-level
                        (#PCDATA)                                       >

<!-- *************** End domain ***************                       -->

<!-- *************** Constraint entities ***************              -->
<!--                    set-operations                                 -->
<!--                    List of all set operations                     -->
<!ENTITY % set-operations
                       "intersect | union | and | or | not"             >

<!--                    comparison-operations                          -->
<!--                    List of all comparison operations              -->
<!ENTITY % comparison-operations
                       "eq"                                             >

<!--                    constraints-list                               -->
<!--                    List of common constraints                     -->
<!ENTITY % constraint-list
       "pfx:type | pfx:fwd | pfx:inv | pfx:val | %set-operations;"      >

<!--                    constraints                                    -->
<!--                    Constraint list                                -->
<!-- Output: input instance subset that match child constraints        -->
<!ENTITY % constraints "(%constraint-list;)*"                           >

<!--                    Constraints or several compound constraint
                        groups                                         -->
<!ENTITY % constraints-or-group
                       "((%constraint-list; |
                        %comparison-operations;)+ | (grp)+)"            >

<!--                    Yes or no answer                               -->
<!--                    Defines values of boolean type attributes      -->
<!ENTITY % yes-no      "(yes | no | true | false)"                      >

<!--                    Entity name                                    -->
<!ENTITY % ent         "ent NMTOKEN"                                    >

<!--                    Attribute name                                 -->
<!ENTITY % attr        "attr NMTOKEN"                                   >

<!--                    Aggregate member(s)                            -->
<!--                    Possible values: * or integer index            -->
<!ENTITY % aggr        "aggr CDATA"                                     >

<!-- *************** End constraint entities ***************          -->

<!--                    Aggregate size                                 -->
<!--                    Value: an integer                              -->
<!ENTITY % aggr-size   "aggr-size CDATA"                                >

<!-- *************** Query library elements ***************           -->
<!--                    query-ent                                      -->
<!--                    Query library entity definition                -->
```

```
<!ELEMENT  query-ent     (query-type, (query-fwd | query-val)* )       >
<!--         name        Name of the entity                            -->
<!ATTLIST  query-ent
             name        NMTOKEN                              #REQUIRED >

<!--                     query-type                                    -->
<!--                     Query library type constraint definition      -->
<!ELEMENT  query-type    %constraints;                                  >

<!--                     query-fwd                                     -->
<!--                     Query library fwd constraint definition       -->
<!ELEMENT  query-fwd     %constraints;                                  >
<!--         %attr;      Attribute name
             target      Type of the target entity instance            -->
<!ATTLIST  query-fwd
            %attr;                                            #REQUIRED
             target      NMTOKEN                              #IMPLIED  >

<!--                     query-val                                     -->
<!--                     Query library val constraint definition       -->
<!ELEMENT  query-val     %constraints;                                  >
<!--         %attr;      Attribute name
             select      Optional select path                          -->
<!ATTLIST  query-val
            %attr;                                            #REQUIRED
             select      NMTOKENS                             #IMPLIED  >

<!-- *************** End query library elements ***************      -->

<!-- *************** Constraints ***************                     -->
<!-- Query result definition                                         -->
<!--                     result                                      -->
<!--                     Query result definition                     -->
<!ELEMENT  result        %constraints;                                  >
<!--         name        Name of the named query result              -->
<!ATTLIST  result
             name        NMTOKEN                              #IMPLIED  >

<!--                     pfx:type                                    -->
<!--                     (Sub)Type of entity constraint              -->
<!-- Output: input instance subset that match the specified type     -->
<!-- Child constraints can further restrict the output set.          -->
<!-- See: ENTITY % constraints                                       -->
<!ELEMENT  pfx:type      %constraints;                                  >
<!--         ent         Entity type
             exact       If this is an exact (sub)type               -->
<!ATTLIST  pfx:type
             %ent;                                            #REQUIRED
             exact       %yes-no;                             #IMPLIED  >

<!--                     pfx:fwd                                     -->
<!--                     Forward reference (attribute value)         -->
<!-- Output: a set of instances that are attribute values of         -->
<!--         the input instance set                                  -->
<!-- Child constraints can further restrict the output set.          -->
<!-- See: ENTITY % constraints                                       -->
<!ELEMENT  pfx:fwd       %constraints;                                  >
<!--         %ent;       Entity to which attribute belongs
             %attr;      Attribute of the reference
             %aggr;      Aggregate memeber(s)
             %aggr-size;Aggregate size(s). Only one of aggr or
                         aggr-size can be specified.
             target      Type of the target entity instance          -->
<!ATTLIST  pfx:fwd
```

```
            %ent;                                           #IMPLIED
            %attr;                                          #REQUIRED
            %aggr;                                          #IMPLIED
            %aggr-size;                                     #IMPLIED
             target     NMTOKEN                             #IMPLIED  >

<!--                    pfx:inv                                       -->
<!--                    Inverse reference                             -->
<!-- Output: a set of instances that reference (are users of)         -->
<!--         the input instance set                                   -->
<!-- Child constraints can further restrict the output set.           -->
<!-- See: ENTITY % constraints                                        -->
<!ELEMENT  pfx:inv      %constraints;                                  >
<!--         %attr;     Attribute pointing to current entities
             %ent;      Entity to which attribute belongs
             %aggr;     Aggregate member(s)
             %aggr-size;Aggregate size(s). Only one of aggr or
                        aggr-size can be specified.                   -->
<!ATTLIST  pfx:inv
            %attr;                                          #REQUIRED
            %ent;                                           #REQUIRED
            %aggr;                                          #IMPLIED
            %aggr-size;                                     #IMPLIED  >

<!--                    pfx:val                                       -->
<!--                    Value of the attribute which is not of
                        instance type. Use fwd for attributes that
                        are entity instances.                         -->
<!-- Output: input instance subset that matches specified value       -->
<!--         constraints or is set if child constraints are           -->
<!--         not specified                                            -->
<!ELEMENT  pfx:val      (%comparison-operations; | and | or)?          >
<!-- Only %comparison-operations; can be used inside and/or but       -->
<!-- DTD is not restrictive enough                                    -->
<!--         %ent;      Entity to which attribute belongs
             %attr;     Attribute of the reference
             select     Select path
             %aggr;     Aggregate member(s)
             %aggr-size;Aggregate size(s). Only one of aggr or
                        aggr-size can be specified.                   -->
<!ATTLIST  pfx:val
            %ent;                                           #IMPLIED
            %attr;                                          #REQUIRED
             select     NMTOKENS                            #IMPLIED
            %aggr;                                          #IMPLIED
            %aggr-size;                                     #IMPLIED  >

<!--                    intersect                                     -->
<!--                    Intersects resulting entity sets              -->
<!-- Output: intersection of output instance sets that are            -->
<!--         received at the end of each child constraint path        -->
<!ELEMENT  intersect    %constraints-or-group;                         >

<!--                    union                                         -->
<!--                    Unites resulting entity sets                  -->
<!-- Output: union of output instance sets that are                   -->
<!--         received at the end of each child constraint path        -->
<!ELEMENT  union        %constraints-or-group;                         >

<!--                    and                                           -->
<!--                    And operation on input instances              -->
<!-- Output: input instance subset that matches all                   -->
<!--         child constraints                                        -->
<!ELEMENT  and          %constraints-or-group;                         >
```

```
<!--                    or                                          -->
<!--                    Or operation on input instances             -->
<!-- Output: input instance subset that matches at least one        -->
<!--         child constraint                                       -->
<!ELEMENT  or           %constraints-or-group;                       >

<!--                    not                                         -->
<!--                    Instances that don't match constrain.       -->
<!-- Output: input instance subset that does not match              -->
<!--         child constraint                                       -->
<!ELEMENT  not          %constraints;                                >
<!-- *************** End constraints ***************               -->

<!-- *************** value comparison ***************              -->
<!-- Comparison operators can be combined using and/or              -->
<!--                    eq                                          -->
<!--                    Equal match.                                -->
<!ELEMENT  eq           (#PCDATA)                                    >

<!--                    neq                                         -->
<!--                    Not equal match.                            -->
<!ELEMENT  neq          (#PCDATA)                                    >
<!-- *************** End value comparison ***************          -->

<!--                    grp                                         -->
<!--                    Group of several constraints as one
                        compound constraint                         -->
<!ELEMENT  grp          %constraints;                                >

<!--                    items                                       -->
<!--                    Defines result set items. Can be used only
                        as direct last element of <result>          -->
<!-- Output: input instances (if instances is "include") and child -->
<!-- constraint instances and/or values                             -->
<!ELEMENT  items        %constraints-or-group;                       >
<!--         instances  Specifies if the input instance set has to be
                        included in the result set.                 -->
<!ATTLIST  items
             instances  (include|exclude)                 "include" >

<!-- *************** End query ***************                       -->
<!-- Change history:
     $Log: query-v1.1.dtd,v $
     Revision 1.1  2003/07/03 16:52:50  vaidas
     Specification upgraded to version 1.1

     Revision 1.4  2003/01/24 10:49:12  vaidas
     Fixed typo in comments.

     Revision 1.3  2002/12/12 16:39:45  vaidas
     First real specification.
     Updates from query implementation prototype.
     Comments describing how constraints work added.

     Revision 1.2  2002/12/12 16:32:12  vaidas
     First incomplete draft dated 2002-11-21
-->
```

## *SDAI XML Query example*

### Excerpt from `GetDocumentsAndNames` example:

```
public static void getDocumentsAndNames(String stepFile)
    throws SdaiException, ParserConfigurationException,
           IOException, SAXException {

    SdaiSession sdaiSession = SdaiSession.openSession();
    SdaiTransaction transaction =
        sdaiSession.startTransactionReadWriteAccess();
    SdaiRepository repository =
        sdaiSession.importClearTextEncoding("", stepFile, null);
    SdaiModel model = repository.findSdaiModel("default");

    DocumentBuilderFactory documentBuilderFactory =
        DocumentBuilderFactory.newInstance();
    documentBuilderFactory.setNamespaceAware(true);
    documentBuilderFactory.setIgnoringComments(true);
    documentBuilderFactory.setIgnoringElementContentWhitespace(true);
    DocumentBuilder documentBuilder =
        documentBuilderFactory.newDocumentBuilder();
    queryDocumentsAndNames(sdaiSession, model, documentBuilder);
}

public static void queryDocumentsAndNames(SdaiSession sdaiSession,
                                          SdaiModel model,
                                          DocumentBuilder documentBuilder)
    throws SdaiException, ParserConfigurationException,
           IOException, SAXException {

    URL queryLibUrl =
        GetDocumentsAndNames.class.getResource("arm214DocumentLib.xml");
    Document queryLibSpec = documentBuilder
        .parse(queryLibUrl.openStream(), queryLibUrl.toString());
    SdaiQuery queryLib = sdaiSession.newQuery(queryLibSpec);

    URL queryUrl =
      GetDocumentsAndNames.class.getResource("queryDocumentsAndNames.xml");
    Document querySpec = documentBuilder
        .parse(queryUrl.openStream(), queryUrl.toString());

    SdaiQuery query = sdaiSession.newQuery(querySpec);
    query.execute(model);
    QueryResultSet rSet = query.getResultSet("documents-and-names");
     while(rSet.next()) {
        EEntity document = (EEntity)rSet.getItem(1);
        String name = (String)rSet.getItem(2);
        System.out.println(document.getPersistentLabel() + "   \t" + name);
    }
}
```

### Query `queryDocumentsAndNames.xml`:

```
<?xml version="1.0" encoding="utf-8"?>

<query xmlns="http://www.lksoft.com/SDAI/Query/V1.1"
       xmlns:arm214="jsdai:query-lib:arm214DocumentLib"
       query-element-prefixes="arm214" context="local">
  <result name="documents-and-names">
    <arm214:type ent="document"/>
    <items>
      <arm214:val ent="document" attr="name"/>
    </items>
  </result>
</query>
```

### Query library `arm214DocumentLib.xml`:

```
<?xml version="1.0" encoding="utf-8"?>

<query-lib id="arm214DocumentLib"
           xmlns="http://www.lksoft.com/SDAI/Query/V1.1"
           xmlns:aim214="jsdai:schema:automotive_design"
           query-element-prefixes="aim214">
  <query-ent name="document">
    <query-type>
      <aim214:type ent="product">
        <aim214:inv ent="product_related_product_category"
                    attr="products" aggr="*"/>
        <aim214:val attr="name">
          <eq>document</eq>
        </aim214:val>
      </aim214:type>
    </query-type>
    <query-val attr="name">
      <aim214:type ent="product"/>
      <aim214:val attr="name"/>
    </query-val>
  </query-ent>
</query-lib>
```

### Output using `d14-1-v1.stp` file:

```
#62     Reifen
#358    Kardanwelle
#111    Felge
#536    Riemenantrieb mit Zusatzaggregaten
#258    Feder Re
#446    Lenktrapez
#202    Rumpfmotor
#580    Achskoerper Hinterachse angetrieben
#402    Achskoerper Vorderachse
#302    Schaltgetriebe
```