International

Virtual

Observatory

Alliance

IVOA SkyNode Interface
Version 1.03

IVOA Working Draft 11 Jul 2006

This version:
1.03 http://www.ivoa.net/Documents/WD/SNI/SkyNodeInterface-20060511.doc[S1]

Latest version:
http://www.ivoa.net/Documents/latest/SkyNodeInterface.html

Previous versions:
none

Working Group:
http://www.ivoa.net/twiki/bin/view/IVOA/IvoaVOQL

Editors:
Maria A. Nieto-Santisteban, William O'Mullane, Masatoshi Ohishi

Authors:
IVOA VOQL Working group

Abstract

This document describes the required interface that SHALL be supported to participate in the IVOA as a SkyNode.

Status of this document

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsolete by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress". [S2]

Acknowledgments

This working draft has been developed based on discussions at various IVOA meetings and continuing emails on the mailing list. The editors express their appreciation for many valuable contributions from the VOQL WG.

Contents

1 Introduction

Astronomical data (e.g., catalogs) may be published as SkyNodes. Data providers publish the SkyNodes by registering the services with the VO registry, so that they become discoverable by client applications (e.g., Open SkyQuery[1] portal). SkyNodes provide the front-end to the actual databases, using the interface described in this document. At a fundamental level, SkyNodes may be called by client programs. At a higher level, portals (e.g., Open SkyQuery and JVO QueryPortal) provide access to SkyNodes, calling the registry to discover SkyNode services and building an execution plan that coordinates query processing [Figure 1].

[1] http://www.openskyquery.net

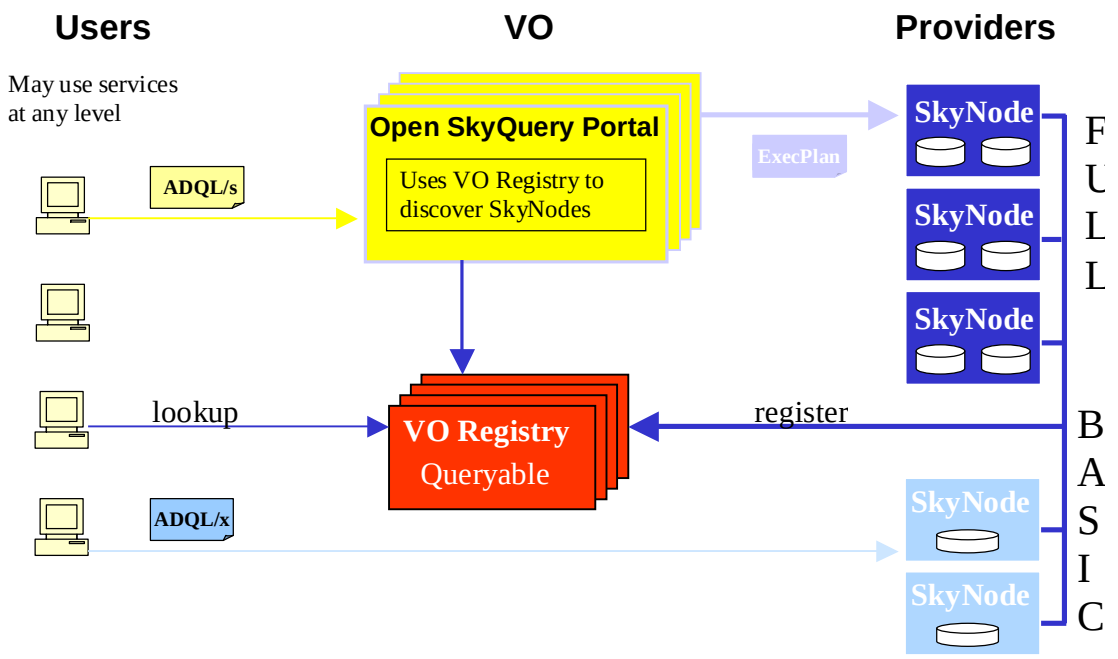

Figure 1. SkyNode query architecture (need update)

The Open SkyQuery Portal would accept a form of string based query, which is in fact ADQL/s (as shown in Figure 1). A parser converts this into ADQL/x.. The portal uses the registry to resolve server names for individual SkyNodes, makes the Execution Plan (see Section 3.2 below), and passes it on to the first node in the plan setting up an execution sequence as in the current SkyQuery portal.

The SkyNode concept originated with the SkyQuery portal (www.skyquery.net), which demonstrated the use of the Microsoft .NET framework and SOAP Web services to access federated astronomical databases. This document redefines SkyNodes to be SOAP Web services that are independent of the underlying framework and compliant with other IVOA standards. SkyNodes are Web services that may be implemented in at least Java/Axis and C#/.NET. From the perspective of the client, the framework used to implement a SkyNode is irrelevant, as demonstrated by the Open SkyQuery portal.

Clients interact with SkyNodes using ADQL (Astronomical Data Query Language), which is specified separately in the ADQL Specification Document [1]. ADQL is an SQL-like language used by the IVOA to represent astronomy queries posted to VO data services (e.g., SkyNodes). ADQL has two forms: ADQL/s (string form) and ADQL/x (XML representation corresponding to ADQL/s). SkyNodes are XML Web services that accept ADQL/x queries, posted by clients or translated from ADQL/s by clients or portals. The ADQL Specification Document distinguishes between the mandatory core of the language and optional extensions that allow higher-level query capabilities.

SkyNode interfaces are defined by WSDL (Web Services Definition Language) documents, which specify the functional interface (methods) provided by SkyNodes. Data providers may chose to implement SkyNodes at various compliance level. Based on the level of compliance to the ADQL syntax and SkyNode interface, a Skynode is classifed into 5 types, that is, ENTRY, XMATCH, AGNET, ADVANCED and FULL.

This document describes definitions of SkyNode interfaces.

2 IVOA Standard Interface

IVOA standard interfaces are defined elsewhere, and they are regarded as upper level interfaces. To be a complience with much wider range of services, they should be considered to be implemented on the skynode.

3 SkyNode Interface

3.1 General

QI-1 SkyNodes SHALL be implemented as web services and their interface is described in a WSDL document. All the standard interfaces described in this document SHALL be defined under a namespace "http://www.ivoa.net/xml/SkyNode/2006". The latest WSDL file is based on ADQL 1.5, and it is located at http://www.ivoa.net/xml/SNI/SkyNode-v1.?.wsdl[S4] (not provided yet).

QI-2 SkyNodes SHALL return an error message as a SOAP exception, when an error is detected on the services. The error message composed of an integer "error code", a predefined "reason" string, and more detailed explanation "detail" supplied by a service implementator.

```
<xs:element name="ErrorResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="errorCode"/>
      <xs:element name="reason"/>
      <xs:element name="detail"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

error code	reason	comment
1	syntax error	An invalid or not supported syntax of ADQL is detected.
2	not supported feature	An unsupported feature, such as function, operation is detected.
3	service busy	Too many queries are being requested and the query is rejected.
4	unauthorized	prohibited to access the resource.

5	timeout	The query takes too long time, and process is killed.
6	I/O error	I/O error related with data storage, data trasfer, etc.
7	internal error	Error related with internal unexpected error.
254	others	

3.2 Query Interface

QI-3 SkyNodes SHALL implement a **DataExist** method, which returns true if there is a record that satisfies the query condition, otherwise false. The result may be used by a search engine to check whether the table contains data of requesed region, and to contruct a coverage map for all the skynodes. The estimate should be insignificant compared to the time to execute a QueryCost described below.

```
boolean DataExist()

<s:element name="DataExist">
  <s:complexType>
    <s:sequence>
      <s:element ref="ADQL" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:element name="DataExistResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="DataExistResult" type="s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>
```

QI-4 SkyNodes SHALL implement a **QueryCost** method, which returns the number records that satisfies the query condition. The results may be used to build efficient execution plans for distributed query. The estimate should be insignificant compared to the time to execute a PerformQuery described below.

```
long QueryCoat(adql)

<s:element name="QueryCost">
  <s:complexType>
    <s:sequence>
      <s:element ref="ADQL" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:element name="QueryCostResponse">
  <s:complexType>
```

```
      <s:sequence>
        <s:element name="QueryCostResult" type="s:integer" />
      </s:sequence>
    </s:complexType>
  </s:element>
```

QI-5 SkyNodes SHALL implement a **PerformQuery** method, which takes an XML document including an ADQL and an optional string parameter called `returnType` that specifies how the result to return. The `returnType` MUST be one of the strings listed from a call to the "returnTypes" interface. The PerformQuery method returns a document including a single appropriate "VOData" type element. "VOData" SHALL be an abstract type with multiple subtypes. SkyNodes SHALL support at least "SOAP" return type. When the `returnType` parameter is omitted, "SOAP" is assumed. "HTTPRef", "FTPRef", "Attachment" and/or "VOSpaceRef" MAY be supported as a returnType. Regardless of any return type specified, the query result MUST be a form of VOTable version 1.1 or higher. When "select into …" statement is used to store the query result on VOSpace, reference to the VOSpace where the result is sotred returns.

```
VOData PerformQuery(adql [, returnType])

<s:element name="PerformQuery">
   <s:complexType>
     <s:sequence>
       <s:element ref="ADQL" />
       <s:element minOccurs="0" name="returnType"
       type="s:string" />
     </s:sequence>
   </s:complexType>
 </s:element>

 <s:element name="PerformQueryResponse">
   <s:complexType>
     <s:sequence>
       <s:element name="PerformQueryResult" type="VOData" />
     </s:sequence>
   </s:complexType>
 </s:element>

 <s:complexType name="VOData" abstract="true"/>

 <s:complexType name="VOTable">
   <s:complexContent mixed="false">
     <s:extension base="VOData"/>
       <s:element ref="VOTABLE"/>
     </s:extension>
   </s:complexContent>
 </s:complexType>

 <s:complexType name="AccessRef">
   <s:complexContent mixed="false">
     <s:extension base="VOData"/>
```

```
        <s:element name=”AccessRef” type=”s:string”/>
      </s:extension>
    </s:complexContent>
  </s:complexType>
```

QI-6 SkyNodes MAY implement a **VOTableJoin** method that takes `ADQL`, `VOData` and an optional `returnType` parameters.

```
VOData VOTableJoin(adql, ArrayOfVOData [, returnType])

<s:element name="VOTableJoin">
  <s:complexType>
    <s:sequence>
      <s:element ref="ADQL" />
      <s:element maxOccures=”unboundd” ref="VOData" />
      <s:element minOccurs="0" name="returnType"
      type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:element name="VOTableJoinResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="VOTableJoinResult" type="VOData" />
    </s:sequence>
  </s:complexType>
</s:element>
```

QI-7 SkyNodes MAY implement an **ExecutePlan** method that takes an `ExecPlan` and passes the relevant part of the plan to the next node. From that node it shall receive a `VOData`. If there are no more nodes in the plan it simply executes the ADQL query and returns the resulting `VOData`. The return type MAY be specified in the plan at each step. The logical node and physical replicas SHALL be sent with the plan. The ExecPlan structure contains a portalURL the plan came from and which is used for logging, the return type of the final result, and an ordered array of PlanElements. A node that recieved an excutePlan excute the first element in the PlanElements and passes the rest of the element to the next node.

```
VOData ExecutePlan(Plan)

<s:element name="ExecutePlan">
  <s:complexType>
    <s:sequence>
      <s:element name="Plan" type="ExecPlan" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:element name="ExecutePlanResponse">
```

```
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" name="ExecutePlanResult"
        type="VOData" />
      </s:sequence>
    </s:complexType>
  </s:element>

  <s:complexType name="ExecPlan">
    <s:sequence>
      <s:element minOccurs="0" name="ReturnType"
      type="s:string" />
      <s:element minOccurs="0" name="PortalURL"
      type="s:string" />
      <s:element minOccurs="0" name="PlanElements"
      type="ArrayOfPlanElement" />
    </s:sequence>
  </s:complexType>

  <s:complexType name="ArrayOfPlanElement">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded"
      name="PlanElement" nillable="true" type="PlanElement" />
    </s:sequence>
  </s:complexType>
```

The PlanElement structure SHALL contain the statement (ADQL document), the Target for this element (a logical node name) and an ordered array of hosts (physical nodes which should behave as the logical name – there may be only one). The order of the hosts SHOULD be in most preferable first, this is of course form the portal perspective – a node MAY decide to re-rank the nodes for itself or simply send a quick query to all mirrors and take the first which responds. The PlanElement SHOULD contain the format of output required from this Node.

```
<s:complexType name="PlanElement">
  <s:sequence>
    <s:element minOccurs="0" name="Statement" type="ADQL" />
    <s:element minOccurs="0" name="returnType" type="s:string"/>
    <s:element minOccurs="0" name="Target" type="s:string" />
    <s:element minOccurs="0" name="Hosts" type="ArrayOfString"/>
  </s:sequence>
</s:complexType>
```

QI-8 SkyNodes MAY implement the footprint service. This would take a region specified in the region XML and return a new region which is the intersection of the survey and the given region. The Region specification has also the ability to deal with Spectral and Temporal footprints, this implies a node should be able to deal with these entities. Currently we know how to deal with regions (just about) but have not really dealt with temporal nor spectral overlaps.

```
RegionType Footprint(region)

<s:element name="Footprint">
  <s:complexType>
    <s:sequence>
      <s:element name="region" type="regionType" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:element name="FootprintResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" name="FootprintResult"
      type="regionType" />
    </s:sequence>
  </s:complexType>
</s:element>
```

3.3 Metadata Access Interface

The SkyNode interface includes methods that return Metadata with information about dataset. The metadata are exposed through ADQL, however it has been considered important to provide methods for querying the metadata using Web services calls.

QI-9 SkyNodes SHALL implemnts a **ADQLSpecs** interface, which returns a list of supported ADQL specifications:

- ADQLSpecList: (ADQLSpec[]) Array of ADQL syntax specification metadata.
    - ADQLVersion: (String) A version of ADQL.
    - ListOfADQLExtensionID: (String[]) Array of Extension IDs.

```
ArrayOfADQLSepc ADQLSpecs()

<s:element name=”ADQLSpecs”>
  <s:complexType/>
</s:element>

<s:element name="ADQLSpecsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" name="ADQLSpecsResult"
       type="ArrayOfMetaADQLSpec" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ArrayOfMetaADQLSpec">
  <s:sequence>
```

```
      <s:element maxOccurs="unbounded" name="MetaADQLSpec"
      type="MetaADQLSpec" />
    </s:sequence>
  </s:complexType>

  <s:complexType name="MetaADQLSpec">
    <s:sequence>
      <s:element minOccurs="0" maxOcrus="unbounded"
      name="ExtensionID" type="s:string" />
    </s:sequence>
    <s:attribute name="version" type="s:string"/>
  </s:complexType>
```

QI-10 SkyNodes SHALL implement a "**ReturnTypes**" interface, which takes no parameters and returns a list of supported return types for Query Results. "SOAP" format MUST be included in the list.

```
ArrayOfString ReturnTypes()

<s:element name="ReturnTypes">
  <s:complexType />
</s:element>

<s:element name="ReturnTypesResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="ReturnTypesResult"
      type="ArrayOfMetaReturnType" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ArrayOfMetaReturnType">
  <s:complexType>
    <s:sequence>
      <s:element maxOccurs="unbounded" name="MetaReturnType"
      type="MetaReturnType" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="MetaReturnType">
  <s:complexType>
    <s:sequence>
      <s:element maxOccurs="unbounded" name="MetaReturnType"
      type="s:string"/>
    </s:sequence>
  </s:complexType>
</s:element>
```

SkyNodes SHALL implement a "**Tables**" interface, which takes an optional "name" and "includeColumns" parameter, and returns an array of table metadata. When "name" parameter is specified, metadata of all the specified tables returns, otherwise all the metadata returns. When includeColumns parameter is specified and is true, column metadata of corresponding tables are included in the response. The metadata contains the following information about the tables included in the SkyNode:

- Name: Name of the table (String)
- MaxRecord: Maximum records to return. –1 if no limit (long)
- Description: Text description of table contents (String)
- UCD: UCD of the table (String)
- Class: Class name of the table. "general", "objects", "image" or "spectrum" (String)
- RowCount: Row count, if known, otherwise -1 (Long)
- Rank: Relative importance of the table, if known, otherwise -1 (Small Int). Larger numeric rank indicates higher importance.
- Pos_coord1: Name of a column that is referred to as a first coordinate in a region search if exists, otherwise empty string.
- Pos_coord2: Name of a column that is referred to as a second coordinate in a region search if exists, otherwise empty string.
- Pos_coord: Name of a position_2d data type column that is referred to as a coordinate in a region search if exits, otherwise empty.
- Frames: A list of supported frame names for the tables in regional query (Frame[])
- Last_modified: last modified time of contents of a table.
- Columns: An array of column metadata (ArrayOfMetaColumn). This is included if "includeColumns" parameter is true. When the parameter is not specified or is false, this is not included in a response.

```
ArrayOfMetaTable Tables([table])

<s:element name="Tables">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" name="table" type="s:string"/>
    </s:sequence>
  </s:complexType>
 </s:element>

<s:element name="TablesResponse">
  <s:complexType>
    <s:sequence>
```

```
      <s:element minOccurs="0" name="TablesResult"
       type="ArrayOfMetaTable" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ArrayOfMetaTable">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
    name="MetaTable" nillable="true" type="s0:MetaTable" />
  </s:sequence>
</s:complexType>

<s:complexType name="MetaTable">
  <s:sequence>
     <s:element name="Name" type="s:string" />
     <s:element maxOccurs="0" maxOccurs="unbonded" name="UCD"
     type="s:string" />
     <s:element maxOccurs="1" name="MaxRecords"
     type="s:long" />
     <s:element minOccurs="0" name="Class" type="s:string" />
     <s:element minOccurs="0" name="Rank" type="s:string" />
     <s:element minOccurs="0" name="PosCoord1"
     type="s:string" />
     <s:element minOccurs="0" name="PosCoord2"
     type="s:string" />
     <s:element minOccurs="0" name="PosCoord"
     type="s:string"/>
     <s:element minOccurs="0" ref="Frame"/>
     <s:element minOccurs="0" name="Description"
     type="s:string" />
     <s:element minOccurs="0" name="Rows" type="s:long" />
     <s:element minOccurs="0" name="lastModified"
     type="s:dateTime" />
     <s:element minOccurs="0" ref="ArrayOfMetaColumn"/>
  </s:sequence>
</s:complexType>
```

QI-12 SkyNodes SHALL implement the "**Columns**" interface, which takes an optional tableName parameter and an optional columnName parameter, and returns an array of MetaTable that contains information about columns. When neither of the tableName nor columnName parameters are specified, metadata of all the columns of the services are returned. When only the tableName parameter is specified, all the columns that belongs to the table are returned. When only the columnName parameter is specified, metadata of all the column of which the name is the same as that specified by the parameter are returned. When both of the parameters are specifed, metadata of the specified column are returned.

- ColumnName: Name of the column (String)
- TableName: Name of a table to which the column belons to (String).

- Description: Text description of table contents (String)
- DataType: ADQL data type name that will appear in the resulting VOTable if this column is requested (String)
- Units: Physical units of the data in the column, using standard IVOA nomenclature. Zero length string for dimensionless data (String).
- Frame: Coordinate frame if it is defined (Frame).
- Arraysize: Dimension of the column value. Positive integer or "*".
- Precision: Number of significant figures, either as a number of decimal places (e.g. precision ="F2" to express 2 significant figures after the decimal point), or as a number of significant figures (e.g. precision="E5" indicates a relative precision of 10^{-5}).
- Rank: Relative importance of the table if known, otherwise -1 (Small Int)
- UTYPE: UTYPE of the column if known, otherwise empty.
- OridinalPosition: position number (>=0) of the column in the table.
- UCD: Universal Content Descriptor. Zero length string if undefined or unspecified (String).
- ForeignKey: List of columns that this column has foreign key constraints on.
- PrimaryKey: true if the column is a primary key, otherwise false (boolean).

```
ArrayOfMetaColumn Columns([table] [,] [name])

<s:element name="Columns">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="table"
      type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="column"
      type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:element name="ColumnsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs=″unlimited″
         name="ColumnsResult" type="ArrayOfMetaColumn" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ArrayOfMetaColumn">
  <s:sequence>
```

```
      <s:element minOccurs="0" maxOccurs="unbounded"
      name="MetaColumn" nillable="true" type="MetaColumn" />
    </s:sequence>
  </s:complexType>

  <s:complexType name="MetaColumn">
    <s:sequence>
      <s:element name="Table" type="s:string" />
      <s:element name="Column" type="s:string" />
      <s:element name="DataType" type="s:string" />
      <s:element minOccurs="0" name="Unit" type="s:string" />
      <s:element minOccurs="0" name="Arraysize"
      type="s:string" />
      <s:element minOccurs="0" name="Precision"
      type="s:string" />
      <s:element minOccurs="0" maxOccurs="unlimited" name="UCD"
      type="s:string" />
      <s:element minOccurs="0" name="UType"
      type="s:string" />
      <s:element name="OrdinalPosition" type="s:int"/>
      <s:element minOccurs="0" name="PrimaryKey"
      type="s:boolean"/>
      <s:element minOccurs="0" name="ForeignKey"
      type="ArrayOfColumn"/>
      <s:element minOccurs="0" name="Rank" type="s:string" />
      <s:element minOccurs="0" name="Description"
      type="s:string" />
    </s:sequence>
  </s:complexType>

  <s:complexType type="ArrayOfColumn">
    <s:sequence>
      <s:element maxOccurs="unbounded" name="Column"
      type="columnType"/>
    </s:sequence>
  </s:complexType>

  <s:complexType type="columnType" mixed="false">
    <s:sequence/>
    <s:attribute name="name" type="s:string"/>
    <s:attribute name="table" type="s:string"/>
  </s:complexType>
```

QI-13 SkyNodes SHALL implement a "**Functions**" interface, which takes no parameters returns a structure that contains the following information about supported ADQL functions:

- FunctionName: Name of the function (String)
- Description: Text description of table contents (String)

- ParamList: Array of parameter specifications
  - ParamName: Name of the parameter (String)
  - ParamType: Data type of parameter (String)
  - ParamDescription: Text description of the parameter (String)
- OutputType: Data type of function result (String)

```
ArrayOfMetaFunction Functions(name)
```

```
<s:element name="Functions">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs=”0” name=”name” type=”s:string”/>
    </s:sequence>
  </s:complexType>
</s:element>

<s:element name="FunctionsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
      name="FunctionsResult" type="ArrayOfMetaFunction" />
    </s:sequence>
  </s:complexType>
</s:element>

<s:complexType name="ArrayOfMetaFunction">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
    name="MetaFunction" nillable="true" type="MetaFunction"/>
  </s:sequence>
</s:complexType>

<s:complexType name="MetaFunction">
  <s:sequence>
    <s:element name="name" type="s:string" />
    <s:element minOccurs="0" name="parameters"
    type="ArrayOfParameter" />
    <s:element minOccurs="0" name="description"
    type="s:string" />
  </s:sequence>
</s:complexType>

<s:complexType name="ArrayOfParameter">
  <s:sequence>
    <s:element maxOccurs="unbounded" name="Parameter"
    nillable="true" type="Parameter" />
  </s:sequence>
</s:complexType>
```

```
<s:complexType name="Parameter">
  <s:sequence>
    <s:element name="name" type="s:string" />
    <s:element name="type" type="s:string" />
    <s:element minOccurs="0" name="description"
    type="s:string" />
  </s:sequence>
</s:complexType>
```

QI-14 SkyNodes SHALL implement a "**Service**" interface, which takes no parameters returns a structure that contains all the metadata described above:

- ADQLSpec: An array of ADQL specification metadata (MetaADQLSpec[]).
- ReturnType: An array of supported format (MetaReturnType[])
- Tables: An array of MetaTable, each MetaTable contains an array of MetaColumn (MetaTable[]).
- Functions: An Array of MetaFunction (MetaFunction[]).

```
Service Service()
<s:element name="ServiceResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="ADQLSpecs"
      type="ArrayOfMetaADQLSpec/>"
      <s:element minOccurs="0" name="Formats"
      type="ArrayOfMetaFormat" />
      <s:element minOccurs="0" name="Tables"
      type="ArrayOfMetaTable" />
      <s:element minOccurs="0" name="Functions"
      type="ArrayOfMetaFunction" />
    </s:sequence>
  </s:complexType>
</s:element>
```

4 Content of the returned VOTable

QI-15 The following requirements are places on the contents of the returned VOTable:

- VOTable version 1.1 or higher MUST be used.
- The VOTable MUST contains a RESOURCE element, identified with the tag type="result", containing a single TABLE element with the result of the query.
- ADQL/s
- Name attribute of FIELD elements is filled by one of the following forms:
  - `<field_name>` : When a selected column originates from an uploaded VOTable, the fiel d name MUST be kept in the result.

  - `<alias_name>` : When a selected column or derived-column (expression) is aliased in ADQL, the name attribute of correspoing FIELD MUST be the alias name.
  - `<table_alias> . <column_name>` : When a selected column is not aliased the column name qualified by the table alias name SHOULD be supplied.
  - `<expression>` : When a derived-column is not aliased, the expression for the derviced-column SHOULD be supplied.

- ADQL-specific data type names, such as date, time, timestamp, time_interval, position_2d, region_2d and service specific data type, SHOULD be expressed by a PARAM element grouped with a FIELDref element by using a GROUP element.

```
<FIELD datatype="char" id="c1" name="time"
arraysize="*"/>
<GROUP>
    <FIELDref ref="col1"/>
    <PARAM name="datatype" datatype="char" arraysize="*"
        value="timestamp"/>
</GROUP>
```

- Metadata of each field SHOULD be supplied by FIELD element attribute or by PARAM element grouped with FIELDref element in an GROUP element.

```
<FIELD datatype="double" ID="col1" name="t.ra"
       unit="deg" ucd="pos.eq.ra;meta.main"/>
<FIELD datatype="double" ID="col1" name="t.ra"
       unit="deg" ucd="pos.eq.ra;meta.main"/>
<GROUP>
   <FIELDref ref="col1"/>
   <FIELDref ref="col2"/>
   <PARAM name="frame" datatype="char" arraysize="*"
          value="FK5">
</GROUP>
```

- The order of the FIELDs MUST follows the order of returned columns as specified in the ADQL specificiation .

5 Types of Skynode

QI-16 SkyNodes are classified into 5 levels of services based on the capability: ENTRY, XMATH, AGENT, ADVANCED, and FULL Skynode. ENTRY Skynode is the most fundamental one, and it accepts ADQL Core and implemnet small part the Skynode interfaces. ADVANCED Skynode accept MOST of ADQL extensions that are part of standard SQL. XMATCH SkyNode implement VOTableJoin interface and can do a distance-proximity join between an internal table with an external VOTable. Agent SkyNode can communicate with the other SkyNode and accept a plan file.

The following tables specify the required features to be classified to each SkyNode class.

SkyNode Methods	Entry	XMatch	Agent	Advanced	Full
DataExist	O	O	O	O	O
QueryCost	O	O	O	O	O
PerfomQuery	O	O	O	O	O
VOTableJoin		O			O
ADQLSpecs	O	O	O	O	O
Tables	O	O	O	O	O
Columns	O	O	O	O	O
Functions	O	O	O	O	O
ExcutionPlan			O		O
Footprint					O

ADQL Extension	Entry	XMatch	Agent	Advanced	Full
DST				O	O
OFF					O
EXP				O	O
INT					O
TML		O	O	O	O
TJN				O	O
TID					O
VOT		O			O
TSQ				O	O
EXI				O	O
GBY				O	O
OBY				O	O
HVN				O	O
FUN				O	O
CON					O
DOP				O	O
NAR					O
ITV					O
GEO					O
AGR				O	O

Function	Entry	XMatch	Agent	Advanced	Full
GC_distance					
Position					
Circle					
Box					
Join_chi2			O		O

Join_Distance		O	O		O

6 Resource Metadata specific for a SkyNode

QI-17 SkyNodes SHALL register with the registry with type="OpenSkyNode[S12]" according to the IVOA registry schema. Following skynode specific metadata SHALL be included in the resource metadata.

- Capability: (String[]) Entry, XMatch, Agent, Advanced, and/or Full.
- ADQLSpecs: Supported ADQL syntax.
  - ADQLVersion: (String)
  - ADQLExtensionIDs: (String[])
- Coverages: A list of coverages of table contents. (TableCorverage[])
  - TableName: table name (String).
  - Coverage: coverage of the table represented by VOResource Coverage complexType (Corverage).

7 Changes from previous versions

- None. This is the first release.

8 References[S13]

[1] IVOA VOQL Working group; IVOA Astronomical Data Query Language http://www.ivoa.net/Documents/latest/ADQL.html

[2] VOTable Format Specification 1.2 http://www.ivoa.net/Documents/latest/VOT.html

[3] Space-Time Coordinate for the Virtual Observatory 1.30 http://www.ivoa.net/Documents/latest/STC.html

---

Appendix[S14]

_____