Heliophysics Integrated Observatory

Project No.: 238969
Call: FP7-INFRA-2008-2

Service Interface Specification

Definition of Required Capabilities

Version 1.1

Title:	**Service Interface Specification**
Document No.:	HELIO-UCL-S2-003-RQ Deliverable No. **S1.8**
Date:	15 October 2013
Editor:	**Kevin Benson**, UCL-MSSL
Contributors:	Anja Le Blanc, R.D. Bentley
Distribution:	Project

Revision History

Version	Date	Released by	Detail
V0.1	20/02/12	K. Benson	1st Draft
V0.2	03/07/12	A. Le Blanc	Many comments
V1.0	24/07/12	K. Benson	Released version
V1.1	15/10/13	K. Benson	Minor Clean up on fonts and shading. UCD response of VOTable added. Plus WSDL and param updates on HQI

1 Introduction

Document describes each of the primary interfaces used as services in the HELIO architecture. HELIO clients as well as other external clients can view the specification and understand how to interact with each of the services. This document does not describe the location and connection of each of these interfaces in the HELIO framework. Primary interfaces are:

- HELIO Query Interface – Query interface used by the HELIO Framework for accessing the services as an HTTP GET/POST style queries.
- HELIO Query Interface Asynchronous – Query interface described for longer running queries where a standard HTTP GET/POST could not be used.
- Registry – Which follows an IVOA standard interface to describe resources in HELIO. Resources in this context can be the services, table metadata, applications, and other metadata such as observatories that were deemed useful for HELIO clients in the framework.
- Universal Worker Service (UWS) – Is the interface to describe a particular command-line based application that can be ran as a service over the web. For example HELIO clients can obtain a particular image or coordinate information by using the UWS Flare Plotting and Coordinate Transformation Services.

IVOA specifications are referenced in the HELIO Interfaces and specifications; specific versions are given as references below when applicable. All IVOA document can be obtained from their repository located at:

http://www.ivoa.net/documents/index.html

2 HELIO Query Interface

2.1 Introduction

HELIO Query Interface (HQI) is a Web-service, which enables the retrieval of information in a VOTable format, targeted at time-domain data though not restricted to just the time-domain. Queries are submitted using simple parameters and keywords that are based around HELIO Missions and Instruments. Queries are targeted for HELIO datasets but can be generally used for any time-based data sets. Queries may be submitted using the required HTTP-GET/POST protocol or SOAP based service.

This document formalizes the syntax and requirements of HQI as a higher-level parameter-based query language for querying tabular data. HQI complements the requirements of the IVOA Parameterized Query Language (PQL) and offers more expressive and powerful extensions described further in the document. HQI also allows SQL based searching when connected to JDBC compliant databases. Positional searches as described in PQL have also been extended to accompany the HELIO environment.

2.2 Document References

PQL specification:

http://www.ivoa.net/internal/IVOA/TableAccess/PQL-0.2-20090520.pdf

VOSI specification:

http://www.ivoa.net/documents/VOSI/20110531/REC-VOSI-1.0-20110531.pdf

VOTable specification:

http://www.ivoa.net/Documents/VOTable/20091130/REC-VOTable-1.2.pdf

2.3 Interface Requirements

Two types of interfaces are defined below: HTTP (using both GET and POST methods) and SOAP. A provider MAY implement one or both interfaces. Interfaces are registered as capabilities in a IVOA compatible Registry, which allows clients to determine the interface available for querying. Use of HTTP '**must'** support both GET and POST methods. Use of the HQI Component by HELIO, which connects to Relational Databases, implements both interfaces. HELIO primary GUI uses the SOAP based interface. It is advisable to implement both interfaces.

2.4 HTTP-GET

HTTP-GET interface requests a URL having two parts. Requirements are defined:

1. A base URL of the form:

- http://<server address>/path?[extra http-get argument&[...]]

Where <server address> and <path> are standard URI for the domain address and location path where the service is located. The [extra http-get arguments] are not part of this protocol and are viewed as part of the base URL.

- The URL **must** end in a '?' if no [extra http-get argument(s)] are specified otherwise it **must** end in a '&'.

Constraints are <name>=<value> pairs as the standard GET '**&**' (ampersand) parameters.

- <name> **must** be a parameter name defined in this specification.
- <value> **must** be the parameter name value.
- The <name> **must not** be case sensitive and <value> **must** be case sensitive.

2. The service must respond with an XML document in the VOTable format. The VOTable must adhere to the rules as described in section 2.9.

2.4.1 Order and cardinality of parameters

Parameters in a request **may** be specified in any order.

When request parameters are duplicated with conflicting values, the response from the service is undefined. The service **may** reject the request or it **may** pick one value for the parameter. Clients **should not** repeat parameters in a request.

2.5 HTTP-POST

Parameters may be submitted as HTTP-POST requests to the HTTP interface.

2.6 Soap Based Service

A SOAP implementation of the HQI is available and **must** follow the Web Service Description Language (WSDL) described in the table below. The SOAP interface is similar to the HTTP-GET interface by following many of the parameters, but allow clients to create a contract with the WSDL.

HQI WSDL -- http://msslkz.mssl.ucl.ac.uk/helio-ics/HelioService1_1?wsdl

Asynchronous HQI WSDL -- http://msslkz.mssl.ucl.ac.uk/helio-ics/HelioLongQueryService1_1?wsdl
**More details of Asynchronous HQI in section 3.

2.7 Parameters

Parameters and Specification constraints are defined below. Parameters in **BOLD** are required:

- STARTTIME – ISO8601 input – YYYY-MM-dd['T'HH:mm:ss[SSS]]
- ENDTIME – ISO8601 input – YYYY-MM-dd['T'HH:mm:ss[SSS]]
    - Both start time and end time can have one or array of values; the output response will have data for multiple time (includes start time and end time). Remember MAXRECORDS value is applicable for each pair of start and end time. For example MAXRECORDS=10 and start and end time has each 2

values. VOTable returned will have 2 tables having (maximum) 10 records each.
    - Arrays of STARTTIME and ENDTIME values should have the same length. They are combined in a dot product fashion (first with first, second with second, and so on). Arrays with different lengths will be combined up to the shortest length.
- TIME – Not used by the HELIO interface, but defined in the PQL Generic Dataset section. This parameter is best described as STARTTIME/ENDTIME ISO8601 format separated by a '/'. HELIO Query Interface understands this parameter in case of use by external clients.
- **FROM** – Table or Instrument name to query on, one or many tables can be queried; each table name should be separated by comma.
    - If an Instrument name is used in the FROM field the service **must** define a '<SupportedInstrument>' element in the registration defining instruments the client may query on. Instrument names are defined in a separate file here: http://msslkz.mssl.ucl.ac.uk/helio-ics/HelioService1_0?xsd=1
    - The Service provider **must** use these Instrument names for convention.
- MAXRECORDS – Maximum number of records the user wants to return. A server may have defined its own maximum records that cannot be overridden by the user.
- STARTINDEX – Index number of the first row in the dataset. The client **should** use this when the maximum number of rows returned by the server is reached.
- POS, SIZE – See the PQL specification, Section 3.1.1 and the detailed section on Positional queries below.
- REGION – See the PQL specification, Section 3.1.2 and the detailed section on Positional queries below.
- SELECT – See the PQL specification, Section 3.1.3. Optional and the default will be all columns.
- WHERE – Where clause that follows the PQL syntax.
- SQLWHERE – Where Clause that follows the standard SQL syntax.
- IVOA TAP Conformance (Optional) – Described in Section 2.7.7 below.

2.8 Details of Parameter

2.8.1 FROM

The FROM parameter is required and **must** include at least one valid table or instrument name. In order to perform multiple queries the FROM parameter values are to be separated by commas. Responses of a VOTable as read in section 3.3 **must** contain one 'Resource' per query.

2.8.2 SELECT

The Select parameter is optional, but can be used to reduce the columns returned in the query result object. Column names **must** be separated with commas. When the FROM parameter has multiple tables defined the SELECT parameter **must** include a table name joined with a '.' separator on the defined column name.

- The following syntax **must** be followed: TableName.ColumnName when the FROM has multiple tables defined.
- When only one table is defined in the FROM parameter the SELECT parameter should handle both cases of:
    - Only a column name or

- TableName.ColumnName

One HELIO component (DPAS) – Data Provider Access Service, which is not connected to a relational database, uses the SELECT parameter to choose a particular provider. DPAS automatically selects providers with the highest rank defined in the PAT table, Provider Access Table. Clients may specify a particular provider by using the 'SELECT' parameter for the SOAP and Restful services.

A view of all the Providers and rankings can be found at '/HelioPatServlet'. Example viewing of the common HELIO DPAS can be found here.
http://msslkz.mssl.ucl.ac.uk/helio-dpas/HelioPatServlet

Tests
http://msslkz.mssl.ucl.ac.uk/helio-dpas/HelioQueryServlet?STARTTIME=2009-02-14T20:00:00&ENDTIME=2009-02-15T03:59:59&FROM=SOHO__EIT&SELECT=MEDOCSOHO

2.8.3 WHERE

The WHERE parameter follows the PQL specification, except when the FROM parameter is overloaded with more than one Table:

- Column names **must** include a Table Name defined in the FROM parameter followed by a '.' when more than one table is defined in the FROM parameter. This syntax is optional when 'only' one FROM parameter is defined to follow the PQL specification.
  - e.g. TableNameA.columnName

Examples:
SQL for one table:
WHERE=TableA.mag,4
SQL: Where TableA.mag=4
WHERE=TableA.mag,/4.8
SQL: Where TableA.mag<=4.8
WHERE=TableA.mag,4.7/8.1
SQL: Where TableA.mag between 4.7 and 8.1
Complex Where taken from PQL document
WHERE=vmag,4.5/5.5; imag,4.5/; bmag,/5.5; flag,4,5,6; jmag,4.5/5.5,/3.0,9.0/; name,*Lon*; kmag,4.5/5.5; flux,null; last,1
SQL: vmag between 4.5 and 5.5 and imag >= 4.5 and bmag <= 5.5 and (flag = 4 or flag = 5 or flag = 6) and (jmag between 4.5 and 5.5 or jmag <= 3.0 or jmag >= 9.0) and (name like '%Lon%') and (kmag between 4.5 and 5.5) and (flux is null) and (last = 1)
MULTIPLE SQL:
FROM=TableA,TableB
WHERE=TableA.mag,/4.8;TableB.wave,7.2/
SQL-TableA: Where TableA.mag<=4.8
SQL-TableB: Where TableB.wave >= 7.2
(It is advisable to put your table constraints together but as demonstrated in the below example it is not required.)
FROM=TableA,TableB
WHERE=TableA.mag,/4.8;TableB_wave,7.2/;TableA.pos,3;TableB.flag,7
SQL-TableA: Where TableA.mag<=4.8 and TableA.pos=3
SQL-TableB: Where TableB.wave >= 7.2 and TableB.flag=7

```
FROM=TableA,TableB
WHERE=TableA.mag,/TableB.mag
```

2.8.4 SQLWhere

HQI services that are connected to relational databases may also place a direct SQL Where clause as a parameter. Certain clients that construct SQL clauses can use this parameter to query an HQI service. A client should be prepared to view the Registry contents of a service to determine if the SQLWhere parameter can be used on Restful Service.

The SOAP has a 'SQLSELECT' interface that uses the standard SQL Where syntax for queries.

2.8.5 POS, SIZE, REGION

The POS and SIZE parameters provide a straightforward facility for performing spatial queries of astronomical catalogs, index tables, or other tables, which are spatially indexed. The use of the parameters follows PQL convention, though it is recommended to specify the coordinate system with the POS:

POS=X,Y,Z;CRS

Where CRS corresponds to the Coordinate Reference System (e.g., HCI for HelioCentric Inertial System), and X,Y,Z are respectively, the x-axis, the y-axis, the z-axis values of the corresponding CRS for which data are requested.
Example: POS=X,Y,Z;HCI&SIZE=dX,dY,dZ;Rectangle

More information can be found in the HELIO Spatial and PQL documents.

2.8.6 MAXRECORDS, STARTINDEX

The service **must** accept a *MAXRECORDS* parameter specifying the maximum number of table records (rows) to be returned. If *MAXRECORDS* is not specified in a query, the service **may** apply a default value or **may** set no limit. If the result set for a query exceeds this value, the service **must** only return the requested number of rows. If the result set is truncated in this fashion, it **must** include an overflow indicator as specified in section 2.8.2. STARTINDEX defines where the records should start. Clients may view the VOTableINFO elements to determine if all resources have been reached.

2.8.7 IVOA TAP Protocol Conformance (Optional)

With minor additional parameters a HELIO Query Service Implementation can conform to the primary query mechanism that is defined in the IVOA Table Access Protocol. By conforming to the TAP protocol would allow other clients that conform to the specification to query on HELIO Services. The HELIO Query Interface Component implements these optional parameters to allow IVOA clients to query on the HELIO services. Parameters to be added:

- REQUEST parameter that will be set to 'doQuery'

- LANG parameter that '**must'** be either ADQL or PQL.
- QUERY parameter used when the LANG is set to ADQL. The QUERY parameter is a string version of the ADQL. In majority of cases this is a SQL string and is expected for HELIO support at this stage to be a standard SQL string.
- MAXREC is the same as MAXRECORDS as described in section 2.7.6
- /sync – TAP expects http-get (or POST) queries to be registered with a '/sync' base url.

The full IVOA TAP protocol can be described here: http://www.ivoa.net/documents/TAP/20100327/REC-TAP-1.0.html. Following section 2.7.7 points, majority of clients would be able to query on HELIO services.

2.9 Response of the interface

The Response of an interface **must** conform to XML VOTable 1.2 format conforming to the IVOA VOTable schema http://www.ivoa.net/xml/VOTable/VOTable-1.1.xsd.

The Requirements of the VOTable **must** follow the requirements listed below:

- MIME type **should** be in the form of '*text/xml*' or optionally '*text/xml;x-votable*'
- There **must** be a '*Resource*' element per data set query and that 'Resource' contains a single '*TABLE*'.
- Each 'Resource' element **must** contain an 'INFO' element with a 'name' attribute set to "QUERY_STATUS" with the 'value' attribute set to either "OK" or "ERROR".
- The 'TABLE' **must** have 'FIELD' elements with UCD and UTYPE attributes. Only one 'FIELD' allowed for each UTYPE.
- HELIO Query Interfaces usually contain time information and other common response fields. HELIO interfaces should use these ucd and utypes:
  - TIME – UCD: time.phase Utype: helio:time.time Should use defined as xtype=iso8601
  - Julian Integer – UCD: time Utype: helio:trajectories.julian_date.julian_date_int
  - Images – UCD: VOX:Image_AccessReference (VO Extension ucd used by other clients to recognise a particular image field in the votable)
- The service **must** respond with a VOTABLE in the case of error. The VOTABLE **must** contain a single '*INFO*' element with the 'name' attribute set to 'ERROR' e.g. name="ERROR" with the corresponding 'value' attribute explaining the error, if the error occurs before any processing of results. Errors such a Overflow, Invalid Parameters, Invalid Query, and Writing result streams are defined in the following sections and **must** have this VOTABLE returned when encountered.
- Must be one Resource with a Table per dataset query.
- If more than one STARTTIME and ENDTIME pair are used in a query, then each Table element of the VOTABLE **must** correspond to a pair of STARTTIME and ENDTIME.

2.9.1 Successful

Example:

```
<?xml version="1.0"?>
<!DOCTYPE VOTABLE SYSTEM "http://us-vo.org/xml/VOTable.dtd">
<VOTABLE version="1.0">
<RESOURCE ID="SOHO_EIT456">
<DESCRIPTION>
SOHO EIT and CDS time based query.
Note the second Table below.
</DESCRIPTION>
<TABLE ID="SOHO_EIT_TABLE_Results">
<DESCRIPTION> SOHO EIT</DESCRIPTION>
<FIELD name="unique_id" datatype="char" arraysize="*" ucd="ID_MAIN">
<DESCRIPTION> Integer key </DESCRIPTION>
</FIELD>
<FIELD name="FitsURL" datatype="char" arraysize="*" ucd=”VOX:Image_AccessReference”>
<DESCRIPTION> Url to the Fits file. </DESCRIPTION>
</FIELD>
<FIELD name="StartTime" datatype="char" arraysize="*" ucd=”VOX:START_TIME”>
<DESCRIPTION> Observing start time. </DESCRIPTION>
</FIELD>
<FIELD name="EndTime" datatype="char" arraysize="*" ucd=”VOX:END_TIME”>
<DESCRIPTION> Observing end time. </DESCRIPTION>
</FIELD>
<DATA>
<TABLEDATA>
<TR>
<TD>384559</TD><TD>http://www.sohodata/eit/2008/09/22/fits/eit.fits</TD>
<TD>2008-02-20T21:33:10</TD><TD>2008-02-20T21:34:10</TD>
</TR>
<TR>
<TD>384559</TD><TD>http://www.sohodata/eit/2008/09/22/fits/eit.fits</TD>
<TD>2008-02-20T21:33:10</TD><TD>2008-02-20T21:34:10</TD>
</TR>
</TABLEDATA>
</DATA>
</TABLE>
<TABLE>
<DESCRIPTION> SOHO CDS</DESCRIPTION>
</TABLE>
</RESOURCE>
</VOTABLE>
```

2.9.2 Overflows

If a query is executed by a HQI service, the number of rows in the table of results may exceed a limit requested by the user (using the MAXRECORDS parameter) or a limit set by the service implementation (the default or maximum value of MAXRECORDS). In these cases, the query is said to have 'overflowed'. Typically, a HQI service will not detect an overflow until some part of the table of results has been sent to the client.

If an overflow occurs, the HQI service **must** produce a table of results that is valid, in the required output format, and which contains all the results up to the point of overflow. Since an output overflow is not an error condition, the MIME type of the

output **must** be the same as for any successful query and the HTTP status-code **must** be as for a successful, complete query.

If the output format is VOTable, section *"Error: Reference source not found"* describes the method by which the overflow is reported. No method of reporting an overflow is defined for formats other than VOTable.

Example:
```
<RESOURCE type="results">
<INFO name="QUERY_STATUS" value="ERROR">
<DESCRIPTION>unrecognized operation</DESCRIPTION>
</INFO>
<INFO name="SPECIFICATION" value="TAP"/>
<INFO name="VERSION" value="1.0"/>
<INFO name="REQUEST" value="doQuery"/>
<INFO name="baseUrl" value="http://webtest.aoc.nrao.edu/ivoa-dal"/>
<INFO name="serviceVersion" value="1.0"/
...
</RESOURCE>
```

If an overflow occurs (result exceeds MAXRECORDS), the service **must** close the table and append another INFO element to the RESOURCE (after the TABLE) with *name="QUERY_STATUS"* and the value="*OVERFLOW".*

Example:
```
<RESOURCE type="results">
<INFO name="QUERY_STATUS" value="OK"/>
...
<TABLE>...</TABLE>
<INFO name="QUERY_STATUS" value="OVERFLOW"/>
</RESOURCE>
```

In the above example, the TABLE should have exactly MAXRECORDS rows.
If the TABLE does not contain the entire query result, one INFO element with *value="OVERFLOW"* or *value="ERROR"* **must** be included after the table.

2.9.3 Error – Invalid Parameter, Invalid Query, Writing of Results

Example Invalid Parameter:

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

<!DOCTYPE VOTABLE SYSTEM "http://us-vo.org/xml/VOTable.dtd">
<VOTABLE version="1.0">
<DESCRIPTION>SOHO EIT query service</DESCRIPTION>
<INFO name="EXECUTED_AT" value="2010-06-04 19:02:31"/>
<INFO name="QUERY_STRING">
SELECT obs_id,obs_name,obs_type,obs_start_date,obs_end_date,obs_operation_type FROM
observatory WHERE ( obs_start_date>='1890-AA-20T20:30:56' AND obs_end_date<='2009-10-20
0:30:56' ) AND obs_id like '%%' ORDER BY observatory.obs_start_date LIMIT 5000
</INFO>
<INFO ID="Error" name="Error" value="Start Time could not be parsed"/>
```

```
</VOTABLE>
```

Example of Invalid Query:

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

<!DOCTYPE VOTABLE SYSTEM "http://us-vo.org/xml/VOTable.dtd">
<VOTABLE version="1.0">
<DESCRIPTION>SOHO EIT query service</DESCRIPTION>
<INFO name="EXECUTED_AT" value="2010-06-04 19:02:31"/>
<INFO name="QUERY_STRING">
SELECT obs_id,obs_name,obs_type,obs_start_date,obs_end_date,obs_operation_type FROM
observatory WHERE (incorrect_start_date  >='1890-AA-20T20:30:56' AND obs_end_date<='2009-10-
20 0:30:56' ) AND obs_id like '%%' ORDER BY observatory.obs_start_date LIMIT 5000
</INFO>
<INFO ID="Error" name="Error" value="Query Error: Unknown column incorrect_start_date"/>
</VOTABLE>
```

If an error occurs while writing the rows of the VOTable, the service **must** close the table and append another INFO element to the RESOURCE, after the TABLE, with *name="QUERY_STATUS"* and the value="ERROR*".*

Example:

```
<RESOURCE type="results">
<INFO name="QUERY_STATUS" value="OK"/>
...
<TABLE>...</TABLE>
<INFO name="QUERY_STATUS" value="ERROR" />
</RESOURCE>
```

The content of these trailing INFO elements is optional and intended for users; client software **should not** depend on it.
Thus, one INFO element with *name="QUERY_STATUS"* and *value="OK"* or *value="ERROR"* **must** be included before the TABLE.

2.10 Registration of HELIO Query Interface

The HELIO Registry component describes metadata about a particular service. HQI has optional capabilities depending on what the service allows. This information needs to be registered so it can be presented to clients to distinguish what is capable. Table metadata for the HQI service should also be registered though it is not required. Capabilities and Table metadata uses the IVOA VOSI (VO Support Interface) to be described, the HELIO Registry can harvest this data to be placed into the Registry.

2.10.1 Capabilities and Extension

Capability metadata elements in the Registry describe the location of the service(s) along with an ID that corresponds to a particular standard. A Resource in the Registry may describe several capabilities focused on a particular standard the service implements. The HQI defines two capabilities for the Query Service. One

for the HTTP GET/POST style 'ParamHTTP' and another standard for SOAP 'WebService', this allows clients to more easily query for a particular standard. Registry metadata allows the SOAP and REST to be mixed into one capability, but we discovered that various Registry parsers sometimes ignored mixed interfaces in a single capability. This provoked us to separate them into two different capability elements. Multiple mirrored copies of the same service are also demonstrated in the sample, by defining two accessURL elements in the same interface.

SQL and Positional elements are optional and they are captured in a special extension to the IVOA registry metadata.

Sample of HQI Capability

```
<capability standardID="ivo://helio-vo.eu/std/FullQuery/Soap/v1.0">
<interface xsi:type="vr:WebService">
<accessURL use="full">
http://msslkz.mssl.ucl.ac.uk/helio-ils/HelioService
</accessURL>
<accessURL use="full">
http://msslkr.phys.ucl.ac.uk/helio-ils/HelioService
</accessURL>
<SQLEnabled>true</SQLEnabled>
<PositionalQueryEnabled>true</PositionalQueryEnabled>
</interface>
</capability>
<capability standardID="ivo://helio-vo.eu/std/FullQuery/v1.0">
<interface xsi:type="vs:ParamHTTP">
<accessURL use="full">
http://msslkz.mssl.ucl.ac.uk/helio-ils/HelioQueryService
</accessURL>
<accessURL use="full">
http://msslkr.phys.ucl.ac.uk/helio-ils/HelioQueryService
</accessURL>
<SQLEnabled>true</SQLEnabled>
<PositionalQueryEnabled>true</PositionalQueryEnabled>
</interface>
</capability>
```

2.10.2 Table metadata

Table metadata for a Catalogue service can be useful for the clients to construct queries. Table metadata **must** be available at the service, and it is recommended that they be registered as a separate Resource in the Registry.

This separate resource **must** identify the HQI query service using a relationship tag.

```
<relationship>
<relationshipType>serviced-by</relationshipType>
 <relatedResource>ivo://helio-vo.eu/ils</relatedResource>
</relationship>
```

Table metadata describes the Catalogue along with Columns, UCD, descriptions and datatypes. This data would allow certain clients to construct a knowledgable query for a user.

```
<table>
<name>instruments</name>
<column>
 <name>ins_id</name>
 <description>Instrument Name</description>
 <ucd>meta.id;instr</ucd>
</column>
<column>
<name>ins_start_date</name>
<description>Instrument Start Date</description>
<ucd>time.start</ucd>
</column>
</table>
```

2.10.3 VOSI Registration

Registry will harvest VOSI capabilities for particular metadata; this information is harvested and placed into the Registry. VOSI is an IVOA standard VO Support Interfaces that describes capabilities and tables. Capabilities are the urls that define the services and tables that describe table metadata where applicable. This allows technical metadata that would normally come from a service or database to be harvested, while scientific core data could be submitted manually. HELIO follows the IVOA VOSI specification defined at:
http://www.ivoa.net/documents/VOSI/20110531/REC-VOSI-1.0-20110531.pdf

3 Asynchronous and Long Running Queries

Queries that can take a considerable amount of time or have the advantage of saving the results on the server, should consider implementing Long Running Query Interface. This interface allows the client to poll the server to discover when a query is completed.

SOAP and HTTP-GET interfaces are available.

3.1 Base URL for HTTP GET

All Long Running Queries work off the base URL and by using the same parameters.

3.1.1 MODE=query

To start a query in the Long Running Interface a MODE=query **must** be one of the parameters along with the query constraints defined in the 'Query Interface' section above.

3.1.2 Response

Response should be an xml page of simply an ID.

<ID>{unique id}</ID>

Example:

```
<ID>HQ21296</ID>
```

3.2 MODE=phase&ID={id}

To check the status of a query set a parameter MODE=phase along with the unique ID given from the query.

3.2.1 Response

```
<Status><ID>HQ61334</ID>  <status>COMPLETED</status>  <description>query completed</description></Status>
```

3.2.2 Error

```
<Status><ID>HQ61334</ID> <status>ERROR</status> <description>Improper Date and Time</description></Status>
```

3.3 MODE=result&ID={id}

Results are given with a MODE=result followed by the ID parameter given from the query.

3.3.1 Response

```
<ResultInfo><ID>HQ61334</ID>
<resultURI>http://manunja.cesr.fr/MDES/MDES_HQ61334.xml</resultURI> <fileInfo></fileInfo>
<status>COMPLETED</status>
<description>Data Is Ready</description>
</ResultInfo>
```

3.4 SOAP Interface

The SOAP interface is similar to HTTP-GET, but is described in a Web Service Description Language (WSDL). The WSDL standard allows clients to create a contract with the Service and have access to all methods.

HQI WSDL -- http://msslkz.mssl.ucl.ac.uk/helio-ics/HelioService1_1?wsdl

Asynchronous HQI WSDL -- http://msslkz.mssl.ucl.ac.uk/helio-ics/HelioLongQueryService1_1?wsdl

3.5 Registration of Long Running Query Interface

Similar to HQI but capabilities ID conforms to a different standard.

```
<capability standardID="ivo://helio-vo.eu/std/LongFullQuery/Soap/v1.0">
<interface xsi:type="vr:WebService">
<accessURL use="full">
http://msslkz.mssl.ucl.ac.uk/helio-ils/HelioLongQueryService
</accessURL>
<accessURL use="full">
http://msslkr.phys.ucl.ac.uk/helio-ils/HelioLongQueryService
</accessURL>
<SQLEnabled>true</SQLEnabled>
<PositionalQueryEnabled>true</PositionalQueryEnabled>

</interface>
</capability>
<capability standardID="ivo://helio-vo.eu/std/LongFullQuery/v1.0">          <interface
xsi:type="vs:ParamHTTP">
<accessURL use="full">
http://msslkz.mssl.ucl.ac.uk/helio-ils/LongRunningQueryService
</accessURL>
<accessURL use="full">
http://msslkr.phys.ucl.ac.uk/helio-ils/LongRunningQueryService
</accessURL>
<SQLEnabled>true</SQLEnabled>
<PositionalQueryEnabled>true</PositionalQueryEnabled>

</interface>
</capability>
```

4 Registry

The Registry will allow a client to be able to locate, get details of, and make use of, any resource located anywhere in the IVO space, on any Virtual Obervatory such as the HELIO Virtual Observatory. The IVOA will define the protocols and standards whereby different registry services are able to interoperate and thereby realise this goal.

4.1 Identifier and XML Schema Standards

The Registry is designed to describe resources, information, protocols, and services in the form of XML Schemas. These schemas can be found at: http://www.ivoa.net/xml/index.html

Identifiers are a global unique identifier for every resource of the Registry. Each resource contains an identifier of the form of URI following this syntax: ivo://{autorhityId}/{resourcekey}

The authority id is a type of naming authority or organization. A Registry may manage many authorities, but the standard one used by HELIO currently is: 'helio-vo.eu'. The resource key is unique to that naming authority. Although no real meaning is given to a resource key it is common practice to give a name that is understandable. Software clients may use identifiers to quickly get at a Resource in a registry, but it is common for identifiers to be hidden from the Users.

See 'How to use Registry' document on creating new entries. More information on identifiers may found here:

http://www.ivoa.net/Documents/REC/Identifiers/Identifiers-20070302.html

4.2 Interface or Reference

The Registry component implements the IVOA Registry interface, that can be found at: http://www.ivoa.net/Documents/RegistryInterface/

4.3 Harvesting HELIO Services

The Registry interface defines a Harvesting interface for Harvesting other Registries. This interface allows the concepts of global Registries to be created which will contain all the Resources of a 'publishing' registry. This Harvesting interface is based on an Open Archives Initiatives (OAI) concept. The HELIO Registry has this harvesting interface, but also has another ability to harvest Virtual Observatory Standard Interface (VOSI) documents or even full Resource entries maybe pulled from a service into a registry. Two types of VOSI resources that HELIO utilizes are:

Capabilities: This is a XML capabilities element following the Registry XML Schemas that define capabilities and location of services. One service may define several capabilities i.e. ParamHTTP service (REST), SOAP, or other VOSI services i.e. tables.

Tables: is XML data following Registry XML Schemas based around Catalogues that define table metadata of a particular Catalogue.

See ‘How to use Registry’ on starting a Harvest of VOSI documents. The Registry will harvest a ‘Capabilities’ and automatically fill the resource in the Registry. If other VOSI information is noticed such as Tables, this is also picked up and harvested into the desired resource.

5 Running Applications – Universal Worker Service - HELIO Context Service

The Context Service primary ability is to run applications on a remote server asynchronously. The applications can handle any number of inputs and outputs; the outputs are URL references to the results. Applications that are run remotely **must not** be interactive, meaning they cannot respond to user input in the middle of an application. Applications may produce any type of file, but **must not** be GUI based requiring a Graphical interface.

5.1 Interface

The Interface to the UWS follows the IVOA standard. Located here: http://www.ivoa.net/Documents/UWS/index.html

5.2 Job Language

The UWS interface only references a Job Language to be used but does not require the Job Language. In HELIO we adopted software from a previous project, which used its own XML Job Language and later conformed to the IVOA UWS interface. The XML schema that describes the Job Language is now part of the IVOA schema repository. The JDL described is a general descriptive XML, which accommodates a series of inputs and outputs.

5.2.1 Details of Job Description Language (JDL):

The JDL uses an xml schema defined by a previous project component called CEA Common Execution Architecture. The XML allows you to enter the inputs and outputs to run a particular application. These inputs and outputs are defined in the registry and originated from the setup of the CEA component on a server. The JDL allows you to specify if a parameter is 'indirect'. Setting a parameter of indirect to 'true' will allow you to place a reference for the input i.e. commonly a URL reference.

Outputs required a modification for HELIO, the current CEA component required outputs to go to a particular storage space in the IVOA i.e. known as VOSpace. Though the CEA also kept the output internally on the server. VOSpace was not being used at the time of HELIO and a decision was made to allow the Output to use a special reference URI reference of 'internalstorage://'. This allows CEA component to detect the special URI and return back an 'http' URL reference that is internal to the CEA component.

Example:

```
<tool xmlns="http://www.ivoa.net/xml/CEA/types/v1.2" id="ivo://helio-vo.eu/cxs/goesplotter"
interface="simple">
 <input>
   <parameter id="StartDate" indirect="false">
   <value>2000-01-30T00:00:00</value>
     </parameter>
 <parameter id="EndDate" indirect="false">
  <value>2000-12-31T03:59:59</value>
     </parameter>
```

```
 <parameter id="Type" indirect="false">
  <value>Proton</value>
     </parameter>
</input>
<output>
   <parameter id="goes_plot.png" indirect="true">
     <value>internalstorage:/</value>
 </parameter>
</output>
</tool>
```

5.3 Registration of UWS services

The HELIO UWS component installation and configured will contain a registration URL that contains the whole registration document. All metadata content are described with the CXS component, this allows Registration by uploading a CXS Registration URL to the Registry. The Registry will take the URL expecting XML in the format that corresponds to the IVOA XML schemas and insert it into the database. Example of a registration can be found here: http://msslkz.mssl.ucl.ac.uk/cxs/uws/reg. See 'All resources in single document' to view the contents the Registry will contain and separate each entry into a separate Resource into the Registry.