SPIRE

SUBJECT: Telemetry and Data Frame Interface Control Document

PREPARED BY: S.Guest

DOCUMENT No: **SPIRE-RAL-DOC-000788**

ISSUE: 0.1 ***draft*** **Date:** **21st August 2001**

APPROVED BY: **Date:**

Agreed By			Date

Distribution

Otto H. Bauer
Milena Benedettini
Jon Brumfitt
Odile Coeur-Joly
Albrecht de Jonge
Wim De Meester
Matt Fox
Kevin Galloway
Matthew Graham
Steve Guest
Ana Heras
Rik Huygen
Do Kester
Ken King
Tanya Lim
Jean-Jacques Mathieu
Seb Oliver
Sabine Osterhage
Göran Pilbratt
Craig Porrett
Rafael Prades
Timo Prusti
Johannes Riedinger
Peter Roelfsema
Hassan Siddiqui
Sunil Sidher
Eckhard Sturm
Bart Vandenbussche
Stephane Veillat
Ekkehard Wieprecht
Erich Wiezorrek
Peer Zaal
Rob Zondag

Change Record

ISSUE **DATE**

TABLE OF CONTENTS

FIGURES

TABLES

Glossary

1. INTRODUCTION

1.1 Purpose

This document specifies an interface for accessing packets and data frames. It is not restricted to database retrieval.

Originally it was envisaged that separate mechanisms would be used to access packets and data frames. It became evident that there were enough similarities in the requirements to combine the two, thus avoiding duplication of effort.

This interface specifies a one-way (read-only) protocol for homogeneous access to packets and data frames. It is not designed for clients wishing to update these objects, but database clients are not prevented from doing so. Neither is it a generic object server. The concept is based on data frames being the starting point for data processing, and therefore providing a convenient method for their access. As noted above, the requirements for packets are similar.

The intention is that objects can be streamed from a variety of sources, including over networks. Clients are not required to have a Versant installation (though they will be able to do more if they do). This approach also allows the possibility to design IA as an internet-aware application.

1.2 Scope

This interface applies to all mission phases, not just ILT, in accordance with the concept of smooth transition. Thus it is intended for all of RTA, QLA and IA.

It includes:

- Getting them from a local database.
- Getting them from a database across a network, including via the Internet.
- Getting them from local (non-Versant) persistent storage.
- Getting them from the router.

It does not include:

- Any access to objects other than packets or data frames. However, *some* elements of the design *might* prove to be applicable to other objects, particularly products.
- Any means of writing to any persistent storage, including the database. This functionality is provided by other interfaces, such as the data formatter for writing external files.
- A direct interface to the telemetry ingestor. This is expected to be via the router. *This works fine for packets – does NRT access to data frames need an interface to the ingestor?*

However, it does not *preclude* normal database operations *if* the client is a Versant one. In other words, it imposes no restrictions on trusted database clients.

1.3 Applicable Documents

AD1	TM/Data Frame Interface Technical Note	SPIRE-RAL-NOT-000717	1.0	12 June 2001
AD2	Herschel EGSE Packet Router ICD	SRON-G/EGSE/ICD/2001-xxx	draft2	8 February 2001

1.4 Reference Documents

RD1	FIRST/Planck Packet Structure ICD	SCI-PT-ICD-7527	Issue 1.0	1 September 2000
RD2	Herschel Science Ground Segment to Instruments	FIRST-FSC-DOC-2000	Issue 1	13 July 2001
RD3	HCSS User Requirements Document	FIRST/FSC/DOC/0115	Issue 2.0	3 May 2001

1.5 List of Acronyms

AD	Applicable Document
API	Application Programming Interface
BBID	Building Block Identifier
CCM	Core Class Model
EGSE	Electrical Ground Support Equipment
HCSS	Herschel Common Science System
HTTP	Hypertext Transfer Protocol
IA	Interactive Analysis
ICC	Instrument Control Centre
ICD	Interface Control Document
ILT	Instrument Level Test
JVM	Java Virtual Machine
MIB	Mission Information Base
NRT	Near Real Time
OBSID	Observation Identifier
ODBMS	Object DataBase Management System
QLA	Quick Look Analysis
RD	Reference Document
RTA	Real-Time Assessment
SCOS	Spacecraft Operating System
TCP/IP	Transfer Control Protocol / Internet Protocol
TM	Telemetry
URL	Uniform Resource Locator

2. OPERATIONAL ASSUMPTIONS AND CONSTRAINTS

2.1 Communications

Access is required to a database and to a router. No assumptions are made about network configuration.

2.2 Hardware

The interface is applicable to all machines supplying the required software environment.

2.3 Software

- A Java 1.2 or later run-time environment is required for all components.
- Database components are tested with the Versant 6.0 ODBMS. They may well run on other versions of Versant. Database clients will run on Solaris, Linux or Windows NT.
- HTTP access (if used) requires a web server. Any standard server (eg Apache, IIS) will work. *HTTP access is not yet implemented.*

2.4 User

N/A

2.5 Timing

N/A

3. REQUIREMENTS

3.1 Functional Requirements

These requirements have been extracted from the output of Action CSDT#5/2-2 (see **[AD1]**), compiled by Kevin Galloway, and modified to be less ILT-specific.

Func-Req-1: The HCSS shall provide telemetry source packets and data frames to both internal and external clients.

Func-Req-2: Clients shall be able to request:

1. Near real time (NRT) access to telemetry source packets
2. NRT access to data frames
3. Access to telemetry source packets in the database based on various selection criteria
4. Access to data frames in the database based on various selection criteria

The selection criteria currently identified for the request types 3 and 4 above are:

i. Give me telemetry source packets/ data frames as they arrive (near real-time connection)
ii. Give me telemetry source packets/ data frames for specified time period
iii. Give me telemetry source packets/ data frames associated with observation execution x
iv. Give me data frames associated with observation x and building block y

Note 1: Bullet (iv) Currently all telemetry source packets are associated with an observation as not all packets contain, or have accessible, a BBID (see core class model).

The client developers will be provided with an interface, which they can use to perform these requests.

Func-Req-3: The interface used by clients shall be the same for all of the above request types. The developer simply selects the request type/ connection he wants. Irrespective of the request type the telemetry source packets and/ or the data frames should be made available to the developer in the same way.
Func-Req-4: It must be possible to serve data to a remote site, even through a firewall.
Func-Req-5: It shall not be possible for an unauthorised user to modify the database in any way.[1] This implies a mechanism by which users can be authorised.
Func-Req-6: All data rights with respect to access shall be respected.

3.2 On-Line Delivery Requirements

OnLD-Req-1: Near real time delivery shall be in <1 second (**TBC**).
OnLD-Req-2: Delivery from a database shall be in < 1 minute (**TBC**).

3.3 Off-Line Delivery Requirements

N/A

[1] This requirement previously read "Access to the telemetry source packets and data frames shall be read only".

4. INTERFACE CHARACTERISTICS

4.1 Interface Location and Medium

No assumptions are made.

4.2 Hardware Characteristics and Limitations

Performance is bound by:

- The database server. This is an optimisation issue as well as a hardware one.
- Network connections.

4.3 Data Source, Destination and Transfer Mechanism

Potential data sources are discussed in section **1.2**. The destination is any client.

The transfer mechanism can be on a local machine or via TCP/IP (with or without an HTTP wrapper). Data that is transferred across a network is subject to Java serialization.

4.4 Node and Device Addressing

- Connection to a socket-based data server requires that the name of the host (or IP address) and the listening port be specified.
- Connection to an HTTP-based data server requires a URL to be specified in the normal way. The port will default to 80 (*ie* the default HTTP port) unless otherwise specified.

4.5 Relationship with other interfaces

4.5.1 Router

The interface to packets and data frames in NRT is via the router (see **[AD2]**). Data frames will have to be converted "on-the fly". The classes RouterConnection and RouterProductStream support this interface.

4.5.2 RTA

TBW

4.5.3 Database

The requirements for on-line delivery performance imply requirements on the database schema and optimisation. *This in turn impacts the CCM.* The implications of this are **TBS**.

5. ACCESS

5.1 Programs generating or using the Interface Data

Unrestricted. Any application conforming to the interface may access the data.

5.2 Failure Protection, Detection and Recovery Procedures

- Any failure should cause the connection to be broken and an exception to be thrown.
- A failure should cause any database transaction to end immediately, and rollback to be applied (if applicable). As there is a requirement that the database should not be modified, this should not be an issue.

5.3 File Naming Conventions

See section 7.1 for Java property file conventions.

5.4 Storage and File Deletion Requirements

N/A

5.5 Security Requirements

- Servers must be kept up-to-date with the latest security patches. This is especially (but not exclusively) true for servers not protected by a firewall.
- It shall not be possible for an unauthorised user to modify the database in any way.
- Server-side code should be reviewed with regard to any compromises to the integrity of either the operating system or the database.

5.6 Data Integrity Checks

None. The interface provides no direct means of modifying data. This responsibility lies elsewhere.

5.7 Backup Requirements

None. Backup of both served data and software are the responsibility of individual ICCs.

5.8 Input / Output Protocols, Calling Sequence

See section 8.

5.9 Synchronisation Requirements

- Server-side code must be implemented in such a way that multiple threads can run concurrently.
- A locking mechanism must be used to prevent data being extracted from the database whilst it is in the process of being written or modified.

5.10 Error Handling

All unhandled errors will cause an Exception to be thrown.

6. DETAILED INTERFACE SPECIFICATIONS

6.1 Data Structure

See the (**TBW**) definitions of telemetry source packets and data frames. The internal structure of the HCSS access objects used to encapsulate requests is implementation dependent. The interface to these objects is described below.

6.2 Generation Method

Generation of packets and data frames is out of the scope of this document, which is concerned with an interface to the data. HCSS access objects are constructed in client (Java) code.

6.3 Data passed across the interface - direction of transfer

Telemetry source packets	server - client
Data frames	server - client
HCSS access objects	client - server

Note that the overall architecture may be multi-tier *ie* more than one server layer.

6.4 Size and Frequency of Transfers

Data requests and transfers are expected to be infrequent by database standards.

The size of a request will be <1 Kbyte.

The size of a response can be anything from a single packet to all the data frames in the database (**TBS** ~1Tbyte?). It is **TBD** whether and how a maximum data size is enforced. Currently a client cannot find out how large a request will be in advance as data volumes are not stored in the CCM.

7. DATA DEFINITIONS

7.1 Properties file

This file is used to specify the values of default parameters. Currently a single file is used for both client and server parameters (though there can be a different instance of the file on the server than on the client). It is also currently shared with the **store** package. These files may be separated at a later date. The JVM first checks if there is a System property set with the name of "props". If there is, then the value of that property is used as the file name, otherwise it looks for a file with the name of "props".

Most of the values of these properties are also available through methods of the HCSSConnection class (see the API).

Note that only parameters that are actually used need to be specified, so a server does not need to worry about setting client parameters.

7.1.1 File Characteristics

Java properties file.

7.1.2 Header Records

N/A

7.1.3 Data Records

7.1.3.1 Database (store package) parameters

dbfactory	Full name of the factory class for the "store" implementation
dbname	Name of the database to use

7.1.3.2 Client-side parameters

Client.CONNECTION_TYPE	Full name of the default class to use for connections.
Client.NETWORK_TYPE	Default type of network connection, `socket` or `http`
Client.SOCKET_HOST	Name of host for socket connection
Client.SOCKET_PORT	Port number of socket connection
Client.URL	URL for http connections

7.1.3.3 Server-side parameters

Server.SOCKET_PORT	Port number to listen on
Server.QUERY_FACTORY	Full name of class to manufacture queries

7.1.3.4 Router parameters

Router.HOST	Name of host router is running on
Router.PORT	Port number for router connection

7.1.4 File example

```
# Properties file for TM/Data frame clients and servers
#
# Author S.Guest   RAL/SPIRE

# database access stuff
dbfactory = nl.esa.herschel.store.vers.StoreFactoryImpl
dbname = herschel

# client-side parameters
Client.CONNECTION_TYPE =
nl.esa.herschel.access.db.LocalConnection
Client.NETWORK_TYPE = socket
Client.SOCKET_HOST = oakwell.bnsc.rl.ac.uk
Client.SOCKET_PORT = 8050
Client.URL = http://scott1.bnsc.rl.ac.uk/servlets/

# server-side parameters
Server.QUERY_FACTORY =
nl.esa.herschel.access.db.VersantQueryFactory
Server.SOCKET_PORT = 8050

# router parameters
Router.HOST = oakwell.bnsc.rl.ac.uk
Router.PORT = 9876
```

7.2 Data Definition

This section could define how local storage on a client machine is handled. This is **TBD**.

7.2.1 File Characteristics

7.2.2 Header Records

7.2.3 Data Records

7.2.4 File example

8. APPLICATION PROGRAMMING INTERFACE

This section replaces the software data definition sections, which are inappropriate to describe an object-oriented specification. See **[AD1]** for a design description.

Note: This is a specification. The ultimate usage documentation will be the javadoc output.

The sequence of (client) use of this interface is as follows.
1. Create an HCSSAccess object and tailor it for the request (but see restrictions on this class).
2. Create a ProductReader, passing the HCSSAccess object as an argument.
3. Call the openStream() method of ProductReader to get a ProductStream object.
4. Iterate through the objects in the ProductStream, casting them to TMSourcePacket or DataFrame as appropriate.
5. Optionally, close the stream.

8.1 HCSSAccess

8.1.1 Purpose

This class encapsulates a request made by the client.

8.1.2 Constructors

```
public HCSSAccess (String classname);
public HCSSAccess (String classname, starttime, endtime);
public HCSSAccess (String classname, int obsid);
public HCSSAccess (String classname, int obsid, int bbid);
```

8.1.3 Methods

```
public int getObsid();
public int getBBid();
public getStartTime();
public getEndTime();
public int getTimeout();
public void setObsid (int obsid);
public void setBBid (int bbid);
public void setStartTime (starttime);
public void setEndTime (endtime);
public void setTimeout (int milliseconds);
public String getClassName();
public void setClassName (String classname);
```

8.1.4 Restrictions and notes

- How should time be specified?
- In a network connection objects of this class will be serialized and written to the server via an output stream.
- Requests will be checked for internal consistency when a ProductReader object is created.

8.2 PacketAccess

8.2.1 Purpose

This is a subclass of HCSSAccess tailored for packets. Its main purpose is that of convenience.

8.2.2 Constructors

```
public PacketAccess ();
public PacketAccess (starttime, endtime);
public PacketAccess (int obsid);
public PacketAccess (int obsid, int bbid);
```

8.2.3 Methods

```
public int getApid();
public void setApid (int apid);
public int getType();
public void setType (int type);
public int getSubType();
public void setSubType (int subtype);
public int getScosid();
public void setScosid (int scosid);
```

8.2.4 Restrictions and notes

8.3 ProductReader

8.3.1 Purpose

This interface defines the client-side protocol for opening a connection. In principle this could be to any product (hence the name), but only packets and data frames have been considered up to now.

8.3.2 Methods

```
public ProductStream openStream() throws IOException;
public void close() throws IOException;
public boolean isOpen() throws IOException;
```

8.4 HCSSConnection

8.4.1 Purpose

This is a factory class that manufactures ProductReaders. It also provides a number of utility methods.

8.4.2 Methods

```
public static ProductReader getConnection(HCSSAccess access);
```

```
public static ProductReader getConnection(String classname,
HCSSAccess access);
public static String getDefaultConnectionType() {
public static String getDefaultNetworkType() {
public static String getDefaultSocketHost() {
public static int getDefaultSocketPort() {
public static URL getDefaultURL() {
```

8.4.3 Restrictions and notes

- This should be the normal interface for making connections. It is also possible to create instances with "new". Some implementations are described below.
- There is no public constructor.
- Default parameters are read from a properties file.

8.5 LocalConnection

8.5.1 Purpose

This class implements ProductReader for a local database connection.

8.5.2 Constructors

```
public LocalConnection (HCSSAccess access);
```

8.5.3 Methods

See ProductReader.

8.5.4 Restrictions and notes

Other local implementations (eg FileConnection) will have identical or similar interfaces. See the javadoc for full details.

8.6 NetworkConnection

8.6.1 Purpose

This class implements ProductReader for a connection over a network.

8.6.2 Constructors

```
public NetworkConnection (HCSSAccess access);
public NetworkConnection (URL url, HCSSAccess access);
public NetworkConnection (String host, int port);
public NetworkConnection (String host, int port, HCSSAccess
access);
```

8.6.3 Methods

See ProductReader.

8.6.4 Restrictions and notes

8.7 RouterConnection

8.7.1 Purpose

This class implements ProductReader for a connection to the router. This class is intended for use in NRT applications. There are enough differences in implementation from NetworkConnection to justify its own class.

8.7.2 Constructors

```
public RouterConnection (HCSSAccess access);
public RouterConnection (String host, int port, HCSSAccess
access);
```

8.7.3 Methods

See ProductReader.

8.7.4 Restrictions and notes

- The router communicates with instances of this class from a different thread.

8.8 ProductStream

8.8.1 Purpose

This interface specifies the access to products through a connection. It is an extension of java.util.Iterator.

8.8.2 Methods

```
public boolean hasNext();
public Object next();
```

8.8.3 Restrictions and notes

- The Iterator interface also specifies a `remove()` method. This will not and must not be implemented.
- There is a request from PACS for a `previous()` method, as well as some sort of "marking" mechanism. This would require implementing caching layers. It is planned to implement this later. This functionality is not included in the initial prototype, as it requires some additional design to incorporate a caching strategy concept. It would also require an extension of this interface.
- This interface currently has three implementing classes, which correspond to the three connection implementations. These are DirectProductStream (local), InputProductStream (network) and RouterProductStream. There is no need to create them directly as they are returned through the ProductReader interface.

8.9 Example code segment

```
// create an access object for (any) SPIRE data frames
// with obervation id 1234
HCSSAccess access = new HCSSAccess ("SpireDataFrame", 1234);

// set up a connection using the system defaults
ProductReader connect = HCSSConnection.getConnection (access);

// open a stream to the connection
ProductStream stream = connect.openStream();

// iterate over the result
while (stream.hasNext()) {
    SpireDataFrame frame = (SpireDataFrame) stream.next();
    System.out.print (frame.getStatus()+"..");
}
```