Draft Specification Concerning Space Data System Standards

SPACE LINK EXTENSION—APPLICATION PROGRAM INTERFACE FOR THE FORWARD CLTU SERVICE

DRAFT RECOMMENDED PRACTICE

CCSDS 916.1-M-0

September 2005
AUTHORITY

Issue: Draft Magenta Book, Issue 0
Date: September 2005
Location: N/A

(WHEN THIS RECOMMENDED PRACTICE IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF AUTHORITY:)

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS Recommendations is detailed in the Procedures Manual for the Consultative Committee for Space Data Systems, and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the address below.

This Recommended Practice is published and maintained by:

CCSDS Secretariat
Program Integration Division (Code MT)
National Aeronautics and Space Administration
Washington, DC 20546, USA
STATEMENT OF INTENT

(WHEN THIS RECOMMENDED PRACTICE IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF INTENT:)

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of member space Agencies. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed Recommendations and are not considered binding on any Agency.

This Recommended Practice is issued by, and represents the consensus of, the CCSDS Plenary body. Agency endorsement of this Recommended Practice is entirely voluntary. Endorsement, however, indicates the following understandings:

- Whenever an Agency establishes a CCSDS-related standard, this standard will be in accord with the relevant Recommendation. Establishing such a standard does not preclude other provisions which an Agency may develop.

- Whenever an Agency establishes a CCSDS-related standard, the Agency will provide other CCSDS member Agencies with the following information:
  -- The standard itself.
  -- The anticipated date of initial operational capability.
  -- The anticipated duration of operational service.

- Specific service arrangements shall be made via memoranda of agreement. Neither this Recommended Practice nor any ensuing standard is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this Recommended Practice will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or, (3) be retired or canceled.

In those instances when a new version of a Recommendation is issued, existing CCSDS-related Agency standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each Agency to determine when such standards or implementations are to be modified. Each Agency is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommended Practice.
FOREWORD

(WHEN THIS RECOMMENDED PRACTICE IS FINALIZED, IT WILL CONTAIN THE FOLLOWING FOREWORD:)

This document is a technical Recommended Practice for use in developing ground systems for space missions and has been prepared by the Consultative Committee for Space Data Systems (CCSDS). The Application Program Interface described herein is intended for missions that are cross-supported between Agencies of the CCSDS.

This Recommended Practice specifies service type specific extensions of the Space Link Extension Application Program Interface for Transfer Services specified by CCSDS (reference [6]). It allows implementing organizations within each Agency to proceed with the development of compatible, derived Standards for the ground systems that are within their cognizance. Derived Agency Standards may implement only a subset of the optional features allowed by the Recommended Practice and may incorporate features not addressed by the Recommended Practice.

Through the process of normal evolution, it is expected that expansion, deletion or modification to this document may occur. This Recommended Practice is therefore subject to CCSDS document management and change control procedures, as defined in the Procedures Manual for the Consultative Committee for Space Data Systems. Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be addressed to the CCSDS Secretariat at the address indicated on page i.
At time of publication, the active Member and Observer Agencies of the CCSDS were:

**Member Agencies**

- Agenzia Spaziale Italiana (ASI)/Italy.
- British National Space Centre (BNSC)/United Kingdom.
- Canadian Space Agency (CSA)/Canada.
- Centre National d’Etudes Spatiales (CNES)/France.
- Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- National Aeronautics and Space Administration (NASA)/USA.
- National Space Development Agency of Japan (NASDA)/Japan.
- Russian Space Agency (RSA)/Russian Federation.

**Observer Agencies**

- Austrian Space Agency (ASA)/Austria.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- Centro Tecnico Aeroespacial (CTA)/Brazil.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Communications Research Laboratory (CRL)/Japan.
- Danish Space Research Institute (DSRI)/Denmark.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Industry Canada/Communications Research Centre (CRC)/Canada.
- Institute of Space and Astronautical Science (ISAS)/Japan.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- Korea Aerospace Research Institute (KARI)/Korea
- Ministry of Communications (MOC)/Israel.
- National Oceanic & Atmospheric Administration (NOAA)/USA.
- National Space Program Office (NSPO)/Taipei.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.
PREFACE

This document is a draft CCSDS Recommended Practice. Its draft status indicates that the CCSDS believes the document to be technically mature and has released it for formal review by appropriate technical organizations. As such, its technical contents are not stable, and several iterations of it may occur in response to comments received during the review process.

Implementers are cautioned not to fabricate any final equipment in accordance with this document’s technical content.
## DOCUMENT CONTROL

<table>
<thead>
<tr>
<th>Document</th>
<th>Title</th>
<th>Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCSDS 916.1-M-0</td>
<td>Space Link Extension—Application Program Interface for the Forward CLTU Service, Draft Recommended Practice, Issue 0</td>
<td>September 2005</td>
<td>Pre-approval draft</td>
</tr>
</tbody>
</table>
CONTENTS

Section                          Page
1 INTRODUCTION................................. 1-1
  1.1 PURPOSE.................................... 1-1
  1.2 SCOPE...................................... 1-1
  1.3 APPLICABILITY............................ 1-1
  1.4 RATIONALE................................. 1-2
  1.5 DOCUMENT STRUCTURE.................... 1-2
  1.6 DEFINITIONS, NOMENCLATURE, AND CONVENTIONS.............. 1-5
  1.7 REFERENCES................................ 1-8
2 OVERVIEW....................................... 2-1
  2.1 INTRODUCTION............................. 2-1
  2.2 PACKAGE CLTU SERVICE INSTANCES......... 2-1
  2.3 PACKAGE CLTU OPERATIONS................ 2-12
3 CLTU SPECIFIC SPECIFICATIONS FOR API COMPONENTS............ 3-1
  3.1 API SERVICE ELEMENT..................... 3-1
  3.2 SLE OPERATIONS........................... 3-14
  3.3 SLE APPLICATION........................... 3-14

ANNEX A CLTU SPECIFIC INTERFACES.......................... A-1
ANNEX B ACRONYMS......................................... B-1
ANNEX C INFORMATIVE REFERENCES.......................... C-1
CONTENTS (continued)

Figure

1-1 SLE Services and SLE API Documentation ................................................................. 1-4
2-1 CLTU Service Instances ............................................................................................... 2-3
2-2 CLTU Service Instances ............................................................................................. 2-13

Table

2-1 Production Events Reported via the Interface ICLTU_SIUpdate ................................. 2-5
2-2 CLTU Configuration Parameters Handled by the Service Element ......................... 2-7
2-3 CLTU Status Parameters Handled by the Service Element ....................................... 2-8
2-4 CLTU Production Status ............................................................................................. 2-9
2-5 Mapping of CLTU Operations to Operation Object Interfaces ................................. 2-13
1 INTRODUCTION

1.1 PURPOSE

The Recommended Practice ‘Application Program Interface for Transfer Services – Core Specification’ (reference [6]) specifies a C++ API for CCSDS Space Link Extension Transfer Services. The API is intended for use by application programs implementing SLE transfer services.

Reference [6] defines the architecture of the API and the functionality on a generic level, which is independent of specific SLE services and communication technologies. It is thus necessary to add service-type specific specifications in supplemental Recommended Practices. The purpose of this document is to specify extensions to the API needed for support of the Command Link Transmission Unit (CLTU) service defined in reference [5].

1.2 SCOPE

This Recommended Practice defines extensions to the SLE API in terms of:

a) the CLTU specific functionality provided by API components;

b) the CLTU specific interfaces provided by API components; and

c) the externally visible behavior associated with the CLTU interfaces exported by the components.

It does not specify

a) individual implementations or products;

b) the internal design of the components; and

c) the technology used for communications.

This Recommended Practice only defines interfaces and behavior that must be provided by implementations supporting the forward CLTU service in addition to the specification in reference [6].

1.3 APPLICABILITY

The CLTU Application Program Interface specified in this document supports two versions of the CLTU service, namely:

a) version 1 as specified by reference [4]; and

b) version 2 as specified by reference [5].

Support for version 1 of these services is included for backward compatibility purposes for a limited time and may not be continued in future versions of this specification. Support for
version 1 of the CLTU service implies that SLE API implementations of this specification are able to interoperate with peer SLE systems that comply with the specification of the Transport Mapping Layer (TML) in ‘Specification of a SLE API Proxy for TCP/IP and ASN.1’, ESOC, SLES-SW-API-0002-TOS-GCI, Issue 1.1, February 2001.

Version dependent provisions within this Recommended Practice are marked as follows:

a) [V1:] for provisions specific to version 1; and

b) [V2:] for provisions specific to version 2.

1.4 RATIONALE

This Recommended Practice specifies the mapping of the forward CLTU service specification to specific functions and parameters of the SLE API. It also specifies the distribution of responsibility for specific functions between SLE API software and application software.

The goal of this Recommended Practice is to create a standard for interoperability between:

a) application software using the SLE API and SLE API software implementing the SLE API; and

b) SLE user and SLE provider applications communicating with each other using the SLE API on both.

This interoperability standard also allows exchangeability of different products implementing the SLE API, as long as they adhere to the interface specification of this Recommended Practice.

1.5 DOCUMENT STRUCTURE

1.5.1 ORGANIZATION

This document is organized as follows:

- Section 1 provides purpose and scope of this specification, identifies conventions, and lists definitions and references used throughout the document;

- Section 2 provides an overview of the CLTU service and describes the API model extension including support for the CLTU service;

- Section 3 contains detailed specifications for the API components and for applications using the API;

- Annex A provides a formal specification of the API interfaces and data types specific to the CLTU service;
1.5.2 SLE SERVICE DOCUMENTATION TREE

The SLE suite of Recommendations is based on the cross support model defined in the SLE Reference Model (reference [3]). The services defined by the reference model constitute one of the three types of Cross Support Services:

a) Part 1: SLE Services;

b) Part 2: Ground Domain Services; and

c) Part 3: Ground Communications Services.

The SLE services are further divided into SLE service management and SLE transfer services.

The basic organization of the SLE services and SLE documentation is shown in figure 1-1. The various documents are described in the following paragraphs.
Figure 1-1: SLE Services and SLE API Documentation

a) Cross Support Reference Model—Part 1: Space Link Extension Services; a Recommendation that defines the framework and terminology for the specification of SLE services.

b) Cross Support Concept—Part 1: Space Link Extension Services; a Report introducing the concepts of cross support and the SLE services.

c) Space Link Extension Services—Executive Summary; an Administrative Report providing an overview of Space Link Extension (SLE) Services. It is designed to assist readers with their review of existing and future SLE documentation.

d) Forward SLE Service Specifications; a set of Recommendations that provide specifications of all forward link SLE services.

e) Return SLE Service Specifications; a set of Recommendations that provide specifications of all return link SLE services.
f) **Internet Protocol for Transfer Services**; a Recommendation providing the specification of the wire protocol used for SLE transfer services.

g) **SLE Service Management Specifications**; a set of Recommendations that establish the basis of SLE service management.

h) **Application Program Interface for Transfer Services—Core Specification**; a Recommended Practice document specifying the generic part of the API for SLE transfer services.

i) **Application Program Interface for Transfer Services—Summary of Concept and Rationale**; a Report describing the concept and rationale for specification and implementation of a Application Program Interface for SLE Transfer Services.

j) **Application Program Interface for Return Services**; a set of Recommended Practice documents specifying the service-type specific extensions of the API for return link SLE services.

k) **Application Program Interface for Forward Services**; a set of Recommended Practice documents specifying the service-type specific extensions of the API for forward link SLE services.

l) **Application Program Interface for Transfer Services—Application Programmer's Guide**; a Report containing guidance material and software source code examples for software developers using the API.

1.6 DEFINITIONS, NOMENCLATURE, AND CONVENTIONS

1.6.1 DEFINITIONS

1.6.1.1 Definitions from Telecommand Channel Service

This Recommended Practice makes use of the following terms defined in reference [1]:

- a) Command Link Transmission Unit (CLTU);

1.6.1.2 Definitions from Telecommand Data Routing Service

This Recommended Practice makes use of the following terms defined in reference [2]:

- Command Link Control Word (CLCW).

1.6.1.3 Definitions from SLE Reference Model

This Recommended Practice makes use of the following terms defined in reference [3]:
a) Forward CLTU service;
b) operation;
c) service provider (provider);
d) service user (user);
e) SLE transfer service instance;
f) SLE transfer service production;
g) SLE transfer service provision;
h) space link data unit (SL-DU).

1.6.1.4 Definitions from CLTU Service

This Recommended Practice makes use of the following terms defined in reference [5]:

a) association;
b) communications service;
c) confirmed operation;
d) invocation;
e) parameter;
f) performance;
g) port identifier;
h) return;
i) service instance provision period;
j) unconfirmed operation.

1.6.1.5 Definitions from ASN.1 Specification

This Recommended Practice makes use of the following terms defined in reference [8]:

a) Object Identifier;
b) Octet String.

1.6.1.6 Definitions from UML Specification

This Recommended Practice makes use of the following terms defined in reference [C7]:
1.6.1.7 Definitions from API Core Specification

This Recommended Practice makes use of the following terms defined in reference [6]:

a) Application Program Interface;

b) Component.

1.6.2 NOMENCLATURE

The following conventions apply throughout this Recommended Practice:

a) the words ‘shall’ and ‘must’ imply a binding and verifiable specification;

b) the word ‘should’ implies an optional, but desirable, specification;

c) the word ‘may’ implies an optional specification;

d) the words ‘is’, ‘are’, and ‘will’ imply statements of fact.

1.6.3 CONVENTIONS

This document applies the conventions defined in reference [6].

The model extensions in section 2 are presented using the Unified Modelling Language (UML) and applying the conventions defined in reference [6].

The CLTU specific specifications for API components in section 3 are presented using the conventions specified in reference [6].

The CLTU specific interfaces in annex A are specified using the conventions defined in reference [6].
1.7 REFERENCES

The following documents contain provisions, which through reference in this text, constitute provisions of this document. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this document are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS Reports and Recommendations.

NOTE – A list of informative references is provided in annex C.


2 OVERVIEW

2.1 INTRODUCTION

This section describes the extension of the SLE API model in reference [6] for support of the CLTU service. Extensions are needed for the API components API Service Element and SLE Operations.

In addition to the extensions defined in this chapter, the component API Proxy must support encoding and decoding of CLTU specific protocol data units.

2.2 PACKAGE CLTU SERVICE INSTANCES

2.2.1 OVERVIEW

The CLTU extensions to the component API Service Element are defined by the package CLTU Service Instances. Figure 2-1 provides an overview of this package. The diagram includes classes from the package API Service Element specified in reference [6], which provide applicable specifications for the CLTU service.

The package adds two service instance classes:

a) CLTU SI User, supporting the service user role; and

b) CLTU SI Provider, supporting service provider role.

These classes correspond to the placeholder classes I<SRV>_SI User and I<SRV>_SI Provider defined in reference [6].

Both classes are able to handle the specific CLTU operations.

For the class CLTU SI User, this is the only extension of the base class SI User.

The class CLTU SI Provider adds two new interfaces:

a) ICLTU_SIAdmin by which the application can set CLTU specific configuration parameters; and

b) ICLTU_SIUpdate by which the application must update dynamic status information, required for generation of status reports.

These interfaces correspond to the placeholder interfaces I<SRV>_SIAdmin and I<SRV>_SIUpdate defined in reference [6].

CLTU specific configuration parameters are defined by the internal class CLTU Configuration. The class CLTU Status Information defines dynamic status parameters maintained by the service instance. In addition, the service instance maintains a set of
parameters for the last CLTU processed and for the last CLTU that was successfully radiated. These parameters are defined by the classes CLTU Last Processed and CLTU Last OK.

Although the CLTU service allows only a single service instance to be bound to a service provider at any point of time, the service element does not constrain the number of CLTU service instances on the user side or the provider side. More than one service instance might be needed for back-up purposes. In addition, this specification does not exclude that a single service element be used to serve several CLTU production engines or to connect to several providers. Therefore, the service element shall not enforce that only a single CLTU service instance is bound.

All specifications provided in this section refer to a single service instance. If more than one service instance is used, each service instance must be configured and updated independently.
2.2.2 COMPONENT CLASS CLTU SI USER

The class defines a CLTU service instance supporting the service user role. It ensures that SLE PDUs passed by the application and by the association are supported by the CLTU service and handles the CLTU operation objects defined in 2.3. It does not add further features to those provided by the base class SI User.
2.2.3 COMPONENT CLASS CLTU SI PROVIDER

The class defines a CLTU service instance supporting the service provider role. It exports the interfaces ICLTU_SIAdmin for configuration of the service instance after creation and ICLTU_SIUpdate for update of dynamic status parameters during operation.

2.2.3.1 Responsibilities

2.2.3.1.1 Service Specific Configuration

The service instance implements the interface ICLTU_SIAdmin to set the CLTU specific configuration parameters defined by the class CLTU Configuration. The methods of this interface must be called after creation of the service instance. When all configuration parameters (including those set via the interface ISLE_SIAdmin) have been set, the method ISLE_SIAdmin::ConfigCompleted() must be called. This method verifies that all configuration parameters values are defined and are in the range defined in reference [5].

In addition, the interface ICLTU_SIAdmin provides read access to the configuration parameters.

2.2.3.1.2 Update of Dynamic Status Parameters

The class implements the interface ICLTU_SIUpdate to inform the service instance of specific events in the CLTU production process. The methods of this interface update status parameters defined by the classes CLTU Status Information, CLTU Last Processed, and CLTU Last OK. The events reported via ICLTU_SIUpdate and the parameters updated via this interface are listed in table 2-1.

In order to ensure that the status information is always up to date the events listed in table 2-1 must be reported to the service instance during its complete lifetime, independent of the state of the service instance.

In addition, the class derives some of the parameters in CLTU Status Information from CLTU PDUs exchanged between the service user and the service provider. The methods used to update each of the parameters are defined in 2.2.5.

The interface ICLTU_SIUpdate provides read access to all status parameters.
Table 2-1: Production Events Reported via the Interface ICLTU_SIUpdate

NOTE – The notification type actually transmitted depends on the method arguments and partially or the value of the production status.

<table>
<thead>
<tr>
<th>Event</th>
<th>Method</th>
<th>Arguments</th>
<th>Status parameters updated</th>
<th>Notification sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radiation of a CLTU started.</td>
<td>CltuStarted</td>
<td>CLTU identification last processed</td>
<td>CLTU identification last processed radiation start time</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>radiation start time</td>
<td>radiation start time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>available buffer size</td>
<td>available buffer size</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLTU status</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>number of CLTUs processed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>available buffer size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLTU identification last OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>radiation stop time</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLTU status</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>number of CLTUs radiated</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>available buffer size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radiation of a CLTU completed.</td>
<td>CltuRadiated</td>
<td>radiation stop time</td>
<td>CLTU identification last OK</td>
<td>CLTU radiated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLTU identification last OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>radiation stop time</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLTU status</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>number of CLTUs radiated</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>available buffer size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radiation of a CLTU could not be started because the latest</td>
<td>CltuNotStarted</td>
<td>CLTU identification failure reason</td>
<td>CLTU identification last processed radiation start time</td>
<td>SLDU expired</td>
</tr>
<tr>
<td>radiation time expired or the production status was interrupted.</td>
<td></td>
<td>CLTU status</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>available buffer size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>number of CLTUs processed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>available buffer size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BufferEmpty</td>
<td>available buffer size</td>
<td>buffer empty</td>
<td></td>
</tr>
<tr>
<td>The CLTU buffer is empty.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The production status changed with or without affecting a CLTU being</td>
<td>ProductionStatusChange</td>
<td>production status</td>
<td>production operational</td>
<td></td>
</tr>
<tr>
<td>being radiated.</td>
<td></td>
<td>available buffer size</td>
<td>production interrupted</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>available buffer size</td>
<td>production interrupted</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>production halted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing of a thrown event completed</td>
<td>EventProcCompleted</td>
<td>event id</td>
<td>action list completed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>event processing result</td>
<td>action list not completed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>action list completed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>result</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The uplink status changed</td>
<td>Set_UplinkStatus</td>
<td>uplink status</td>
<td>uplink status</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.2.3.1.3 Generation of Notifications

If events reported via the interface `ICLTU_SIUpdate` require that a CLTU–ASYNC–NOTIFY invocation be sent to the service user, the class generates and transmits these invocations if that is requested by the application and if the state of the service instance is ‘active’ or ‘ready’. The notifications that are generated and transmitted by the class are listed in table 2-1.

The application can opt not to use this feature, but to generate the notification itself and transmit it using the interface `ISLE_ServiceInitiate`. It is noted that reference [5] defines additional notifications that must always be generated and transmitted by the application.

The SLE API supports different modes for generation of notifications. In ‘deferred’ notification mode, if no CLTU is affected and the production status changes to ‘interrupted’; the notification is deferred until the attempt is made to radiate the next CLTU. In ‘immediate’ notification mode, the ‘production interrupted’ notification is generated immediately.

2.2.3.1.4 Handling of the CLTU–GET-PARAMETER Operation

The class responds autonomously to CLTU–GET–PARAMETER invocations. It generates the appropriate CLTU–GET–PARAMETER return using the parameters maintained by the classes CLTU Configuration and CLTU Status Information.

2.2.3.1.5 Status Reporting

The class generates CLTU–STATUS–REPORT invocations when required using the parameters maintained by the classes CLTU Status Information and CLTU Information.

2.2.3.1.6 Processing of CLTU Protocol Data Units

The class ensures that SLE PDUs passed by the application and by the association are supported by the CLTU service and handles the CLTU operation objects defined in 2.3.

2.2.3.1.7 Processing of CLTU–TRANSFER–DATA Invocations

For incoming CLTU–TRANSFER–DATA invocations the class performs the following checks in addition to those defined in [6]:

a) if the ‘earliest radiation time’ and the ‘latest radiation time’ are both specified, the ‘earliest radiation time’ must not be later than the ‘latest radiation time’;

b) the size of the CLTU contained in the PDU must not be larger than the value of the configuration parameter ‘maximum-sldu-length’ allows.

In contrast to handling of other confirmed operations, the service instance is allowed to pass the operation object to the application after setting the correct diagnostic if these checks fail.
The application is expected to insert the next expected CLTU identification and the available buffer size into the operation object and pass it back to service instance via the interface ISLE_ServiceInitiate. The reasons for this specification are explained in 2.2.8.3.

### 2.2.3.1.8 Processing of CLTU–THROW-EVENT invocations

In contrast to handling of other confirmed operations, the service instance is allowed to pass the operation object to the application after setting the correct diagnostic if checks performed by the service element fail. The application is expected to insert the next expected event invocation identifier into the operation object and pass it back to service instance via the interface ISLE_ServiceInitiate. The reasons for this specification are explained in 2.2.8.3.

### 2.2.4 INTERNAL CLASS CLTU CONFIGURATION

The class defines the configuration parameters that can be set via the interface ICLTU_SIAdmin. These parameters are defined by reference [5]. Table 2-2 describes how the service instance uses these parameters. The column labeled ‘Upd’ indicates whether an update of these parameters is allowed after the initial configuration has been completed. It is noted that an update might be inhibited by service management also when an update is possible according to the table.

#### Table 2-2: CLTU Configuration Parameters Handled by the Service Element

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Used for</th>
<th>Upd</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit-lock-required</td>
<td>CLTU–GET–PARAMETER</td>
<td>Y</td>
</tr>
<tr>
<td>maximum-cltu-length</td>
<td>CLTU–GET–PARAMETER</td>
<td>Y</td>
</tr>
<tr>
<td>modulation-frequency</td>
<td>CLTU–GET–PARAMETER</td>
<td>Y</td>
</tr>
<tr>
<td>plop-in-effect</td>
<td>CLTU–GET–PARAMETER</td>
<td>Y</td>
</tr>
<tr>
<td>rf-available-required</td>
<td>CLTU–GET–PARAMETER</td>
<td>Y</td>
</tr>
<tr>
<td>subcarrier-to-bitrate-ratio</td>
<td>CLTU–GET–PARAMETER</td>
<td>Y</td>
</tr>
<tr>
<td>maximum-cltu-buffer-size</td>
<td>value of the status parameter CLTU buffer available after configuration, CLTU-STOP, CLTU-PEER-ABORT, and protocol abort</td>
<td>N</td>
</tr>
<tr>
<td>modulation-index</td>
<td>the value of the modulation index in milli-radians (for version 1, the amount of carrier suppression in 1/100 dB)</td>
<td>N</td>
</tr>
<tr>
<td>notification-mode</td>
<td>value of the notification mode, either ‘immediate’ or ‘deferred’</td>
<td>N</td>
</tr>
</tbody>
</table>
## 2.2.5 INTERNAL CLASS CLTU STATUS INFORMATION

The class defines global status parameters handled by the service instance. The parameters are defined by reference [5]. Table 2-3 describes how the service element updates each of the parameters and how it uses the parameters.

### Table 2-3: CLTU Status Parameters Handled by the Service Element

| Parameter                  | Update                                                                 | Used for                                       |
|----------------------------|------------------------------------------------------------------------|                                               |
| production-status          | - set by methods of ICLTU_SIUpdate                                      | status reports, notifications                  |
| cltu-buffer-available      | - set to maximum at configuration time                                 | status reports, notifications                  |
|                            | - set by methods of ICLTU_SIUpdate                                      |                                               |
|                            | - extracted from CLTU-TRANSFER-DATA returns                            |                                               |
|                            | - reset to maximum following CLTU–STOP, CLTU-PEER–ABORT and protocol abort |                                               |
| number-of-cltus-received   | - set to zero at configuration time                                     | status reports                                 |
|                            | - incremented for every CLTU-TRANSFER-DATA return with a positive result|                                               |
| number-of-cltus-processed  | - set to zero at configuration time                                     | status reports                                 |
|                            | - incremented with every call to CltuStarted() and CltuNotStarted()    |                                               |
| number-of-cltus-radiated   | - set to zero at configuration time                                     | status reports                                 |
|                            | - incremented with every call to CltuRadiated()                        |                                               |
| expected-cltu-identification | - set to zero at configuration time                                    | CLTU–GET–PARAMETER                             |
|                            | - [V1:] extracted from CLTU-START invocations if the application transmits a return with a positive result and the parameter first-cltu-identification is used. |                                               |
|                            | - [V2:] extracted from CLTU-START invocations if the application transmits a return with a positive result. |                                               |
|                            | - extracted from CLTU-TRANSFER-DATA returns                            |                                               |
| expected-event-invocation-identifier | - set to zero at configuration time                                    | CLTU–GET–PARAMETER                             |
|                            | - extracted from CLTU-THROW-EVENT returns                              |                                               |
| uplink-status              | - set by methods of ICLTU_SIUpdate                                      | status reports, notifications                  |
2.2.6 INTERNAL CLASS CLTU LAST PROCESSED

The class defines the parameters maintained by the service instance for the last CLTU for which radiation started or radiation was attempted. These parameters are defined in reference [5].

All parameters are set via methods in the interface `ICLTU_SIUpdate` (see table 2-1) and are used in status reports and notifications.

2.2.7 INTERNAL CLASS CLTU LAST OK

The class defines the parameters maintained by the service instance for the last CLTU for which radiation was completed. These parameters are defined in reference [5].

All parameters are set via methods in the interface `ICLTU_SIUpdate` (see table 2-1) and are used in status reports and notifications.

2.2.8 FEATURES NOT HANDLED BY THE PROVIDER SIDE SERVICE INSTANCE

2.2.8.1 Introduction

As a general approach, this specification only states what the API is required to do. Features not identified in this specification cannot be expected from a conforming implementation. This section deviates from this approach by discussing features not provided by the API, with the intention to clarify the borderline between the application and the API Service Element.

In addition, this section outlines the rationale for the distribution of responsibilities between the application and the API Service Element in this specification.

2.2.8.2 Production Status

Reference [5] defines a parameter ‘production status’, which reflects the state of the CLTU production engine. The value of the production status is not only included in status reports and notifications, but also determines whether invocations of the operations CLTU–BIND and CLTU–START can be accepted or not. The production status also has an impact on processing of CLTU–TRANSFER–DATA operations, which is discussed in 2.2.8.4.

Table 2-4 lists the possible values of the production status and the required processing of BIND and START invocations.

<table>
<thead>
<tr>
<th>Production Status</th>
<th>BIND invocation</th>
<th>START invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE-RELEASE</td>
<td>PRE-RELEASE</td>
<td>PRE-RELEASE</td>
</tr>
</tbody>
</table>
In a multi-threaded environment, the value of the production status can change concurrently with processing within the service element. That implies, that the value can change after a PDU has been processed by the service element but before the same PDU is handled by the application. Because the service element cannot guarantee that the result of a test is still valid when the PDU reaches the application, this specification does not require that the service element check the production status.

This specification does not exclude that implementations of the service element check the production status and reject BIND or START invocation if required. If both the API and the application are single-threaded, the application could rely on such checks. However, applications cannot expect that other implementations provide the same service. Therefore, applications wishing to maintain substitutability of API components should not rely on such behavior.

### 2.2.8.3 Rejecting Invocations of TRANSFER-DATA and THROW-EVENT Operations

For CLTU–TRANSFER–DATA returns, reference [5] requires that the provider insert the next expected CLTU identification and the available CLTU buffer size. For CLTU–THROW–EVENT returns, reference [5] requires that the provider insert the next expected event invocation identifier. These parameters are available to the service element via the procedures described in 2.2.5. However, the following must be considered.

A service user is not required to wait for a CLTU–TRANSFER–DATA return before transmitting the next CLTU–TRANSFER–DATA invocation. Therefore, several CLTU–TRANSFER–DATA invocations can be in transit. Depending on the implementation of the service element and of the provider application, CLTU–TRANSFER–DATA invocations might be queued between the service element and the application. In such a case, the service element cannot know what values to insert for the next CLTU identification and the available buffer size when it needs to generate a CLTU–TRANSFER–DATA return with a negative result. The same considerations apply to the CLTU–THROW–EVENT operation.

Therefore, this specification defines a procedure for the CLTU–TRANSFER–DATA operation and for the CLTU–THROW–EVENT operation, which deviates from the standard approach described in reference [6]. When a check performed by the service element fails, the service element can set the appropriate diagnostic in the operation object and pass the operation object to the application. The application is expected to check the result of an...
invocation. If the result is negative, the application should insert the next expected CLTU identification and the available buffer size or the next expected event invocation identifier into the operation object and then pass it back to the service element using the method `InitiateOpReturn()` in the interface `ISLE_ServiceInitiate`.

This specification does not exclude that implementations generate a CLTU–TRANSFER–DATA return or a CLTU–THROW–EVENT return if it is possible to insert the correct values for the return parameters. An implementation can apply any of the following approaches:

a) an implementation can always pass invocations for which a check has failed to the application;

b) an implementation can prevent queuing of invocations by withholding an invocation until the previous invocation has been confirmed by the application. In that case, it can always generate the appropriate return when needed; or

c) an implementation can decide to pass invocations to the application on a case by case basis.

Applications wishing to maintain substitutability of API components should always expect to receive CLTU–TRANSFER–DATA invocations and CLTU–THROW–EVENT invocations with a negative result from the service element.

2.2.8.4 Processing of TRANSFER-DATA Invocations

2.2.8.4.1 Blocked State of the Service Instance

When a CLTU cannot be radiated because the production status becomes non-operational or because the latest radiation start time expired, the service instance becomes blocked and further CLTU–TRANSFER–DATA invocations must be rejected. In order to clear the situation, the service user must invoke a CLTU–STOP operation followed by a CLTU–START operation.

The event causing the blocked state of the service instance can depend on the production status, which can change concurrently with processing in the service element. In a multi-threaded environment, the service element cannot guarantee that a CLTU–TRANSFER–DATA invocation that passed the test of the blocked state is still valid when it reaches the application. Therefore, this specification does not require the service element to perform that check.

This specification does not exclude that implementations check the blocked state of the service instance. If both the API and the application are single-threaded, the application could rely on such checks. However, applications cannot expect that other implementations provide the same service. Therefore, applications wishing to maintain substitutability of API components should not rely on such behavior.
2.2.8.4.2 Checking of Time Parameters

CLTU–TRANSFER–DATA invocations carry parameters that specify the earliest and latest radiation times. Reference [5] requires the service provider to check that these times are not expired at the time the invocation reaches the provider. It cannot be excluded that such a time expires after the invocation has been processed by the service element, but before it reaches the application. Therefore, this specification does not require the service element to perform these checks. The service element is, however, required to verify that time periods are defined in a consistent manner.

This specification does not exclude that implementations check times against current time. However, applications wishing to maintain substitutability of API components should not rely on such behavior.

2.2.8.5 Production Time

Reference [5] defines a production period, i.e. the period in which the CLTU production engine is able to radiate CLTUs. This period must overlap with the scheduled provision period of the service instance but need not be the same. Reference [5] requires the service provider to check the validity of CLTU–START invocations and CLTU–TRANSFER–DATA invocations against the production period.

This specification does not require a service element to perform these checks, as they are related to service production and not to service provisioning.

2.3 PACKAGE CLTU OPERATIONS

Figure 2-2 shows the operation object interfaces required for the CLTU service. The package CLTU Operations adds operation objects for the following CLTU operations:

- CLTU–START;
- CLTU–TRANSFER–DATA;
- CLTU–ASYNC–NOTIFY;
- CLTU–STATUS–REPORT;
- CLTU–GET–PARAMETER;
- CLTU–THROW–EVENT.

For other operations the API uses the common operation objects defined in reference [6].
Figure 2-2: CLTU Operation Objects

Table 2-5 maps CLTU operations to operation object interfaces.

Table 2-5: Mapping of CLTU Operations to Operation Object Interfaces

<table>
<thead>
<tr>
<th>CLTU Operation</th>
<th>Operation Object Interface</th>
<th>Defined in Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLTU–BIND</td>
<td>ISLE_Bind</td>
<td>SLE Operations</td>
</tr>
<tr>
<td>CLTU–UNBIND</td>
<td>ISLE_Unbind</td>
<td>SLE Operations</td>
</tr>
<tr>
<td>CLTU Operation</td>
<td>Operation Object Interface</td>
<td>Defined in Package</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>CLTU–START</td>
<td>ICLTU_Start</td>
<td>CLTU Operations</td>
</tr>
<tr>
<td>CLTU–STOP</td>
<td>ISLE_Stop</td>
<td>SLE Operations</td>
</tr>
<tr>
<td>CLTU–TRANSFER–DATA</td>
<td>ICLTU_TransferData</td>
<td>CLTU Operations</td>
</tr>
<tr>
<td>CLTU–ASYNC–NOTIFY</td>
<td>ICLTU_AsyncNotify</td>
<td>CLTU Operations</td>
</tr>
<tr>
<td>CLTU–STATUS–REPORT</td>
<td>ICLTU_StatusReport</td>
<td>CLTU Operations</td>
</tr>
<tr>
<td>CLTU–GET–PARAMETER</td>
<td>ICLTU_GetParameter</td>
<td>CLTU Operations</td>
</tr>
<tr>
<td>CLTU–THROW–EVENT</td>
<td>ICLTU_ThrowEvent</td>
<td>CLTU Operations</td>
</tr>
<tr>
<td>CLTU–PEER–ABORT</td>
<td>ISLE_PeerAbort</td>
<td>SLE Operations</td>
</tr>
</tbody>
</table>
3 CLTU SPECIFIC SPECIFICATIONS FOR API COMPONENTS

3.1 API SERVICE ELEMENT

3.1.1 SERVICE INSTANCE CREATION

3.1.1.1 Although the Forward CLTU service allows only a single service instance to be bound at any point in time the service element shall not constrain the number of service instances supporting the service provider role or the service user role.

NOTE – More than one service instance might be needed for backup purposes. It is noted that a provider side service element is not required to check whether another service instance is already bound when receiving a CLTU-BIND invocation. Depending on the configuration of the service provider, different service instances might be used for different production engines.

3.1.2 SERVICE INSTANCE CONFIGURATION

3.1.2.1 The service element shall provide the interface ICLTU_SIAdmin for configuration of a provider-side service instance after creation.

3.1.2.2 The interface shall provide methods to set the following parameters, which the service element shall use to respond to GET-PARAMETER invocations received from the service user:

- a) bit-lock-required;
- b) maximum-sldu-length;
- c) modulation-frequency;
- d) modulation-index;
- e) plop-in-effect;
- f) rf-available-required; and
- g) subcarrier-to-bitrate-ratio.

NOTE – These parameters are defined in reference [5] for the operation CLTU-GET-PARAMETER.

3.1.2.3 The interface shall provide methods to set the following parameters, which the service instance shall use to initialize parameters of the status report:

- a) the maximum size of the CLTU buffer used to initialise the parameter cltu-buffer-available;
b) the value of the production-status at the time the service instance is configured;

c) the value of the uplink-status at the time the service instance is configured.

NOTE – Further configuration parameters must be set using the interface ISLE_SIAdmin specified in reference [6]. These include the parameter return-timeout-period required for the GET-PARAMETER operation.

3.1.2.4 The interface shall provide methods to set the following parameters, which the service instance shall use to control internal processing options:

the notification mode to allow deferred or non-deferred notification of a production status change to ‘interrupted’, used to initialize the parameter notification-mode.

NOTE – Further configuration parameters must be set using the interface ISLE_SIAdmin specified in reference [6]. These include the parameter return-timeout-period required for the GET-PARAMETER operation.

3.1.2.5 All configuration parameters must be set before the method ConfigCompleted() of the interface ISLE_SIAdmin is called. If one of the parameters is omitted or the value of a parameter is not within the range specified by reference [5], the method ConfigCompleted() shall return an error.

NOTES

1 As defined in reference [6], the service shall start processing of the service instance only after successful configuration.

2 The range of specific parameter values might be further constrained by service management. The service element shall only check on the limits specified by reference [5].

3.1.2.6 Configuration parameters listed in 3.1.2.2 as well as the maximum CLTU buffer size specified in 3.1.2.3 can be modified at any time during operation of the service instance. The service element shall always use the most recent value.

NOTE – Modification of these parameters during the scheduled provision period of the service instance might be inhibited by service management. Such constraints must be handled by the application.

3.1.2.7 Configuration parameters defined in 3.1.2.3, with the exception of the maximum CLTU buffer size specified in 3.1.2.3 must not be modified after successful return of the method ConfigCompleted() defined in the interface ISLE_SIAdmin. The effect of an attempt to set these parameters after completion of the configuration is undefined.
3.1.2.8 The values of all configuration parameters shall remain unmodified following a CLTU-UNBIND or CLTU-PEER-ABORT operation and following a protocol-abort.

3.1.2.9 The interface `ICLTU_SIAdmin` shall provide methods to retrieve the values of the configuration parameters. The values returned by these methods before configuration has been completed are undefined.

3.1.3 STATUS INFORMATION

3.1.3.1 Status Parameters

3.1.3.1.1 The service element shall maintain status parameters for every service instance and uses them for generation of status reports, notifications, and for CLTU–GET–PARAMETER returns.

NOTES

1. The parameters are defined in reference [5] for the operations CLTU-ASYNC-NOTIFY, CLTU-STATUS-REPORT, and CLTU-GET-PARAMETER.

2. Handling of the parameter `reporting-cycle` defined for the CLTU-GET-PARAMETER operation is specified in reference [6].

3.1.3.1.2 The service element shall update the following status parameters in the methods of the interface `ICLTU_SIUpdate` described in 3.1.2.3.

   a) `cltu-identification-last-processed`;
   b) `cltu-status` of the CLTU last processed;
   c) `radiation-start-time` of the CLTU last processed;
   d) `cltu-identification-last-OK`;
   e) `radiation-stop-time` of the CLTU last OK;
   f) `production-status`;
   g) `uplink-status`;
   h) `number-of-cltus-processed`; and
   i) `number-of-cltus-radiated`.

NOTE – The initial values of these parameters following configuration of the service instance are defined in A4.2.

3.1.3.1.3 The service element shall handle the parameter `expected-cltu-identification` as defined by the following specifications:
NOTE – The parameter expected-cltu-identification can be requested by a CLTU-GET-PARAMETER invocation.

a) the parameter shall be set to zero when the service instance is configured;

b) [V1:] for version 1, when the application transmits a CLTU-START return with a positive result, the value shall be set to the value of the parameter first-cltu-identification in the CLTU-START invocation, provided that parameter is not ‘null’;

c) [V2:] for version 2, when the application transmits a CLTU-START return with a positive result, the value shall be set to the value of the parameter first-cltu-identification in the CLTU-START invocation;

d) the value shall be copied from every CLTU-TRANSFER-DATA return issued by the application.

3.1.3.1.4 The service element shall handle the parameter expected-event-invocation-identifier as defined by the following specifications:

NOTE – The parameter expected-cltu-identification can be requested by a CLTU-GET-PARAMETER invocation.

a) the parameter shall be set to zero when the service instance is configured;

b) the value shall be copied from every CLTU-THROW–EVENT return issued by the application.

3.1.3.1.5 The service element shall handle the parameter cltu-buffer-available as defined by the following specifications:

a) at configuration time, the value shall be copied from the configuration parameter maximum-cltu-buffer, defined in 3.1.2.3;

b) when the application transmits a CLTU-TRANSFER-DATA return, the value shall be copied from the parameter set by the application;

c) the value shall be updated whenever passed as argument by one of the methods in the interface ICLTU_SIUpdate;

d) the value is set to the configured maximum CLTU buffer size whenever the method BufferEmpty() is called on the interface ICLTU_SIUpdate;

e) when the application transmits a CLTU-STOP return with a positive result, the value shall be copied from the configuration parameter maximum-cltu-buffer;

f) when the application transmits a CLTU-ASYNC-NOTIFY invocation with the parameter notification-type set to ‘buffer empty’, the value shall be copied from the configuration parameter maximum-cltu-buffer;
g) following a CLTU-PEER-ABORT operation and following protocol-abort, the value shall be copied from the configuration parameter maximum-cltu-buffer.

3.1.3.1.6 The service element shall handle the parameter number-of-cltus-received as defined by the following specifications:

a) the parameter shall be set to zero when the service instance is configured;

b) the parameter shall be incremented whenever the application transmits a CLTU-TRANSFER-DATA return with a positive result.

3.1.3.1.7 Except for the parameter cltu-buffer-available, status parameters defined in this specification shall not be modified as result of CLTU-UNBIND, CLTU-PEER-ABORT, or protocol abort.

3.1.3.1.8 The interface ICLTU_SIUpdate shall provide methods to retrieve the values of all status parameters. The values returned by these methods before configuration has been completed are undefined.

3.1.3.2 Update of Status Information by the Application

3.1.3.2.1 The service element shall provide the interface ICLTU_SIUpdate for every service instance, which must be used by the application to inform the service element of specific events in the production process.

3.1.3.2.2 When the methods of this interface are called the service element shall:

a) update the status parameters according to the arguments passed with the methods;

b) generate and transmit the following notifications if requested by the application and if the state of the service instance is ‘ready’ or ‘active’:

1) ‘cltu radiated’;
2) ‘sldu expired’;
3) ‘production interrupted’;
4) ‘production halted’;
5) ‘production operational’;
6) ‘buffer empty’;
7) ‘action list completed’;
8) ‘action list not completed’; and
9) ‘event condition evaluated to false’.
NOTE – The application can opt to generate and transmit the notifications itself using the interface ISLE_ServiceInitiate as for other PDUs.

3.1.3.2.3 The application must inform the service element of the events defined in 3.1.3.2.5 to 3.1.3.2.12 via the interface ICLTU_SIUpdate during the complete lifetime of the service instance, independent of the state of the service instance.

NOTE – This applies regardless of whether the application opts or not opts to generate and transmit the notifications itself using the interface ISLE_ServiceInitiate as for other PDUs.

3.1.3.2.4 The application should invoke the methods of the interface ICLTU_SIUpdate when one of the events defined in 3.1.3.2.13 occurs to generate the appropriate notification and send it to the service user.

NOTES

1 The methods described in 3.1.3.2.5 to 3.1.3.2.12 update status parameters maintained by the service instance. Status information must be updated in periods in which the service user is not connected such that it is up to date following a successful BIND operation. Failure to report one of the events defined in 3.1.3.2.5 to 3.1.3.2.12 can result in inconsistent status information.

2 Generation and transmission of notifications can be disabled by a method argument if this feature is not wanted.

3 The methods described in 3.1.3.2.13 do not affect status information maintained by the service instance. Therefore, an application generating and transmitting notifications itself does not need to call these methods.

3.1.3.2.5 The application shall call the method RadiationStarted() whenever radiation of a CLTU started. The method shall perform the following actions:

a) it shall increment the parameter number-of-cltus-processed;

b) it shall update the parameters cltu-identification-last-processed and radiation-start-time of the CLTU last processed according to the arguments passed to the method;

c) it shall set the parameter cltu-status to radiation-started;

d) it shall update the parameter cltu-buffer-available according to the argument passed to the method.

NOTE – Due to performance considerations, the method shall not perform any checks. The application must ensure that the preconditions specified in A4.2 are fulfilled.
3.1.3.2.6 The application shall call the method CltuRadiated() whenever radiation of a CLTU completed. The method shall perform the following actions:

a) it shall increment the parameter number-of-cltus-radiated;

b) it shall copy the identification of the cltu-last-processed to the parameter cltu-last-ok;

c) it shall set radiation-stop-time of the CLTU last OK to the value passed as argument;

d) it shall update the radiation-start-time of the CLTU last processed, if this argument is supplied by the application;

NOTE – If the radiation start time is not known precisely at the time the CLTU processing is started, the application may provide an estimate only. Passing the start time to the method CltuRadiated() shall allow storing a more precise value.

e) it shall set the parameter cltu-status of the CLTU last processed to ‘radiated’;

f) on request of the application, it shall send the notification ‘cltu radiated’ if the state of the service instance is ‘ready’ or ‘active’.

NOTE – Transmission of the notification must not be requested unless a radiation report has been requested for the CLTU by the service user. This cannot be checked by the service element.

3.1.3.2.7 The application shall call the method CltuNotStarted() whenever radiation of a CLTU could not be started, because:

a) the latest radiation start time expired (‘expired’); or

b) the production status was interrupted (‘production interrupted’).

3.1.3.2.8 The method CltuNotStarted() shall perform the following actions:

a) it shall increment the parameter number-of-cltus-processed;

b) it shall set the parameter cltu-identification-last-processed to the value passed as argument;

c) it shall set the parameters radiation-start-time of the CLTU last processed to NULL;

d) if the failure reason is ‘expired’, it shall set the parameter cltu-status of the CLTU last processed to ‘expired’;

e) if the failure reason is ‘production interrupted’ it shall set the parameter cltu-status of the CLTU last processed to ‘radiation not started’;
f) it shall update the parameter `cltu-buffer-available` according to the argument passed to the method;

g) on request of the application, it shall send one of the following notifications:

1) ‘sldu expired’, if the failure reason is ‘expired’;

2) ‘production interrupted’, if the failure reason is ‘production interrupted’ and ‘deferred notification’ is in effect;

NOTE – The event ‘CLTU not started’ can only occur if the state of the service instance is ‘active’. If the state of the service instance changes due to an abort after invocation of the method and before the notification can be transmitted, the service element shall inform the application using an appropriate return code.

3) if ‘immediate notification’ is in effect, and the failure reason is ‘production interrupted’, the API shall reject the request.

NOTE – If the production status changes to ‘interrupted’ when no CLTU is being radiated, and the change is notified immediately, the application shall not attempt to start radiation of a CLTU. Therefore the method `CltuNotStarted()` must not be called.

3.1.3.2.9 The application shall call the method `ProductionStatusChange()` whenever the production status changes. The method shall perform the following steps:

a) it shall set the parameter `production-status` to the value passed as argument;

b) it shall update the parameter `cltu-buffer-available` according to the argument passed to the method.

c) if the new value of the `production-status` is 'interrupted' or 'halted' and value of the parameter `cltu-status` is 'radiation started', the `cltu-status` shall be set to 'interrupted';

d) on request of the application, it shall send one of the following notifications if the state of the service instance is 'ready' or 'active':

1) ‘production operational’, if the new value of the production status is ‘operational’ and the ‘reported production status’ is not 'operational';

2) ‘production halted’, if the new value of the production status is ‘halted’;

3) ‘production interrupted’, if the new value of the production status is ‘interrupted’ and ‘immediate notification’ is in effect;
NOTES

1  If ‘deferred notification’ was configured, the notification is not generated unless a CLTU has started radiation. When the application attempts radiating the next CLTU, the application must call CltuNotStarted() with the reason set to ‘interrupted’; this call then generates the notification (see also 3.1.3.2.8 item 2)).

2  If the value of the production status has not changed or the new value is ‘configured’ no notification is sent

4) ‘production interrupted’, if the new value of the production status is ‘interrupted’ and ‘deferred notification’ is in effect and the status of the cltu-identification-last-processed was ‘radiation started’ when the method was invoked.

NOTE – This specification covers change of the production status to ‘interrupted’ while a CLTU is being radiated. When radiation starts for a CLTU, the application must call CltuStarted(), which sets the status of the CLTU to ‘radiation started’. This ensures that the API has the information that the production status has changed to ‘interrupted’ during radiation.

3.1.3.2.10 Whenever the service element sends one of the notifications 'production operational', 'production interrupted', or 'production halted', it shall memorize the reported status.

NOTE – This 'reported production status' shall be used to prevent that the notification 'production operational' is sent to a user that was not informed of a change to a non operational status either because the service instance was not bound when the change occurred or because no packets were affected by the production status 'interrupted'.

3.1.3.2.11 The application shall call the method Set_UplinkStatus() whenever the uplink status changes. The method shall set the value of the parameter uplink-status to the argument passed.

3.1.3.2.12 The application shall call the method BufferEmpty() whenever the application has no further CLTUs buffered for this service instance. The method BufferEmpty() shall perform the following actions:

a)  it shall set the parameter CLTU buffer size to the value of the parameter ‘maximum cltu buffer size’, defined in 3.1.2.3 item a);

b)  if requested by the application, it shall send the notification ‘buffer empty’ if the state of the service instance is ‘ready’ or ‘active’.

NOTE – The method must not be called when the packet buffer is cleared due to one of the events for which reference [5] requires discarding of buffered CLTUs.
3.1.3.2.13 The application shall call the method `EventProcCompleted()` when processing of an event requested by an accepted CLTU-THROW-EVENT operation completes:

   a) when calling the method `EventProcCompleted()` the application shall provide the following information using the method arguments:

      1) the event invocation identification as copied from the CLTU-THROW-EVENT invocation;

      2) the result of execution, indicating whether:

         – the action list associated with the event was completely executed;

         – at least one of the actions in the associated action list failed; or

         – the condition associated with the event evaluated to false.

   b) the method `EventProcCompleted()` shall perform the following actions:

      1) it shall send the notification ‘action list completed’ if the action list associated with the event was completely executed;

      2) it shall send the notification ‘action list not completed’ if at least one of the actions in the associated action list failed;

      3) it shall send the notification ‘event action evaluated to false’ if the condition associated with the event evaluated to false.

3.1.3.2.14 The service element shall apply the following rules for checking of consistency:

   NOTE – Further details concerning the checks performed and return codes passed to the caller are defined in A4.2.

   a) the methods `CltuStarted()` and `CltuRadiated()` shall perform no checks;

   NOTE – These methods must be called frequently during nominal operation. Due to performance considerations, the service element shall fully rely on the application to ensure that the methods are used correctly. Detailed preconditions are defined in A4.2.

   b) for other methods the service element shall verify that the method call is consistent with the values of the status parameters before the method was invoked. If the check fails, the service element shall proceed as follows:

      1) if applying the update results in a consistent set of status parameters, the service element shall perform the update and shall send the notification (if requested) but shall return an error code to the application as a warning;
2) if an update would result in inconsistent status parameters, the service element shall not perform the update, shall not send any notifications, and shall return an appropriate error code.

3.1.4 PROCESSING OF CLTU PROTOCOL DATA UNITS

NOTES

1

The service element shall process CLTU PDUs according to the general specifications in reference [6]. This section only addresses additional checks and processing steps defined for the CLTU service. CLTU specific checks defined in reference [5] but not listed in this section, must be performed by the application. Subsection 2.2.8 provides a discussion of the borderline between the application and the service element.

2

It is noted that 3.1.3 defines processing requirements for other PDUs with respect to update of status information and generation of notifications. Subsection A3 defines the checks that operation objects shall perform when the methods VerifyInvocationArguments() and VerifyReturnArguments() are called. Reference [6] contains specifications defining how the service element shall handle error codes returned by these methods.

3.1.4.1 CLTU-TRANSFER-DATA

3.1.4.1.1 When receiving a CLTU–TRANSFER–DATA invocation, the service element shall perform the following checks in addition to the checks defined in reference [6] for all PDUs. These checks shall be performed in the sequence specified:

   a) if the ‘earliest radiation time’ and the ‘latest radiation time’ are both specified, the ‘earliest radiation time’ must not be later than the ‘latest radiation time’;

   b) the size of the CLTU contained in the PDU must not be larger than the value of the configuration parameter ‘maximum-sldu-length’ allows.

3.1.4.1.2 If any of these checks fail, or a return PDU with a negative result must be generated because a check defined in reference [6] failed, the service element shall proceed as defined by the following specifications:

   a) if the service element can guarantee that all preceding CLTU–TRANSFER–DATA invocations have already been processed by the application or that the PDU processed by the service element shall be the first CLTU–TRANSFER–DATA invocation following START, the service element can generate a CLTU–TRANSFER–DATA return with a negative result and transmit that to the service user;

   NOTE – In that case, the service element shall use the status parameters ‘expected-cltu-identification’ and ‘cltu-buffer-available’ to set the parameters of the CLTU–TRANSFER–DATA return.
b) if the conditions defined in a) are not met or cannot be verified the service element shall set the result parameter to 'negative', shall set the appropriate diagnostic in the operation object, and shall pass the operation object to the application.

c) in order to ensure that the result parameter of the operation object always has a valid reading, the service element shall set the result parameter to ‘positive’ if all checks performed by the service element succeeded.

NOTES

1 It is noted that this processing deviates from the standard way in which confirmed PDUs are handled by the service element. The reasons for this specification are explained in 2.2.8.3.

2 A service element shall not be required to generate and transmit a CLTU-TRANSFER–DATA return also when it could verify that the conditions defined in 3.1.4.1.2 item a) are met. A service element can use one of the following approaches:
   – ensure that no CLTU–TRANSFER–DATA invocations are queued between the service element and the application, and never pass an invocation for which a check has failed to the application;
   – always pass CLTU–TRANSFER–DATA invocations to the application;
   – decide on a case by case basis.

3 Implementations should document the approach used. Applications should always expect that the service element passes CLTU–TRANSFER–DATA invocations with a negative result if substitutability of SLE API components shall be maintained.

4 Processing expected from the application is defined in 3.3.

3.1.4.2 CLTU-THROW-EVENT

3.1.4.2.1 If a CLTU–THROW–EVENT return PDU with a negative result must be generated because a check defined in reference [6] failed, the service element shall proceed as follows:

   a) If the service element can guarantee that all preceding CLTU–THROW–EVENT invocations have already been processed by the application, or that the PDU processed by the service element is the first CLTU–THROW–EVENT invocation following BIND, the service element can generate a CLTU–THROW–EVENT return with a negative result and transmit that to the service user.
NOTE – In that case, the service element shall use the status parameter `expected-event-invocation-identifier` to set the parameter of the CLTU–THROW–EVENT return.

b) If the conditions defined in a) are not met or cannot be verified, the service element shall set the result parameter to ‘negative’, set the appropriate diagnostic in the operation object, and pass the operation object to the application.

c) In order to ensure that the result parameter of the operation object always has a valid reading, the service element shall set the result parameter to ‘positive’ if all checks performed by the service element succeeded.

NOTES

1 It is noted that this processing deviates from the standard way in which confirmed PDUs are handled by the service element. The reasons for this specification are explained in 2.2.8.3.

2 A service element is not required to generate and transmit a CLTU–THROW–EVENT return also when it could verify that the conditions defined in a) are met. A service element can use one of the following approaches:
   – ensure that no CLTU–THROW–EVENT invocations are queued between the service element and the application, and never pass an invocation for which a check has failed to the application;
   – always pass CLTU–THROW–EVENT invocations to the application;
   – decide on a case by case basis.

3 Implementations should document the approach used. Applications should always expect the service element to pass CLTU–THROW–EVENT invocations with a negative result if substitutability of SLE API components shall be maintained.

4 Processing expected from the application is defined in 3.3.

3.1.5 SERVICE INSTANCE SPECIFIC OPERATION FACTORY

For CLTU service instances, the interface `ISLE_SIOpFactory` specified in reference 3.2 shall support creation and configuration of operation objects for all operations specified in 3.2 with exception of the object for the operation CLTU–STATUS–REPORT.

NOTE – The initial values of parameters that shall be set for CLTU specific operation objects are defined in annex A. The operation CLTU–STATUS–REPORT is handled autonomously by the provider-side service element. There is no need for the application to create this object.
3.2 SLE OPERATIONS

3.2.1 The component ‘SLE Operations’ shall provide operation objects for the following CLTU operations in addition to those specified in reference [6]:

a) CLTU–START;

b) CLTU–TRANSFER–DATA;

c) CLTU–ASYNC–NOTIFY;

d) CLTU–STATUS–REPORT;

e) CLTU–GET–PARAMETER;

f) CLTU–THROW–EVENT.

3.2.2 The operation factory shall create the operation objects specified in 3.2.1 when the requested service type is CLTU.

3.2.3 The operation factory shall additionally create the following operation objects specified in reference [6] when the requested service type is CLTU:

a) SLE–BIND;

b) SLE–UNBIND;

c) SLE–PEER–ABORT;

d) SLE–STOP;

e) SLE–SCHEDULE–STATUS–REPORT.

3.3 SLE APPLICATION

NOTE – This section summarizes specific obligations of a CLTU provider application using the SLE API.

3.3.1 Following creation of a service instance, and setting of the configuration parameters defined in reference [6], the application shall set the configuration parameters defined in 3.1.1 via the interface ICLTU_SIAdmin.

3.3.2 The application shall inform the service element of all events defined in 3.1.3.2.3 by invocation of the appropriate methods of the interface ICLTU_SIUpdate.

3.3.3 When receiving a CLTU–TRANSFER–DATA invocation via the interface ISLE_ServiceInform, the application shall check the result parameter of the operation object and perform the following steps:

a) if the result is negative, the application shall set the expected CLTU identification and the available buffer size and then pass the operation back to the service element using
the method InitiateOpReturn() in the interface ISLE_ServiceInitiate;

b) if the result is positive, the application shall perform the checks not specified in 3.1.4:

1) if any of these checks fail, the application shall set the appropriate diagnostic, the expected CLTU identification, and the available buffer size and then pass the operation object to the service element using the method InitiateOpReturn() in the interface ISLE_ServiceInitiate;

2) if all checks succeed, the application shall store the CLTU to the CLTU buffer, set a positive result, the expected CLTU identification, and the available buffer size and then pass the operation object back to the service element using the method InitiateOpReturn() in the interface ISLE_ServiceInitiate.

3.3.4 When receiving a CLTU–THROW–EVENT invocation via the interface ISLE_ServiceInform, the application shall check the result parameter of the operation object and perform the following steps:

a) if the result is negative, the application shall set the expected event invocation and pass the operation back to the service element using the method InitiateOpReturn() in the interface ISLE_ServiceInitiate;

b) if the result is positive, the application shall perform the checks required:

1) if any of these checks fail, the application shall set the appropriate diagnostic and the expected event invocation identifier and then pass the operation object to the service element using the method InitiateOpReturn() in the interface ISLE_ServiceInitiate;

2) if all checks succeed, the application shall perform the required operation, set a positive result, and the expected event invocation identifier and then pass the operation object back to the service element using the method InitiateOpReturn() in the interface ISLE_ServiceInitiate.
ANNEX A

CLTU SPECIFIC INTERFACES

(This annex is part of the Recommended Practice)

A1 INTRODUCTION

This annex specifies CLTU-specific

a) data types;

b) interfaces for operation objects; and

c) interfaces for service instances.

The specification of the interfaces follows the design patterns, conventions and the additional conventions described in reference [6].

The presentation uses the notation and syntax of the C++ programming language as specified in reference [7].
A2 CLTU TYPE DEFINITIONS

File CLTU_Types.h

The following types have been derived from the ASN.1 module CCSDS-SLE-TRANSFER-CLTU-STRUCTURES in reference [5]. The source ASN.1 type is indicated in brackets. For all enumeration types a special value ‘invalid’ is defined, which is returned if the associated value in the operation object has not yet been set, or is not applicable in case of a choice.

CLTU Identification [Cltu Identification]

typedef unsigned long CLTU_Id;
typedef unsigned long CLTU_BufferSize;

Size of the CLTU buffer or the remaining free space in the buffer measured in octets.

typedef enum CLTU_StartDiagnostic
{
  cltuSTD_outOfService = 0,
  cltuSTD_unableToComply  = 1,
  cltuSTD_productionTimeExpired = 2,
  cltuSTD_invalidCltuId = 3,
  cltuSTD_invalid     = -1
} CLTU_StartDiagnostic;

CLTU Transfer Data Diagnostic [DiagnosticCltuTransferData]

typedef enum CLTU_TransferDataDiagnostic
{
  cltuXFD_unableToProcess = 0,
  cltuXFD_unableToStore   = 1,
  cltuXFD_outOfSequence   = 2,
  cltuXFD_inconsistenceTimeRange = 3,
  cltuXFD_invalidTime     = 4,
  cltuXFD_lateSldu        = 5,
  cltuXFD_invalidDelayTime = 6,
  cltuXFD_cltuError       = 7,
  cltuXFD_invalid         = -1
} CLTU_TransferDataDiagnostic;

CLTU Get Parameter Diagnostic [DiagnosticCltuGetParameter]

typedef enum CLTU_GetParameterDiagnostic
{
  cltuGP_unknownParameter = 0,
  cltuGP_invalid         = -1
} CLTU_GetParameterDiagnostic;

CLTU Throw Event Diagnostic [DiagnosticCltuThrowEvent]

typedef enum CLTU_ThrowEventDiagnostic
{
  cltuTED_operationNotSupported = 0,
  cltuTED_outOfSequence        = 1,
  cltuTED_noSuchEvent          = 2,
  cltuTED_invalid              = -1
}
Notification type [CltuNotification]

typedef enum CLTU_NotificationType
{
    cltuNT_cltuRadiated                = 0,
    cltuNT_slduExpired                 = 1,
    cltuNT_unableToProcess             = 2,
    cltuNT_productionHalted            = 3,
    cltuNT_productionOperational       = 4,
    cltuNT_bufferEmpty                 = 5,
    cltuNT_actionListCompleted         = 6,
    cltuNT_actionListNotCompleted      = 7,
    cltuNT_eventConditionEvFalse       = 8,
    cltuNT_invalid                     = -1
} CLTU_NotificationType;

CLTU Service Parameters [CltuParameterName]

typedef enum CLTU_ParameterName
{
    cltuPN_bitLockRequired             = 3,
    cltuPN_deliveryMode                = 6,
    cltuPN_expectedEventInvocationId  = 9,
    cltuPN_expectedSlduIdentification  = 10,
    cltuPN_maximumSlduLength           = 21,
    cltuPN_modulationFrequency         = 22,
    cltuPN_modulationIndex             = 23,
    cltuPN_plopInEffect                = 25,
    cltuPN_reportingCycle              = 26,
    cltuPN_returnTimeoutPeriod         = 29,
    cltuPN_rfAvailableRequired         = 31,
    cltuPN_subcarrierToBitRateRatio    = 34,
    cltuPN_invalid                     = -1
} CLTU_ParameterName;

The parameter name values are taken from the type ParameterName in ASN.1 module CCSDS-SLE-TRANSFER-SERVICE-COMMON-TYPES.

Modulation Index [CltuGetParameter]

typedef unsigned short CLTU_ModulationIndex;

Modulation Frequency [CltuGetParameter]

typedef unsigned long CLTU_ModulationFrequency;

The frequency of the primary modulation of the RF carrier measured in 1/10 Hz.

Sub-carrier Divisor [SubcarrierDivisor]

typedef unsigned short CLTU_SubcarrierDivisor;

Divisor of the sub-carrier frequency. If direct carrier modulation, the value is 1.
PLOP in Effect [CltuGetParameter]

typedef enum CLTU_PlopInEffect
{
    cltuPIE_plop1      =  0,
    cltuPIE_plop2      =  1,
    cltuPIE_invalid    = -1
} CLTU_PlopInEffect;

CLTU Status [CltuStatus]

typedef enum CLTU_Status
{
    cltuST_expired              = sleFDS_expired,
    cltuST_interrupted          = sleFDS_interrupted,
    cltuST_radiationStarted     = sleFDS_productionStarted,
    cltuST_radiated             = sleFDS_radiated,
    cltuST_radiationNotStarted  = sleFDS_productionNotStarted,
    cltuST_invalid              = -1
} CLTU_Status;

Describes the state of the last processed CLTU. It is defined as a subset of the type SLE_ForwardDuStatus specified in reference [6].

Production Status [ProductionStatus]

typedef enum CLTU_ProductionStatus
{
    cltuPS_operational       =  0,
    cltuPS_configured        =  1,
    cltuPS_interrupted       =  2,
    cltuPS_halted            =  3,
    cltuPS_invalid           = -1
} CLTU_ProductionStatus;

The status of the CLTU production engine.

Up-link Status [UplinkStatus]

typedef enum CLTU_UplinkStatus
{
    cltuUS_notAvailable     =  0,
    cltuUS_noRfAvailable    =  1,
    cltuUS_noBitLock        =  2,
    cltuUS_nominal          =  3,
    cltuUS_invalid          = -1
} CLTU_UplinkStatus;

Identifier of a Thrown Event [EventInvocationId]

typedef unsigned long CLTU_EventInvocationId;

CLTU Failure

typedef enum CLTU_Failure
{
    cltuF_expired         =  0,
    cltuF_interrupted     =  1   /* production interrupted */
} CLTU_Failure;
Identifies the reason why radiation of a CLTU could not be started.

**CLTU Abort Reason**

```c
typedef enum CLTU_AbortReason {
    cltuAR_interrupted        =  0, /* production interrupted */
    cltuAR_halted             =  1  /* production halted */
} CLTU_AbortReason;
```

Identifies the reason why radiation of a CLTU could not be completed.

**CLTU Notification Mode**

```c
typedef enum CLTU_NotificationMode {
    cltuNM_immediate        =  0,
    cltuNM_deferred         =  1,
    cltuNM_invalid          = -1
} CLTU_NotificationMode;
```

Identifies the mode for the ‘production interrupted’ notification.

**CLTU Event Processing Result**

```c
typedef enum CLTU_EventResult {
    cltuER_completed        =  0, /* action list completed */
    cltuER_notCompleted     =  1  /* action list not completed */
    cltuER_conditionFalse   =  2  /* event condition evaluated to false */
} CLTU_EventResult;
```

The result of processing a thrown event.
A3  CLTU OPERATION OBJECTS

A3.1  CLTU START OPERATION

Name  ICLTU_Start
GUID   {096578AF-CDC7-4f01-9B76-954ADA315CAB}
Inheritance: IUnknown – ISLE_Operation – ISLE_ConfirmedOperation
File   ICLTU_Start.H

The interface provides access to the parameters of the confirmed operation CLTU START.

Synopsis

```c
#include <CLTU_Types.h>
#include <ISLE_ConfirmedOperation.H>
interface ISLE_Time;
#define IID_ICLTU_Start_DEF { 0x96578af, 0xcdc7, 0x4f01, 
                           { 0x9b, 0x76, 0x95, 0x4a, 0xda, 0x31, 0x5c, 0xab } }
interface ICLTU_Start : ISLE_ConfirmedOperation
{
    virtual bool    Get_FirstCltuIdUsed() const = 0; /* for Version 1 only */
    virtual CLTU_Id Get_FirstCltuId() const = 0;
    virtual const ISLE_Time* Get_StartProductionTime() const = 0;
    virtual const ISLE_Time* Get_StopProductionTime() const = 0;
    virtual CLTU_StartDiagnostic Get_StartDiagnostic() const = 0;
    virtual void    Set_FirstCltuId( CLTU_Id id ) = 0;
    virtual void    Set_StartProductionTime( const ISLE_Time& startTime ) = 0;
    virtual void    Put_StartProductionTime( ISLE_Time* pstartTime ) = 0;
    virtual void    Set_StopProductionTime( const ISLE_Time& stopTime ) = 0;
    virtual void    Put_StopProductionTime( ISLE_Time* pstopTime ) = 0;
    virtual void    Set_StartDiagnostic( CLTU_StartDiagnostic diag ) = 0;
};
```

Methods

```c
bool Get_FirstCltuIdUsed() const;
```

[V1:] Returns TRUE if the first CLTU to be expected is specified and FALSE otherwise. This method is for Version 1 only.

```c
CLTU_Id Get_FirstCltuId() const;
```
Returns the first CLTU identification that the provider shall expect.

[V1:] If the method Get_FirstCltuIdUsed() returns FALSE, the value is undefined.

**Precondition:** [V1:] Get_FirstCltuIdUsed() returns TRUE.

```cpp
const ISLE_Time* Get_StartProductionTime() const;
```

Returns a pointer to the production start time if that parameter has been set. If the parameter has not been specified returns a NULL pointer.

```cpp
const ISLE_Time* Get_StopProductionTime() const;
```

Returns a pointer to the production start time if that parameter has been set. If the parameter has not been specified returns a NULL pointer.

```cpp
CLTU_StartDiagnostic Get_StartDiagnostic() const;
```

Returns the diagnostic code.

**Precondition:** the result is negative, and the diagnostic type is set to ‘specific’.

```cpp
void Set_FirstCltuId( CLTU_Id id );
```

Sets the first CLTU identification the provider shall accept.

[V1:] If this method is called, Get_FirstCltuIdUsed() returns TRUE.

```cpp
void Set_StartProductionTime( const ISLE_Time& startTime );
```

Sets the production start time to a copy of the input argument.

```cpp
void Put_StartProductionTime( ISLE_Time* pstartTime );
```

Stores the input argument to the parameter production start time.

```cpp
void Set_StopProductionTime( const ISLE_Time& stopTime );
```

Sets the production stop time to a copy of the input argument.

```cpp
void Put_StopProductionTime( ISLE_Time* pstopTime );
```

Stores the input argument to the parameter production stop time.

```cpp
void Set_StartDiagnostic( CLTU_StartDiagnostic diag );
```
Sets the result to ‘negative’, the diagnostic type to ‘specific’, and stores the value of the diagnostic code passed by the argument.

## Initial Values of Operation Parameters after Creation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
<th>Created by Service Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>first CLTU used</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>first CLTU Identification</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>start production time</td>
<td>NULL (not used)</td>
<td>NULL (not used)</td>
</tr>
<tr>
<td>stop production time</td>
<td>NULL (not used)</td>
<td>NULL (not used)</td>
</tr>
<tr>
<td>START diagnostic</td>
<td>‘invalid’</td>
<td>‘invalid’</td>
</tr>
</tbody>
</table>

## Checking of Invocation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>first CLTU Identification</td>
<td>[V2:] must be present, i.e. Get_FirstCltuIdUsed() returns TRUE. The required condition is only valid for Version 2 of the CLTU service.</td>
</tr>
</tbody>
</table>

## Additional Return Codes for VerifyInvocationArguments()

- **SLE_E_MISSINGARG** specification of the first CLTU identification is missing.

## Checking of Return Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>start production time</td>
<td>must not be NULL; if the start and the stop time are used, must be earlier than stop time</td>
</tr>
<tr>
<td>stop production time</td>
<td>if the start and the stop time are used, must be later than stop time</td>
</tr>
<tr>
<td>START diagnostic</td>
<td>must not be 'invalid' if the result is 'negative' and the diagnostic type is 'specific'</td>
</tr>
</tbody>
</table>

## Additional Return Codes for VerifyReturnArguments()

- **SLE_E_MISSINGARG** specification of the start production time is missing.
A3.2 CLTU TRANSFER DATA OPERATION

Name: ICLTU_TransferData
GUID: {cd799d7e-097d-11d3-a792-80954a16aa77}
Inheritance: IUnknown – ISLE_Operation – ISLE_ConfirmedOperation
File: ICLTU_TransferData.H

The interface provides access to the parameters of the confirmed operation CLTU-TRANSFER-DATA.

Synopsis
#include <CLTU_Types.h>
#include <ISLE_ConfirmedOperation.H>

interface ISLE_Time;
#define IID_ICLTU_TransferData_DEF { cd799d7e, 0x097d, 0x11d3, \
 { 0xa7, 0x92, 0x80, 0x95, 0x4a, 0x16, 0xaa, 0x77 } }

interface ICLTU_TransferData : ISLE_ConfirmedOperation
{
  virtual CLTU_Id Get_CltuId() const = 0;
  virtual CLTU_Id Get_ExpectedCltuId() const = 0;
  virtual const ISLE_Time* Get_EarliestRadTime() const = 0;
  virtual const ISLE_Time* Get_LatestRadTime() const = 0;
  virtual SLE_Duration Get_DelayTime() const = 0;
  virtual SLE_SlduStatusNotification Get_RadiationNotification() const = 0;
  virtual const SLE_Octet* Get_Data( size_t& length ) const = 0;
  virtual SLE_Octet* Remove_Data( size_t& length ) = 0;
  virtual CLTU_BufferSize Get_CltuBufferAvailable() const = 0;
  virtual CLTU_TransferDataDiagnostic Get_TransferDataDiagnostic() const = 0;
  virtual void Set_CltuId( CLTU_Id id ) = 0;
  virtual void Set_ExpectedCltuId( CLTU_Id id ) = 0;
  virtual void Set_EarliestRadTime( const ISLE_Time& earliestTime ) = 0;
  virtual void Put_EarliestRadTime( ISLE_Time* pearliestTime ) = 0;
  virtual void Set_LatestRadTime( const ISLE_Time& latestTime ) = 0;
  virtual void Put_LatestRadTime( ISLE_Time* platestTime ) = 0;
  virtual void Set_DelayTime( SLE_Duration delay ) = 0;
  virtual void Set_RadiationNotification( SLE_SlduStatusNotification ntf ) = 0;
  virtual void Set_Data( size_t length, const SLE_Octet* pdata ) = 0;
virtual void
  Put_Data( size_t length, SLE_Octet* pdata ) = 0;
virtual void
  Set_CltuBufferAvailable( CLTU_BufferSize bufAvail ) = 0;
virtual void
  Set_TransferDataDiagnostic
    ( CLTU_TransferDataDiagnostic diagnostic ) = 0;
};

Methods

CLTU_Id Get_CltuId() const;

Returns the CLTU identification.

CLTU_Id Get_ExpectedCltuId() const;

Returns the next expected CLTU identification. If the parameter has not been set returns zero.

const ISLE_Time* Get_EarliestRadTime() const;

Returns a pointer to the earliest radiation time, if the parameter has been specified. If the parameter is not set, returns a NULL pointer.

const ISLE_Time* Get_LatestRadTime() const;

Returns a pointer to the latest radiation time, if the parameter has been specified. If the parameter is not set, returns a NULL pointer.

SLE_Duration Get_DelayTime() const;

Returns the parameter delay time.

SLE_SlduStatusNotification Get_RadiationNotification() const;

Returns an indication whether a notification shall be returned when the CLTU has been radiated.

const SLE_Octet* Get_Data( size_t& length ) const;

Returns a pointer to the CLTU data in the object. The data must neither be modified nor deleted by the caller.

Arguments

length the number of bytes in the CLTU
SLE_Octet* Remove_Data(size_t& length);

Returns a pointer to the CLTU data and removes the data from the object. The client is expected to delete the data when they are no longer needed.

Arguments
length the number of bytes in the CLTU

CLTU_BufferSize Get_CltuBufferAvailable() const;

Returns the available CLTU buffer size in bytes if the argument has been set. If the parameter has not been set returns zero.

CLTU_TransferDataDiagnostic Get_TransferDataDiagnostic() const;

Returns the diagnostic code.

Precondition: the result is negative, and the diagnostic type is set to ‘specific’.

void Set_CltuId(CLTU_Id id);

Sets the CLTU identification for the CLTU transferred.

void Set_ExpectedCltuId(CLTU_Id id);

Sets the next expected CLTU identification.

void Set_EarliestRadTime(const ISLE_Time& earliestTime);

Sets the earliest radiation time to a copy of the input argument.

void Put_EarliestRadTime(ISLE_Time* pearliestTime);

Stores the input argument to the parameter earliest radiation time.

void Set_LatestRadTime(const ISLE_Time& latestTime);

Sets the latest radiation time to a copy of the input argument.

void Put_LatestRadTime(ISLE_Time* platestTime);

Stores the input argument to the parameter latest radiation time.

void Set_DelayTime(SLE_Duration delay);

Sets the parameter delay time.
void Set_RadiationNotification( SLE_SlduStatusNotification ntf );
Sets the indication whether a notification shall be sent when the CLTU has been radiated.

void Set_Data( size_t length, const SLE_Octet* pdata );
Copies length bytes from the address pdata to the internal CLTU data parameter.

Arguments
pdata    pointer to the CLTU data
length   the number of bytes in the CLTU

void Put_Data( size_t length, SLE_Octet* data );
Stores the CLTU data to the object. The operation object will eventually delete the data buffer.

Arguments
pdata    pointer to the CLTU data
length   the number of bytes in the CLTU

void Set_CltuBufferAvailable( CLTU_BufferSize bufAvail );
Sets the available CLTU buffer size in byte.

void Set_TransferDataDiagnostic( CLTU_TransferDataDiagnostic diagnostic );
Sets the result to 'negative', the diagnostic type to 'specific', and stores the value of the diagnostic code passed by the argument.

Initial Values of Operation Parameters after Creation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
<th>Created by Service Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLTU identification</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>expected CLTU identification</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>earliest radiation time</td>
<td>NULL (not used)</td>
<td>NULL (not used)</td>
</tr>
<tr>
<td>latest radiation time</td>
<td>NULL (not used)</td>
<td>NULL (not used)</td>
</tr>
<tr>
<td>delay time</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>radiation notification</td>
<td>'invalid'</td>
<td>'invalid'</td>
</tr>
<tr>
<td>CLTU buffer available</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Parameter | Created directly | Created by Service Instance
--- | --- | ---
transfer buffer diagnostic | ‘invalid’ | ‘invalid’

Checking of Invocation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>earliest radiation time</td>
<td>if earliest and latest radiation times are set, must be earlier than latest radiation time</td>
</tr>
<tr>
<td>latest radiation time</td>
<td>if earliest and latest radiation times are set, must be later than earliest radiation time</td>
</tr>
<tr>
<td>radiation notification</td>
<td>Must not be ‘invalid’</td>
</tr>
<tr>
<td>data</td>
<td>Must not be NULL</td>
</tr>
</tbody>
</table>

Additional Return Codes for `VerifyInvocationArguments()`

- **SLE_E_TIMERANGE**
  specification of the earliest and latest radiation times is inconsistent.

Checking of Return Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>expected CLTU identification</td>
<td>If result is ‘positive’, must be CLTU identification + 1</td>
</tr>
<tr>
<td>transfer buffer diagnostic</td>
<td>must not be ‘invalid’ if the result is ‘negative’ and the diagnostic type is ‘specific’</td>
</tr>
</tbody>
</table>
A3.3 CLTU ASYNC NOTIFY OPERATION

Name: ICLTU_AsyncNotify
GUID: {6F37EC88-EF7B-442a-AAE3-06C2E8A35D77}
Inheritance: IUnknown – ISLE_Operation
File: ICLTU_AsyncNotify.H

The interface provides access to the parameters of the unconfirmed operation CLTU-ASYNC-NOTIFY.

Synopsis

```c
#include <CLTU_Types.h>
#include <ISLE_Operation.H>
interface ISLE_Time;

define IID_ICLTU_AsyncNotify_DEF { 0x6f37ec88, 0xef7b, 0x442a, 
{ 0xa, 0xe3, 0x6, 0xc2, 0xe8, 0xa3, 0x5d, 0x77 } }

interface ICLTU_AsyncNotify : ISLE_Operation
{
  virtual CLTU_NotificationType Get_NotificationType() const = 0;
  virtual CLTU_EventInvocationId Get_EventThrownId() const = 0;
  virtual bool Get_CltusProcessed() const = 0;
  virtual CLTU_Id Get_CltuLastProcessed() const = 0;
  virtual const ISLE_Time* Get_RadiationStartTime() const = 0;
  virtual CLTU_Status Get_CltuStatus() const = 0;
  virtual bool Get_CltusRadiated() const = 0;
  virtual CLTU_Id Get_CltuLastOk() const = 0;
  virtual const ISLE_Time* Get_RadiationStopTime() const = 0;
  virtual CLTU_ProductionStatus Get_ProductionStatus() const = 0;
  virtual CLTU_UplinkStatus Get_UplinkStatus() const = 0;
  virtual void Set_NotificationType( CLTU_NotificationType notifyType ) = 0;
  virtual void Set_EventThrownId( CLTU_EventInvocationId id ) = 0;
  virtual void Set_CltuLastProcessed( CLTU_Id id ) = 0;
  virtual void Set_RadiationStartTime( const ISLE_Time& startTime ) = 0;
  virtual void Put_RadiationStartTime( ISLE_Time* pstartTime ) = 0;
  virtual void Set_CltuStatus( CLTU_Status status ) = 0;
  virtual void Set_CltuLastOk( CLTU_Id id ) = 0;
  virtual void Set_RadiationStopTime( const ISLE_Time& stopTime ) = 0;
} 
```
virtual void
  Put_RadiationStopTime( ISLE_Time* pstopTime ) = 0;
virtual void
  Set_ProductionStatus( CLTU_ProductionStatus status ) = 0;
virtual void
  Set_UplinkStatus( CLTU_UplinkStatus status ) = 0;
};

Methods

CLTU_NotificationType Get_NotificationType() const;
Returns the notification type.

CLTU_EventInvocationId Get_EventThrownId() const;
Returns the identification of the thrown event to which the notification refers.

Precondition: notification type is one of 'action list completed', 'action list not completed', 'event condition evaluate to false'.

bool Get_CltusProcessed() const;
Returns TRUE if at least one CLTU has been processed, false otherwise.

CLTU_Id Get_CltuLastProcessed() const;
Returns the identification of the last CLTU processed.

Precondition: Get_CltusProcessed() returns TRUE.

const ISLE_Time* Get_RadiationStartTime() const;
Returns a pointer to the radiation start time of the last CLTU processed, if the parameter has been set. Otherwise returns a NULL pointer.

Precondition: Get_CltusProcessed() returns TRUE.

CLTU_Status Get_CltuStatus() const;
Returns the status of the last CLTU processed.

Precondition: Get_CltusProcessed() returns TRUE.

bool Get_CltusRadiated() const;
Returns TRUE if at least one CLTU has been radiated, false otherwise.
CLTU_Id Get_CltuLastOk() const;

Returns the identification of the last CLTU successfully radiated.

**Precondition:** Get_CltusRadiated() returns TRUE.

const ISLE_Time* Get_RadiationStopTime() const;

Returns a pointer to the radiation stop time of the last CLTU radiated, if the parameter has been set. Otherwise returns a NULL pointer.

**Precondition:** Get_CltusRadiated() returns TRUE.

CLTU_ProductionStatus Get_ProductionStatus() const;

Returns the current value of the production status.

CLTU_UplinkStatus Get_UplinkStatus() const;

Returns the current value of the uplink status.

void Set_NotificationType( CLTU_NotificationType notifyType );

Sets the notification type.

void Set_EventThrownId( CLTU_EventInvocationId id );

Sets the identification of the thrown event to which the notification refers.

void Set_CltuId( CLTU_Id id );

Sets the identification of the CLTU for which the notification is sent.

void Set_CltuLastProcessed( CLTU_Id id );

Sets the identification of the last CLTU processed and sets ‘CLTUs processed’ to TRUE.

void Set_RadiationStartTime( const ISLE_Time& startTime );

Sets the radiation start time of the last processed CLTU to a copy of the input argument.

void Put_RadiationStartTime( ISLE_Time* pstartTime );

Stores the input argument to the parameter radiation start time of the CLTU last processed.

void Set_CltuStatus( CLTU_Status status );
Sets the status of the last processed CLTU.

```c
void Set_CltuLastOk( CLTU_Id id );
```

Sets the identification of the last CLTU radiated and sets ‘CLTUs radiated to TRUE.

```c
void Set_RadiationStopTime( const ISLE_Time& stopTime );
```

Sets the radiation stop time of the last radiated CLTU to a copy of the input argument.

```c
void Put_RadiationStopTime( ISLE_Time* pstopTime );
```

Stores the input argument to the parameter radiation stop time of the CLTU last radiated.

```c
void Set_ProductionStatus( CLTU_ProductionStatus status );
```

Sets the value of the parameter production status.

```c
void Set_UplinkStatus( CLTU_UplinkStatus status );
```

Sets the value of the parameter uplink status.

### Initial Values of Operation Parameters after Creation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
<th>Created by Service Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>notification type</td>
<td>'invalid'</td>
<td>'invalid'</td>
</tr>
<tr>
<td>event thrown identifier</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLTUs processed</td>
<td>FALSE</td>
<td>TRUE if the number of CLTUs processed is &gt; 0, FALSE otherwise</td>
</tr>
<tr>
<td>CLTU identification last</td>
<td>0</td>
<td>value stored for status reports</td>
</tr>
<tr>
<td>processed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>radiation start time</td>
<td>NULL (not used)</td>
<td>value stored for status reports</td>
</tr>
<tr>
<td>CLTU status</td>
<td>'invalid'</td>
<td>value stored for status reports</td>
</tr>
<tr>
<td>CLTUs radiated</td>
<td>FALSE</td>
<td>TRUE if the number of CLTUs radiated is &gt; 0, FALSE otherwise</td>
</tr>
<tr>
<td>CLTU identification last</td>
<td>0</td>
<td>value stored for status reports</td>
</tr>
<tr>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>radiation stop time</td>
<td>NULL (not used)</td>
<td>value stored for status reports</td>
</tr>
<tr>
<td>production status</td>
<td>'invalid'</td>
<td>value stored for status reports</td>
</tr>
<tr>
<td>uplink status</td>
<td>'invalid'</td>
<td>value stored for status reports</td>
</tr>
</tbody>
</table>
Checking of Invocation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>notification type</td>
<td>Must not be 'invalid'.</td>
</tr>
<tr>
<td>CLTUs processed</td>
<td>Must not be FALSE if the notification type is 'cltu radiated', 'sldu expired', or 'production interrupted'. Must not be FALSE if CLTUs radiated is TRUE.</td>
</tr>
<tr>
<td>radiation start time</td>
<td>Must not be NULL if 'CLTUs processed' is TRUE AND 'cltu status' is one of 'radiation started', 'radiated', or 'interrupted'</td>
</tr>
<tr>
<td>CLTU status</td>
<td>Must not be 'invalid' if 'CLTUs processed' is TRUE</td>
</tr>
<tr>
<td>CLTUs radiated</td>
<td>Must not be FALSE if the notification type is 'cltu radiated'.</td>
</tr>
<tr>
<td>radiation stop time</td>
<td>Must not be NULL if 'CLTUs radiated' is TRUE</td>
</tr>
<tr>
<td>production status</td>
<td>Must not be 'invalid'.</td>
</tr>
<tr>
<td>uplink status</td>
<td>Must not be 'invalid'</td>
</tr>
</tbody>
</table>
A3.4 CLTU STATUS REPORT OPERATION

Name ICLTU_StatusReport
GUID {8f6a1c4c-097e-11d3-bf5c-80954a16aa77}
Inheritance: IUnknown – ISLE_Operation
File ICLTU_StatusReport.H

The interface provides access to the parameters of the unconfirmed operation CLTU-STATUS-REPORT.

Synopsis
#include <CLTU_Types.h>
#include <ISLE_Operation.H>
interface ISLE_Time;

#define IID_ICLTU_StatusReport_DEF { 0x8f6a1c4c, 0x097e, 0x11d3, \ 
    { 0xbf, 0x5c, 0x80, 0x95, 0x4a, 0x16, 0xaa, 0x77 } }

interface ICLTU_StatusReport : ISLE_Operation
{
    virtual CLTU_Id
        Get_CltuLastProcessed() const = 0;
    virtual const ISLE_Time*
        Get_RadiationStartTime() const = 0;
    virtual CLTU_Status
        Get_CltuStatus() const = 0;
    virtual CLTU_Id
        Get_CltuLastOk() const = 0;
    virtual const ISLE_Time*
        Get_RadiationStopTime() const = 0;
    virtual CLTU_ProductionStatus
        Get_ProductionStatus() const = 0;
    virtual CLTU_UplinkStatus
        Get_UplinkStatus() const = 0;
    virtual unsigned long
        Get_NumberOfCltusReceived() const = 0;
    virtual unsigned long
        Get_NumberOfCltusProcessed() const = 0;
    virtual unsigned long
        Get_NumberOfCltusRadiated() const = 0;
    virtual CLTU_BufferSize
        Get_CltuBufferAvailable() const = 0;
    virtual void
        Set_CltuLastProcessed( CLTU_Id id ) = 0;
    virtual void
        Set_RadiationStartTime( const ISLE_Time& startTime ) = 0;
    virtual void
        Put_RadiationStartTime( ISLE_Time* pstartTime ) = 0;
    virtual void
        Set_CltuStatus( CLTU_Status status ) = 0;
    virtual void
        Set_CltuLastOk( CLTU_Id id ) = 0;
    virtual void
        Set_RadiationStopTime( const ISLE_Time& stopTime ) = 0;
    virtual void
        Put_RadiationStopTime( ISLE_Time* pstopTime ) = 0;
    virtual void
        Set_ProductionStatus( CLTU_ProductionStatus status ) = 0;
}
virtual void
    Set_UplinkStatus( CLTU_UplinkStatus status ) = 0;
virtual void
    Set_NumberOfCltusReceived( unsigned long numRecv ) = 0;
virtual void
    Set_NumberOfCltusProcessed( unsigned long numProc ) = 0;
virtual void
    Set_NumberOfCltusRadiated( unsigned long numRad ) = 0;
virtual void
    Set_CltuBufferAvailable( CLTU_BufferSize size ) = 0;
};

Methods

CLTU_Id Get_CltuLastProcessed() const;

Returns the identification of the CLTU last processed.

_Precondition:_ the number of CLTUs processed is not zero.

const ISLE_Time* Get_RadiationStartTime() const;

Returns a pointer to the radiation start time of the last CLTU processed, if the parameter has been set. Otherwise returns a NULL pointer.

_Precondition:_ the number of CLTUs processed is not zero and the CLTU status is neither ‘expired’ nor ‘radiation not started’.

CLTU_Status Get_CltuStatus() const;

Returns the status of the CLTU last processed.

_Precondition:_ the number of CLTUs processed is not zero.

CLTU_Id Get_CltuLastOk() const;

Returns the identification of the CLTU last radiated.

_Precondition:_ the number of CLTUs radiated is not zero.

const ISLE_Time* Get_RadiationStopTime() const;

Returns a pointer to the radiation stop time of the CLTU last radiated, if the parameter has been set. Otherwise returns a NULL pointer.

_Precondition:_ the number of CLTUs radiated is not zero.

CLTU_ProductionStatus Get_ProductionStatus() const;
Returns the current value of the production status.

```cpp
CLTU_UplinkStatus Get_UplinkStatus() const;
```

Returns the current value of the up-link status.

```cpp
unsigned long Get_NumberOfCltusReceived() const;
```

Returns the number of CLTUs that have been received and accepted by the provider.

```cpp
unsigned long Get_NumberOfCltusProcessed() const;
```

Returns the number of CLTUs that have been processed by the provider.

```cpp
unsigned long Get_NumberOfCltusRadiated() const;
```

Returns the number of CLTUs that have been successfully radiated by the provider.

```cpp
CLTU_BufferSize Get_CltuBufferAvailable() const;
```

Returns the size of the available CLTU buffer.

```cpp
void Set_RadiationStartTime( const ISLE_Time& startTime );
```

Sets the radiation start time of the CLTU last processed to a copy of the input argument.

```cpp
void Put_RadiationStartTime( ISLE_Time* pstartTime );
```

Stores the input argument to the parameter radiation start time.

```cpp
void Set_CltuStatus( CLTU_Status status );
```

Sets the status of the CLTU last processed.

```cpp
void Set_CltuLastOk( CLTU_Id id );
```

Sets the identification of the CLTU last radiated.

```cpp
void Set_RadiationStopTime( const ISLE_Time& stopTime );
```

Sets the radiation stop time of the CLTU last radiated to a copy of the input argument.

```cpp
void Put_RadiationStopTime( ISLE_Time* pstopTime );
```

Stores the input argument to the parameter radiation stop time.
void Set_ProductionStatus( CLTU_ProductionStatus status );

Sets the value of the production status.

void Set_UplinkStatus( CLTU_UplinkStatus status );

Sets the value of the up-link status.

void Set_NumberOfCltusReceived( unsigned long numRecv );

Sets the number of CLTUs received and accepted by the provider.

void Set_NumberOfCltusProcessed( unsigned long numProc );

Sets the number of CLTUs processed by the provider.

void Set_NumberOfCltusRadiated( unsigned long numRad );

Sets the number of CLTUs successfully radiated by the provider.

void Set_CltuBufferAvailable( CLTU_BufferSize size );

Sets the available buffer size.

Initial Values of Operation Parameters after Creation

The interface ISLE_SIOPFactory does not support creation of status report operation objects, as this operation is handled by the service instance internally.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLTU identification last processed</td>
<td>0</td>
</tr>
<tr>
<td>radiation start time</td>
<td>NULL (not used)</td>
</tr>
<tr>
<td>CLTU status</td>
<td>'invalid'</td>
</tr>
<tr>
<td>CLTU identification last OK</td>
<td>0</td>
</tr>
<tr>
<td>radiation stop time</td>
<td>NULL (not used)</td>
</tr>
<tr>
<td>production status</td>
<td>'invalid'</td>
</tr>
<tr>
<td>up-link status</td>
<td>'invalid'</td>
</tr>
<tr>
<td>number of CLTUs received</td>
<td>0</td>
</tr>
<tr>
<td>number of CLTUs processed</td>
<td>0</td>
</tr>
</tbody>
</table>
### Parameter Created directly

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of CLTUs radiated</td>
<td>0</td>
</tr>
<tr>
<td>CLTU buffer available</td>
<td>0</td>
</tr>
</tbody>
</table>

### Checking of Invocation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>radiation start time</td>
<td>must not be NULL if number of CLTUs processed &gt; 0 AND CLTU status is one of 'radiation started', 'radiated', or 'interrupted'</td>
</tr>
<tr>
<td>CLTU status</td>
<td>must not be 'invalid' if number of CLTUs processed &gt; 0</td>
</tr>
<tr>
<td>radiation stop time</td>
<td>must not be NULL if number of CLTUs radiated &gt; 0</td>
</tr>
<tr>
<td>production status</td>
<td>must not be 'invalid'</td>
</tr>
<tr>
<td>uplink status</td>
<td>must not be 'invalid'</td>
</tr>
<tr>
<td>number of CLTUs received</td>
<td>Must be ≥ number of CLTUs processed</td>
</tr>
<tr>
<td>number of CLTUs processed</td>
<td>Must be ≥ number of CLTUs radiated and ≤ number of CLTUs received</td>
</tr>
<tr>
<td>number of CLTUs radiated</td>
<td>Must be ≤ number of CLTUs processed</td>
</tr>
</tbody>
</table>
A3.5 CLTU GET PARAMETER OPERATION

Name ICLTU_GetParameter
GUID {F8CB36FF-14A9-4cca-8695-D0AE668FE200}
Inheritance: IUnknown – ISLE_Operation – ISLE_ConfirmedOperation
File ICLTU_GetParameter.H

The interface provides access to the parameters of the confirmed operation
CLTU-GET-PARAMETER.

Synopsis

#include <CLTU_Types.h>
#include <ISLE_ConfirmedOperation.H>

#define IID_ICLTU_GetParameter_DEF { 0xf8cb36ff, 0x14a9, 0x4cca, 
{ 0x86, 0x95, 0xd0, 0xae, 0x66, 0x8f, 0xe2, 0x0 } }

interface ICLTU_GetParameter : ISLE_ConfirmedOperation
{
    virtual CLTU_ParameterName Get_RequestedParameter() const = 0;
    virtual CLTU_ParameterName Get_ReturnedParameter() const = 0;
    virtual SLE_YesNo Get_BitLockRequired() const = 0;
    virtual SLE_DeliveryMode Get_DeliveryMode() const = 0;
    virtual CLTU_Id Get_ExpectedCltuId() const = 0;
    virtual CLTU_EventInvocationId Get_ExpectedEventInvocationId() const = 0;
    virtual unsigned long Get_MaximumSlduLength() const = 0;
    virtual CLTU_ModulationFrequency Get_ModulationFrequency() const = 0;
    virtual CLTU_ModulationIndex Get_ModulationIndex() const = 0;
    virtual CLTU_PlopInEffect Get_PlopInEffect() const = 0;
    virtual unsigned long Get_ReportingCycle() const = 0;
    virtual unsigned long Get_ReturnTimeoutPeriod() const = 0;
    virtual SLE_YesNo Get_RfAvailableRequired() const = 0;
    virtual CLTU_SubcarrierDivisor Get_SubcarrierToBitRateRatio() const = 0;
    virtual void Set_RequestedParameter( CLTU_ParameterName name ) = 0;
    virtual void Set_BitLockRequired( SLE_YesNo yesno ) = 0;
    virtual void Set_DeliveryMode() = 0;
    virtual void Set_ExpectedCltuId( CLTU_Id id ) = 0;
    virtual void
Methods

CLTU_ParameterName Get_RequestedParameter() const;

Returns the parameter for which the value shall be reported.

CLTU_ParameterName Get_ReturnedParameter() const;

Returns the parameter for which the value is reported. Following the return, this must be identical to the result of Get_RequestedParameter().

SLE_YesNo Get_BitLockRequired() const;

Returns the value of the parameter bit-lock-required.

Precondition: the returned parameter is bit-lock-required.

SLE_DeliveryMode Get_DeliveryMode() const;

Returns the delivery-mode.

Precondition: the returned parameter is delivery-mode.

CLTU_Id Get_ExpectedCltuId() const;

Returns the next expected CLTU identification.

Precondition: the returned parameter is ‘expected SLDU identification’ and the value has been set via a START invocation or as result of a TRANSFER DATA operation.
CLTU_EventInvocationId Get_ExpectedEventInvocationId() const;
Returns the next expected event invocation identifier.

**Precondition:** the returned parameter is expected-event-invocation-id.

unsigned long Get_MaximumSlduLength() const;
Returns the maximum length in bytes of a CLTU supported by the provider.

**Precondition:** the returned parameter is maximum-SLDU-length.

CLTU_ModulationFrequency Get_ModulationFrequency() const;
Returns the modulation frequency measured in Hz.

**Precondition:** the returned parameter is modulation-frequency.

CLTU_ModulationIndex Get_ModulationIndex() const;
Returns the modulation index used by the provider.

**Precondition:** the returned parameter is modulation-index.

CLTU_PlopInEffect Get_PlopInEffect() const;
Returns the PLOP used by the provider.

**Precondition:** the returned parameter is PLOP-in-effect.

unsigned long GetReportingCycle() const;
Returns the reporting cycle requested by the user if periodic reporting is active. If reporting is not active, returns zero.

**Precondition:** the returned parameter is reporting-cycle.

unsigned long Get_ReturnTimeoutPeriod() const;
Returns the return timeout period used by the provider.

**Precondition:** the returned parameter is return-timeout-period.

SLE_YesNo Get_RfAvailableRequired() const;
Returns the value of the parameter rf-available-required.
Precondition: the returned parameter is rf-available-required.

CLTU_SubcarrierDivisor Get_SubcarrierToBitRateRatio() const;

Returns the value of the parameter subcarrier-to-bit-rate-ratio.

Precondition: the returned parameter is subcarrier-to-bit-rate-ratio.

CLTU_GetParameterDiagnostic Get_GetParameterDiagnostic() const;

Returns the diagnostic code.

Precondition: the result is negative, and the diagnostic type is set to ‘specific’.

void Set_RequestedParameter( CLTU_ParameterName name );

Sets the parameter for which the provider shall report the value.

void Set_BitLockRequired( SLE_YesNo yesno );

Sets the returned parameter name to bit-lock-required and sets its value as defined by the argument.

void Set_DeliveryMode();

Sets the returned parameter name to delivery-mode and sets its value to ‘fwd online’.

void Set_ExpectedCltuId( CLTU_Id id );

Sets the returned parameter name to expected-SLDU-identification and sets its value as defined by the argument.

void Set_ExpectedEventInvocationId( CLTU_EventInvocationId id );

Sets the returned parameter name to expected-event-invocation-id and sets its value as defined by the argument.

void Set_MaximumSlduLength( unsigned int length );

Sets the returned parameter name to maximum-SLDU-length and sets its value as defined by the argument.

void Set_ModulationFrequency( CLTU_ModulationFrequency frequency );
Sets the returned parameter name to `modulation-frequency` and sets its value as defined by the argument.

```c
void Set_ModulationIndex( CLTU_ModulationIndex index );
```

Sets the returned parameter name to `modulation-index` and sets its value as defined by the argument.

```c
void Set_PlopInEffect( CLTU_PlopInEffect plop );
```

Sets the returned parameter name to `PLOP-in-effect` and sets its value as defined by the argument.

```c
void Set_ReportingCycle( unsigned long cycle );
```

Sets the returned parameter name to `reporting-cycle` and sets its value as defined by the argument.

```c
void Set_ReturnTimeoutPeriod( unsigned long period );
```

Sets the returned parameter name to `return-timeout-period` and sets its value as defined by the argument.

```c
void Set_RfAvailableRequired( SLE_YesNo yesno );
```

Sets the returned parameter name to `rf-available-required` and sets its value as defined by the argument.

```c
void Set_SubcarrierToBitRateRatio( CLTU_SubcarrierDivisor divisor );
```

Sets the returned parameter name to `subcarrier-to-bit-rate-ratio` and sets its value as defined by the argument.

```c
void Set_GetParameterDiagnostic( CLTU_GetParameterDiagnostic diagnostic );
```

Sets the result to 'negative', the diagnostic type to 'specific', and stores the value of the diagnostic code passed by the argument.

**Initial Values of Operation Parameters after Creation**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
<th>Created by Service Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>requested parameter</td>
<td>'invalid'</td>
<td>'invalid'</td>
</tr>
<tr>
<td>returned parameter</td>
<td>'invalid'</td>
<td>'invalid'</td>
</tr>
</tbody>
</table>
Checking of Invocation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>requested parameter</td>
<td>must not be ‘invalid’</td>
</tr>
</tbody>
</table>

Checking of Return Parameters

The interface ensures consistency between the returned parameter name and the parameter value, as the client cannot set the returned parameter name Therefore, this consistency need not be checked on the provider side. The checks defined below only need to be performed when the return is received by the service user.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returned parameter</td>
<td>must be the same as the requested parameter</td>
</tr>
<tr>
<td>bit lock required</td>
<td>must not be ‘invalid’ if the returned parameter is ‘bit lock required’</td>
</tr>
<tr>
<td>delivery mode</td>
<td>must be ‘fwd online’ if the returned parameter is ‘delivery mode’</td>
</tr>
<tr>
<td>maximum SLDU length</td>
<td>must not be 0 if the returned parameter is ‘maximum SLDU length’</td>
</tr>
<tr>
<td>modulation index</td>
<td>must not be 0 if the returned parameter is ‘modulation index’</td>
</tr>
<tr>
<td>PLOP in effect</td>
<td>must not be ‘invalid’ if the returned parameter is ‘PLOP in effect’</td>
</tr>
<tr>
<td>Parameter</td>
<td>Required condition</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>return timeout period</td>
<td>must not be 0 if the returned parameter is 'return timeout period'</td>
</tr>
<tr>
<td>RF available required</td>
<td>must not be 'invalid' if the returned parameter is 'RF available required'</td>
</tr>
<tr>
<td>sub-carrier to bit-rate ratio</td>
<td>must not be 0 if the returned parameter is 'sub-carrier to bit-rate ratio'</td>
</tr>
<tr>
<td>GET PARAMETER diagnostic</td>
<td>must not be 'invalid' if the result is 'negative' and the diagnostic type is 'specific'</td>
</tr>
</tbody>
</table>
A3.6 CLTU THROW EVENT OPERATION

Name: ICLTU_ThrowEvent
GUID: {5505B552-39D6-44df-B304-6BDFE0A141EE}
Inheritance: IUnknown – ISLE_Operation – ISLE_ConfirmedOperation
File: ICLTU_ThrowEvent.H

The interface provides access to the parameters of the confirmed operation CLTU-THROW-EVENT.

Synopsis
#include <CLTU_Types.h>
#include <ISLE_ConfirmedOperation.H>

#define IID_ICLTU_ThrowEvent_DEF { 0x5505b552, 0x39d6, 0x44df, 
    { 0xb3, 0x4, 0x6b, 0xdf, 0xe0, 0xa1, 0x41, 0xee } }

interface ICLTU_ThrowEvent : ISLE_ConfirmedOperation
{
    virtual unsigned short
        Get_EventId() const = 0;
    virtual CLTU_EventInvocationId
        Get_EventInvocationId() const = 0;
    virtual CLTU_EventInvocationId
        Get_ExpectedEventInvocationId() const = 0;
    virtual const SLE_Octet*
        Get_EventQualifier( size_t & length ) const = 0;
    virtual CLTU_ThrowEventDiagnostic
        Get_ThrowEventDiagnostic() const = 0;
    virtual void
        Set_EventId( unsigned short id ) = 0;
    virtual void
        Set_EventInvocationId( CLTU_EventInvocationId id ) = 0;
    virtual void
        Set_ExpectedEventInvocationId( CLTU_EventInvocationId id ) = 0;
    virtual void
        Set_EventQualifier( size_t length, const SLE_Octet* pdata ) = 0;
    virtual void
        Set_ThrowEventDiagnostic( CLTU_ThrowEventDiagnostic diagnostic ) = 0;
};

Methods

unsigned short Get_EventId() const;

Returns the identification of the event.

CLTU_EventInvocationId Get_EventInvocationId() const;

Returns the invocation identifier of the event.

CLTU_EventInvocationId Get_ExpectedEventInvocationId() const;
Returns the next expected invocation identifier of the event in the return.

```cpp
const SLE_Octet* Get_EventQualifier( size_t& length ) const;
```

Returns a pointer to the event qualifier in the object. The data must neither be modified nor deleted by the caller.

**Arguments**

- `length` the number of bytes of the event qualifier

```cpp
CLTU_ThrowEventDiagnostic Get_ThrowEventDiagnostic() const;
```

Returns the diagnostic code.

**Precondition:** the result is negative, and the diagnostic type is set to ‘specific’.

```cpp
void Set_EventId( unsigned short id );
```

Sets the identifier of the event.

```cpp
void Set_EventInvocationId( CLTU_EventInvocationId id );
```

Sets the invocation identifier for the event in the invocation.

```cpp
void Set_ExpectedEventInvocationId( CLTU_EventInvocationId id );
```

Sets the next expected invocation identifier for the event in the return.

```cpp
void Set_EventQualifier( size_t length, const SLE_Octet* pdata );
```

Copies `length` bytes from the address `pdata` to the internal event qualifier parameter.

**Arguments**

- `pdata` pointer to the event qualifier
- `length` the number of bytes of the event qualifier

```cpp
void Set_ThrowEventDiagnostic( CLTU_ThrowEventDiagnostic diagnostic);
```

Sets the result to ‘negative’, the diagnostic type to ‘specific’, and stores the value of the diagnostic code passed by the argument.

**Initial Values of Operation Parameters after Creation**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
<th>Created by Service Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>event identifier</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

PRE-RELEASE
DRAFT CCSDS RECOMMENDED PRACTICE FOR A SLE CLTU SERVICE API

PAGE A-32

September 2005
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Created directly</th>
<th>Created by Service Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>event invocation identifier</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>expected event invocation id</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>event qualifier</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>THROW EVENT diagnostic</td>
<td>‘invalid’</td>
<td>‘invalid’</td>
</tr>
</tbody>
</table>

### Checking of Invocation Parameters

No checks are performed beyond those defined by the inherited interfaces.

### Checking of Return Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>THROW EVENT diagnostic</td>
<td>must not be ‘invalid’ if the result is ‘negative’ and the diagnostic type is ‘specific’</td>
</tr>
<tr>
<td>expected event invocation id</td>
<td>If result is ‘positive’, must be event invocation id + 1</td>
</tr>
</tbody>
</table>
A4 CLTU SERVICE INSTANCE INTERFACES

A4.1 SERVICE INSTANCE CONFIGURATION

Name ICLTU_SIAdmin
GUID {4A508916-3D5B-4c8d-ABD4-EC6547D51320}
Inheritance: IUnknown
File ICLTU_SIAdmin.H

The interface provides write and read access to the CLTU-specific service instance configuration-parameters. All configuration parameters must be set as part of service instance configuration. When the method ConfigCompleted() is called on the interface ISLE_SIAdmin, the service element checks that all parameters have been set and returns an error when the configuration is not complete.

CLTU specific configuration parameters are not processed or modified by the API. They are only used for the following purposes:

1) to inform the service user via the GET-PARAMETER operation;

for checking of operation parameters.

CLTU configuration parameters can be modified at any time. The API always uses the last value set in GET-PARAMETER returns. Parameters used for initialization of the status report must not be set after invocation of ConfigCompleted(). The effect of invoking these methods at a later stage is undefined.

It is noted that service management might constrain the range of parameters that can be modified after configuration. These constraints are not enforced by the API.

As a convenience for the application, the interface provides read access to the configuration parameters, except for parameters used to initialise the status report. If retrieval methods are called before configuration, the value returned is undefined.

Synopsis

```
#include <CLTU_Types.h>
#include <SLE_SCM.H>

#define IID_ICLTU_SIAdmin_DEF { 0x4a508916, 0x3d5b, 0x4c8d, \
     { 0xab, 0xd4, 0xec, 0x65, 0x47, 0xd5, 0x13, 0x20 } }

@interface ICLTU_SIAdmin : IUnknown
{
    virtual void
    Set_BitLockRequired( SLE_YesNo yesno ) = 0;
    virtual void
    Set_MaximumSlduLength( unsigned long length ) = 0;
}
```
virtual void
    Set_ModulationFrequency( CLTU_ModulationFrequency frequency ) = 0;
virtual void
    Set_ModulationIndex( CLTU_ModulationIndex index ) = 0;
virtual void
    Set_PlopInEffect( CLTU_PlopInEffect plop) = 0;
virtual void
    Set_RfAvailableRequired( SLE_YesNo yesno ) = 0;
virtual void
    Set_SubcarrierToBitRateRatio( CLTU_SubcarrierDivisor divisor ) = 0;
virtual void
    Set_MaximumBufferSize( CLTU_BufferSize size ) = 0;
virtual void
    Set_InitialProductionStatus( CLTU_ProductionStatus status ) = 0;
virtual void
    Set_InitialUplinkStatus( CLTU_UplinkStatus status ) = 0;
virtual void
    Set_NotificationMode( CLTU_NotificationMode mode ) = 0;
virtual SLE_YesNo
    Get_BitLockRequired() const = 0;
virtual unsigned long
    Get_MaximumSlduLength() const = 0;
virtual CLTU_ModulationFrequency
    Get_ModulationFrequency() const = 0;
virtual CLTU_ModulationIndex
    Get_ModulationIndex() const = 0;
virtual CLTU_PlopInEffect
    Get_PlopInEffect() const = 0;
virtual SLE_YesNo
    Get_RfAvailableRequired() const = 0;
virtual CLTU_SubcarrierDivisor
    Get_SubcarrierToBitRateRatio() const = 0;
virtual CLTU_BufferSize
    Get_MaximumBufferSize() const = 0;
virtual CLTU_NotificationMode
    Get_NotificationMode() const = 0;
};

Methods

void Set_BitLockRequired( SLE_YesNo yesno );

Sets the parameter indicating whether bit lock is required to set the production status to operational.

void Set_MaximumSlduLength( unsigned int length );

Sets the maximum size in byte of a CLTU supported by the provider.

void Set_ModulationFrequency( CLTU_ModulationFrequency frequency );

Sets the value of the configuration parameter modulation-frequency.

void Set_ModulationIndex( CLTU_ModulationIndex index );

Sets the modulation index used by the provider.
void Set_PlopInEffect( CLTU_PlopInEffect plop );

Sets the parameter indicating whether PLOP-1 or PLOP-2 is used.

void Set_RfAvailableRequired( SLE_YesNo yesno );

Sets the parameter indicating whether RF lock is required to set the production status to operational.

void Set_SubcarrierToBitRateRatio( CLTU_SubcarrierDivisor divisor );

Sets the parameter subcarrier-to-bit-rate-ratio.

void Set_MaximumBufferSize( CLTU_BufferSize size );

Sets the maximum size in byte of the CLTU buffer supported by the provider. The API uses this value as the initial value for the available buffer size.

void Set_InitialProductionStatus( CLTU_ProductionStatus status );

Sets the production status at the time the service instance is configured.

Precondition: The method ISLE_SIAdmin::ConfigCompleted() has not been invoked yet.

void Set_InitialUplinkStatus( CLTU_UplinkStatus status );

Sets the up-link status at the time the service instance is configured.

Precondition: The method ISLE_SIAdmin::ConfigCompleted() has not been invoked yet.

void Set_NotificationMode( CLTU_NotificationMode mode );

Sets the value of the parameter indicating whether the SLE API shall operate in ‘immediate’ or ‘deferred’ notification mode. When set to ‘immediate’, the SLE API immediately notifies the SLE user when the production status changes to ‘interrupted’. If the API operates in ‘deferred’ mode and no CLTU is affected and the production status changes to ‘interrupted’, the notification is deferred until the attempt is made to radiate the next CLTU.

SLE_YesNo Get_BitLockRequired() const;

Returns the value of the parameter indicating whether bit lock is required to set the production status to operational.

unsigned long Get_MaximumSlduLength() const;
Returns the maximum length of a CLTU.

`CLTU_ModulationFrequency Get_ModulationFrequency() const;`

Returns the value of the parameter modulation-frequency.

`CLTU_ModulationIndex Get_ModulationIndex() const;`

Returns the value of the parameter modulation index.

`CLTU_PlopInEffect Get_PlopInEffect() const;`

Returns the value of the parameter PLOP in effect.

`SLE_YesNo Get_RfAvailableRequired() const;`

Returns the value of the parameter indicating whether RF lock is required to set the production status to operational.

`CLTU_SubcarrierDivisor Get_SubcarrierToBitRateRatio() const;`

Returns the value of the parameter subcarrier-to-bit-rate-ratio.

`CLTU_BufferSize Get_MaximumBufferSize() const;`

Returns the value of the parameter maximum CLTU buffer size.

`CLTU_NotificationMode Get_NotificationMode() const;`

Returns the value of the parameter indicating if ‘immediate’ or ‘deferred’ notification is in effect.
A4.2 UPDATE OF SERVICE INSTANCE PARAMETERS

Name ICLTU_SIUpdate
GUID \{F104EF90-A2BE-413d-B0BA-CEB4C790D4DD\}
Inheritance: IUnknown
File ICLTU_SIUpdate.H

The interface provides methods to update parameters that shall be reported to the service user via the operation STATUS-REPORT. In order to keep this information up to date the appropriate methods of this interface must be called whenever certain events occur (see the specification in 3.1). If these events must be reported to the CLTU service user via a notification, the API can be requested to send the notification. Alternatively the application can generate and send the notification itself.

The methods of this interface must always be called when one of the relevant events occurs, independent of the state of the service instance. Notifications to the user will only be sent, if the service instance state is either ‘ready’ or ‘active’. Failure to inform the API of an event can result in incorrect and inconsistent parameters in the status report.

Because of performance considerations, methods processing nominal events perform no plausibility checks, but completely rely on the application to provide correct and consistent arguments.

The interface provides read access to the parameters set via this interface and to parameters accumulated or derived by the API according to the specifications in 3.1. The retrievable parameters include ‘expected CLTU identification’ and ‘expected event invocation id’. These parameters are not included in the status report but can be read by service user via the operation CLTU–GET–PARAMETER. The API sets the parameters to the initial values specified at the end of this section when the service instance is configured. Parameter values retrieved before configuration are undefined.

Synopsis

```c
#include <CLTU_Types.h>
#include <SLE_SCM.H>
interface ISLE_Time;

#define IID_ICLTU_SIUpdate_DEF {0xf104ef90, 0xa2be, 0x413d, 
    { 0xb0, 0xba, 0xce, 0xb4, 0xc7, 0x90, 0xd4, 0xdd } }

interface ICLTU_SIUpdate : IUnknown
{
    virtual void CltuStarted( CLTU_Id id, 
                              const ISLE_Time& radiationStartTime, 
                              CLTU_BufferSize bufferAvailable ) = 0;

    virtual void CltuRadiated( const ISLE_Time& radiationStopTime, 
                               const ISLE_Time* radiationStartTime, 
                               bool notify ) = 0;

    virtual HRESULT
```
CltuNotStarted( CLTU_Id id,
    CLTU_Failure reason,
    CLTU_BufferSize bufferAvailable,
    bool notify ) = 0;

virtual HRESULT
    ProductionStatusChange( CLTU_ProductionStatus newStatus,
                            CLTU_BufferSize bufferAvailable,
                            bool notify ) = 0;

virtual void
    BufferEmpty(bool notify ) = 0;

virtual void
    EventProcCompleted( CLTU_EventInvocationId id,
                        CLTU_NotificationType result ) const = 0;

virtual void
    Set_UplinkStatus( CLTU_UplinkStatus status ) = 0;

virtual CLTU_ProductionStatus
    Get_ProductionStatus() const = 0;

virtual CLTU_BufferSize
    Get_CltuBufferAvailable() const = 0;

virtual unsigned long
    Get_NumberOfCltusReceived() const = 0;

virtual unsigned long
    Get_NumberOfCltusProcessed() const = 0;

virtual unsigned long
    Get_NumberOfCltusRadiated() const = 0;

virtual CLTU_Id
    Get_CltuLastProcessed() const = 0;

virtual const ISLE_Time*
    Get_RadiationStartTime() const = 0;

virtual CLTU_Status
    Get_CltuStatus() const = 0;

virtual CLTU_Id
    Get_CltuLastOk() const = 0;

virtual const ISLE_Time*
    Get_RadiationStopTime() const = 0;

virtual CLTU_UplinkStatus
    Get_UplinkStatus() const = 0;

virtual CLTU_Id
    Get_ExpectedCltuId() const = 0;

virtual CLTU_EventInvocationId
    Get_ExpectedEventInvocationId() const = 0;
};

Methods

void CltuStarted( CLTU_Id id,
              const ISLE_Time& radiationStartTime,
              CLTU_BufferSize bufferAvailable );

The method must be called when radiation of a CLTU has been started. It performs the following actions:

a) increment the number of CLTUs processed;
b) store the CLTU identification and the radiation start time to the CLTU last processed;
c) set the status of the CLTU last processed to ‘radiation started’;
d) update the available buffer size with the value of the argument passed.
Preconditions:

The client must ensure the following preconditions since they are not checked by the implementation.

a) the state of the service instance must be ‘active’.
b) the production status must be ‘operational’.
c) if the previous CLTU has completed radiation, the method CltuRadiated() must have been called.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>the CLTU identification of the CLTU for which radiation started</td>
</tr>
<tr>
<td>radiationStartTime</td>
<td>the time at which radiation of the CLTU started</td>
</tr>
<tr>
<td>bufferAvailable</td>
<td>the size of the available CLTU buffer at the time of the method call</td>
</tr>
</tbody>
</table>

`void CltuRadiated( const ISLE_Time& radiationStopTime,`  
`const ISLE_Time* radiationStartTime,`  
`bool notify );`

The method must be called when radiation of a CLTU has completed. It performs the following actions:

a) increment the number of CLTUs radiated;
b) set the status of the CLTU last processed to ‘radiated’;
c) copy the identification of the CLTU last processed to the CLTU last OK;
d) store the radiation stop time to the CLTU last OK;
e) if the radiation start time is not NULL, store the radiation start time to the CLTU last processed;
f) if the argument notify is TRUE send the notification ‘radiated’ to the service user provided sending of notifications is allowed according to the state tables in reference [6].

Preconditions:

The client must ensure the following preconditions since they are not checked by the implementation.

a) the production status must be ‘operational’;
b) before the method call, the status of the CLTU last processed must be ‘radiation started’;
c) the radiation stop time must not be earlier than the previously set radiation start time;

d) the argument notify must only be set to TRUE if the service user has requested a notification for the CLTU.

Arguments

| radiationStopTime | the time at which radiation of the CLTU completed |
| radiationStartTime | the exact time at which radiation of the CLTU started, if the time passed with the method CltuStarted() was an estimate, or NULL to confirm the time passed with CltuStarted(). |
| Notify | if TRUE a notification shall be sent to the service user |

HRESULT CltuNotStarted( CLTU_Id id,  
    CLTU_Failure reason,  
    CLTU_BufferSize bufferAvailable,  
    bool notify );

The method must be called when radiation of a CLTU could not be started because the latest radiation time has passed or the production status is interrupted. It performs the following actions:

a) increment the number of CLTUs processed;

b) store the CLTU identification to the CLTU last processed;

c) set the radiation start time of the CLTU last processed to NULL;

d) if the reason is ‘expired’ set the status of the CLTU last processed to ‘expired’;

e) if the reason is ‘production interrupted’, set the status of the CLTU last processed to ‘radiation not started’;

f) update the available buffer size with the value of the argument passed;

g) if the argument notify is TRUE and the reason is ‘expired’ send the notification ‘SLDU expired’ to the service user;

h) if the argument notify is TRUE and the reason is ‘production interrupted’ send the notification ‘production interrupted’ to the service user.

Arguments

| id | the CLTU identification of the CLU for which radiation could not start |
| reason | the reason for the failure (‘expired’ or ‘production interrupted’) |
| bufferAvailable | the size of the available CLTU buffer at the time of the method call |
| notify | if TRUE a notification shall be sent to the service user |
Result codes

S_OK the updates have been made and the notification sent if requested

SLE_E_INCONSISTENT the reason is ‘production interrupted’ but the production status is not ‘interrupted’; OR ‘immediate notification’ is in effect and the production status is already ‘interrupted’ (this would imply that the application attempted to radiate a CLTU while the production status was already interrupted).

– updates have not been performed. and no notification has been sent

SLE_E_STATE the service instance state is ‘unbound’ (it might have aborted) – updates have been performed but the requested notification could not be sent.

HRESULT ProductionStatusChange( CLTU_ProductionStatus newStatus,
CLTU_BufferSize bufferAvailable,
bool notify );

The method must be called when the production status changes. It performs the following actions.

a) set the production status to the value of the argument newStatus;

b) update the available buffer size with the value of the argument passed;

c) if the new production status is 'interrupted' or 'halted' and the status of the CLTU last processed is 'radiation started' set the status of the CLTU last processed to 'interrupted';

d) if the argument notify is TRUE the new production status is ‘operational’ and the production status last reported was not 'operational', send the notification ‘production operational’ to the service user, provided sending of notifications is allowed according to the state tables in reference [6];

e) if the argument notify is TRUE and the new production status is ‘halted’ send the notification ‘production halted’ to the service user, provided sending of notifications is allowed according to the state tables in reference [6];

f) if the argument notify is TRUE and the new production status is ‘interrupted’ and ‘immediate notification’ is in effect, send the notification ‘production interrupted’ to the service user, provided sending of notifications is allowed according to the state tables in reference [6];

g) if the argument notify is TRUE and the new production status is ‘interrupted’ and ‘deferred notification’ is in effect and the status of the CLTU last processed was 'radiation started' at the time the method was invoked, send the notification
‘production interrupted’ to the service user, provided sending of notifications is allowed according to the state tables in reference [6].

### Arguments

- **newStatus**: the new value of the production status
- **bufferAvailable**: the size of the available CLTU buffer at the time of the method call
- **notify**: if TRUE a notification shall be sent to the service user

### Result codes

- **S_OK**: the updates have been made; the notification sent if it was requested and the state of service instance allowed transmission
- **SLE_S_IGNORED**: the production status did not change – updates have not been performed and no notification has been sent.

```cpp
void BufferEmpty( bool notify );
```

The method shall be called when the CLTU buffer becomes empty because all CLTUs were processed. It shall not be called when the packet buffer is cleared due to one of the events for which reference [5] requires discarding of buffered CLTUs.

The method performs the following actions:

1. Sets the parameter CLTU buffer available to the maximum CLTU buffer size set by configuration of the service instance.
2. If the argument notify is TRUE, sends the notification ‘buffer empty’ to the service user provided sending of notifications is allowed according to the state tables in reference [6].

### Arguments

- **notify**: if true a notification shall be sent to the service user

```cpp
void EventProcCompleted(CLTU_EventInvocationId id,
                        CLTU_EventResult result,
                        bool notify) const;
```

The method must be called when the application has finished processing of the event identified with the argument ‘id’. It generates and sends a notification to the user, providing the ‘id’ and the notification type supplied with the ‘result’ argument.

### Arguments

- **id**: the event thrown identifier, for which processing is completed
- **result**: the result of event processing, which tells the API which notification to send to the user.
- **notify**: if TRUE a notification shall be sent to the service user; because sending the notification is the only action of the method, this
argument is not really needed – it is provided for consistency with other methods in this interface.

```c
void Set_UplinkStatus( CLTU_UplinkStatus status );
```

Sets the value of the up-link status.

- **Arguments**
  - **status**
    - the new value of the up-link status

```c
CLTU_ProductionStatus Get_ProductionStatus() const;
```

Returns the value of the production status parameter.

```c
CLTU_BufferSize Get_CltuBufferAvailable() const;
```

Returns the value of the available CLTU buffer size. This value is either a copy of the buffer size parameter in the last TRANSFER-DATA return sent by the application, or the value set by one of methods of this interface, if that method was called after the last TRANSFER-DATA return.

```c
unsigned long Get_NumberOfCltusReceived() const;
```

Returns the number of CLTUs received. The API initializes this number to zero and increments it by one for every TRANSFER-DATA return with a positive result.

```c
unsigned long Get_NumberOfCltusProcessed() const;
```

Returns the number of CLTUs for which radiation has been attempted. The API initializes this number to zero and increments it by one for every invocation of the methods CltuStarted() and CltuNotStarted() .

```c
unsigned long Get_NumberOfCltusRadiated() const;
```

Returns the number of CLTUs, which have been radiated. The API initializes this number to zero and increments it by one for every invocation of the method CltuRadiated().

```c
CLTU_Id Get_CltuLastProcessed() const;
```

Returns the CLTU identification passed with the last call to CltuStarted() or CltuNotStarted(). If the number of CLTUs processed is zero, returns the initial value defined in the table below.

```c
const ISLE_Time* Get_RadiationStartTime() const;
```
Returns the radiation start time passed with the last call to CltuStarted() or CltuNotStarted(). If the number of CLTUs processed is zero, the value is undefined. The method returns a NULL pointer in that case.

CLTU_Status Get_CltuStatus() const;

Returns the CLTU status set by the most recent call to CltuStarted(), CltuRadiated(), CltuNotStarted(), or Set_ProductionStatus(). If the number of CLTUs processed is zero, the value is undefined.

CLTU_Id Get_CltuLastOk() const;

Returns the CLTU identification set by the last call to CltuRadiated(). If the number of CLTUs radiated is zero, returns the initial value as defined in the table below.

const ISLE_Time* Get_RadiationStopTime() const;

Returns the radiation stop time passed with the last call to CltuRadiated(). If the number of CLTUs radiated is zero, the value is undefined. The method returns a NULL pointer in that case.

CLTU_UplinkStatus Get_UplinkStatus() const;

Returns the value of the up-link status as initially set via the interface ICLTU_SIAdmin or by the last call to Set_UplinkStatus().

CLTU_Id Get_ExpectedCltuId() const;

Returns the value of the next CLTU identification expected. This value is a copy of the CLTU identification parameter in the last CLTU-TRANSFER-DATA return sent by the application or of the first CLTU identification specified in the CLTU-START invocation.

CLTU_EventInvocationId Get_ExpectedEventInvocationId() const;

Returns the value of the next event invocation identifier expected. This value is a copy of the event invocation identifier parameter in the last THROW-EVENT return sent by the application.

**Initial Parameter Values**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>production status</td>
<td>initial production status set via the interface ICLTU_SIAdmin</td>
</tr>
<tr>
<td>CLTU identification last processed</td>
<td>0</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>radiation start time</td>
<td>NULL pointer</td>
</tr>
<tr>
<td>CLTU status</td>
<td>'invalid'</td>
</tr>
<tr>
<td>CLTU identification last OK</td>
<td>0</td>
</tr>
<tr>
<td>radiation stop time</td>
<td>NULL pointer</td>
</tr>
<tr>
<td>CLTU buffer available</td>
<td>maximum CLTU buffer size set via the interface ICLTU_SIAAdmin</td>
</tr>
<tr>
<td>number of CLTUs received</td>
<td>0</td>
</tr>
<tr>
<td>number of CLTUs processed</td>
<td>0</td>
</tr>
<tr>
<td>number of CLTUs radiated</td>
<td>0</td>
</tr>
<tr>
<td>uplink status</td>
<td>initial uplink status set via the interface ICLTU_SIAAdmin</td>
</tr>
<tr>
<td>expected CLTU identification</td>
<td>0</td>
</tr>
<tr>
<td>expected event invocation id</td>
<td>0</td>
</tr>
</tbody>
</table>
ANNEX B

ACRONYMS

(This annex is not part of the Recommended Practice)

This annex expands the acronyms used throughout this Recommended Practice.

API Application Program Interface
CCSDS Consultative Committee for Space Data Systems
CLTU Command Link Transmission Unit
GUID Globally Unique Identifier
ID Identifier
IEC International Electrotechnical Commission
ISO International Organization for Standardization
OMG Object Management Group
PDU Protocol Data Unit
SI Service Instance
SLE Space Link Extension
UML Unified Modelling Language
ANNEX C

INFORMATIVE REFERENCES

(This annex is not part of the Recommended Practice)


