CCSDS

The Consultative Committee for Space Data Systems

Recommendation for Space Data System Practices

SPACE LINK EXTENSION— APPLICATION PROGRAM INTERFACE FOR RETURN CHANNEL FRAMES SERVICE

RECOMMENDED PRACTICE

CCSDS 915.2-M-1

MAGENTA BOOK

October 2008

The Consultative Committee for Space Data Systems

Recommendation for Space Data System Practices

SPACE LINK EXTENSION— APPLICATION PROGRAM INTERFACE FOR RETURN CHANNEL FRAMES SERVICE

RECOMMENDED PRACTICE

CCSDS 915.2-M-1

MAGENTA BOOK
October 2008

AUTHORITY

Issue:	Recommended Practice, Issue 1
Date:	October 2008
Location:	Washington, DC, USA

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 documents 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 document is published and maintained by:

> CCSDS Secretariat
> Space Communications and Navigation Office, 7L70
> Space Operations Mission Directorate
> NASA Headquarters
> Washington, DC 20546-0001, USA

STATEMENT OF INTENT

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of its members. 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 in themselves considered binding on any Agency.

CCSDS Recommendations take two forms: **Recommended Standards** that are prescriptive and are the formal vehicles by which CCSDS Agencies create the standards that specify how elements of their space mission support infrastructure shall operate and interoperate with others; and **Recommended Practices** that are more descriptive in nature and are intended to provide general guidance about how to approach a particular problem associated with space mission support. This **Recommended Practice** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommended Practice** is entirely voluntary and does not imply a commitment by any Agency or organization to implement its recommendations in a prescriptive sense.

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 **Recommended Practice** is issued, existing CCSDS-related member Practices and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each member to determine when such Practices or implementations are to be modified. Each member is, however, strongly encouraged to direct planning for its new Practices and implementations towards the later version of the Recommended Practice.

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 [3]). 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 of this document may occur. This Recommended Practice is therefore subject to CCSDS document management and change control procedures, which are 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.
- China National Space Administration (CNSA)/People's Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- Centro Tecnico Aeroespacial (CTA)/Brazil.
- Chinese Academy of Sciences (CAS)/China.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish National Space Center (DNSC)/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.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Korea Aerospace Research Institute (KARI)/Korea.
- MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- Ministry of Communications (MOC)/Israel.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 915.2-M-1	Space Link Extension—Application Program Interface for Return Channel Frames Service, Recommended Practice, Issue 1	October 2008	Original issue

CONTENTS

1 INTRODUCTION

1.1 PURPOSE

The Recommended Practice *Space Link Extension—Application Program Interface for Transfer Services—Core Specification* (reference [3]) 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 [3] 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 Return Channel Frames (RCF) service defined in reference [2].

1.2 SCOPE

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

   a) the RCF-specific functionality provided by API components;

   b) the RCF-specific interfaces provided by API components; and

   c) the externally visible behavior associated with the RCF 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 defines only interfaces and behavior that must be provided by implementations supporting the Return Channel Frames service in addition to the specification in reference [3].

1.3 APPLICABILITY

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

   a) version 1 as specified by reference [C2]; and

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

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 RCF 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.

Any 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 RCF 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) service user and service provider applications communicating with each other using the SLE API on both sides.

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 RCF service and describes the API model extension including support for the RCF 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 RCF service;

– annex B lists all acronyms used within this document;

– annex C lists informative references.

1.5.2 SLE SERVICE DOCUMENTATION TREE

The SLE suite of Recommended Standards is based on the cross support model defined in the SLE Reference Model (reference [1]). 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.
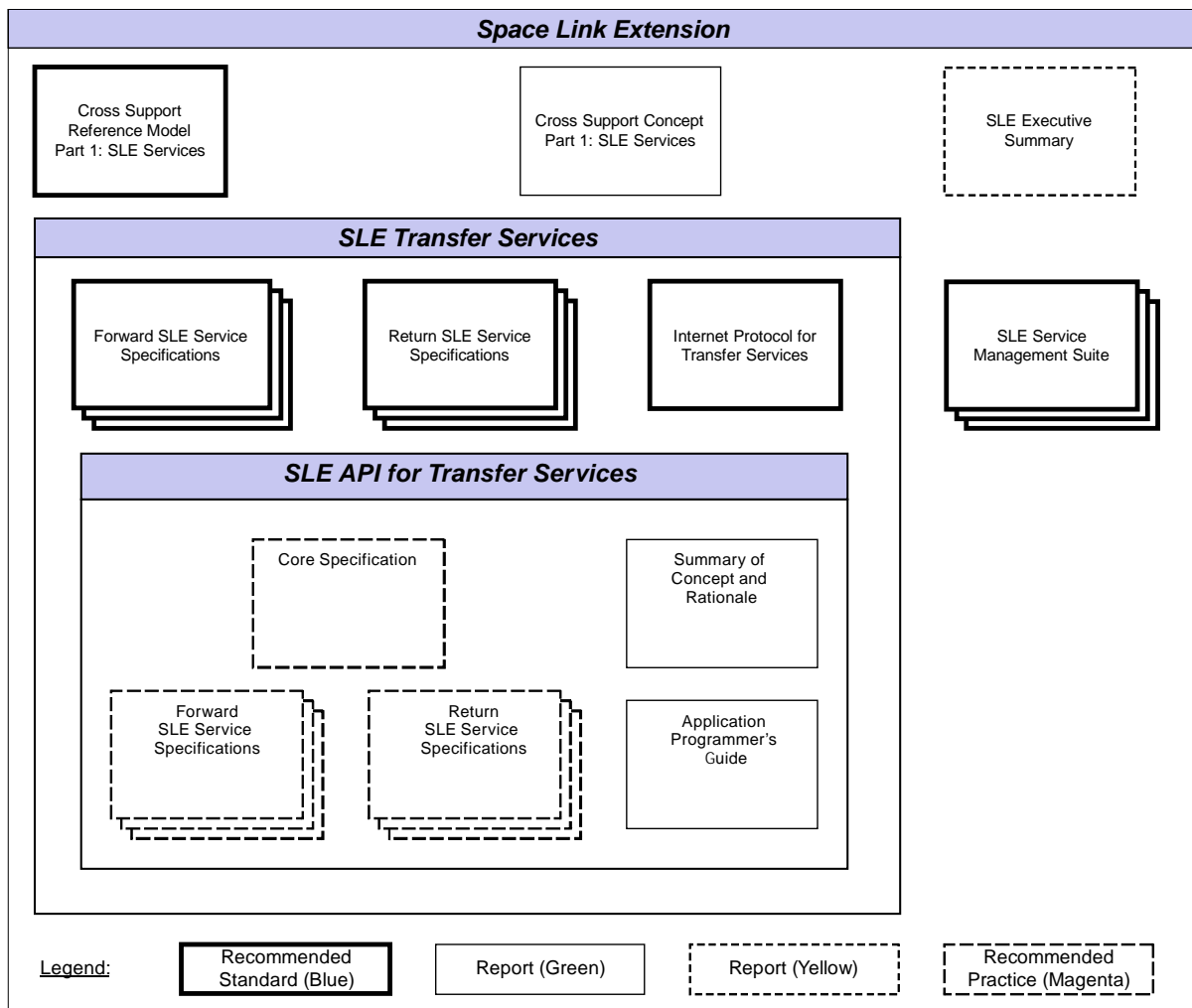
Figure 1-1: SLE Services and SLE API Documentation

a) *Cross Support Reference Model—Part 1: Space Link Extension Services*, a Recommended Standard 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 Recommended Standards that provide specifications of all forward link SLE services.

e) *Return SLE Service Specifications*, a set of Recommended Standards that provide specifications of all return link SLE services.

f) *Internet Protocol for Transfer Services*, a Recommended Standard providing the specification of the wire protocol used for SLE transfer services.

g) *SLE Service Management Specifications*, a set of Recommended Standards 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 SLE Reference Model

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

    a) Return Channel Frames service (RCF 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.

1.6.1.2   Definitions from RCF Service

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

    a) association;

    b) communications service;

    c) confirmed operation;

    d) delivery mode;

    e) global VCID;

    f) invocation;

    g) latency limit;

    h) lock status;

    i) notification;

    j) offline processing latency;

    k) parameter;

    l) performance;

    m) permitted global VCID set;

    n) port identifier;

o) production status;

p) return;

q) service instance provision period;

r) transfer buffer;

s) unconfirmed operation;

t) virtual channel.

1.6.1.3 Definitions from ASN.1 Specification

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

a) Object Identifier;

b) Octet String.

1.6.1.4 Definitions from UML Specification

This Recommended Practice makes use of the following terms defined in reference [C8]:

a) Attribute;

b) Base Class;

c) Class;

d) Data Type;

e) Interface;

f) Method.

1.6.1.5 Definitions from API Core Specification

This Recommended Practice makes use of the following terms defined in reference [3]:

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 [3].

The RCF-specific model extensions in section 2 are presented using the Unified Modeling Language (UML) and applying the conventions defined in reference [3].

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

The RCF-specific data types and interfaces in annex A are specified in the notation of the C++ programming language using the conventions defined in reference [3].

1.7 REFERENCES

The following documents contain provisions which, through reference in this text, constitute provisions of this Recommended Practice. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Recommended Practice 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 documents.

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

[1] *Cross Support Reference Model—Part 1: Space Link Extension Services*. Recommendation for Space Data System Standards, CCSDS 910.4-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, October 2005.

[2] *Space Link Extension—Return Channel Frames Service Specification*. Recommendation for Space Data System Standards, CCSDS 911.2-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, December 2004.

[3] *Space Link Extension—Application Program Interface for Transfer Services—Core Specification*. Specification Concerning Space Data System Standards, CCSDS 914.0-M-1. Magenta Book. Issue 1. Washington, D.C.: CCSDS, October 2008.

[4] *Programming Languages—C++*. International Standard, ISO/IEC 14882:2003. 2nd ed. Geneva: ISO, 2003.

[5] *Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. International Standard, ISO/IEC 8824-1:2002. 3rd ed. Geneva: ISO, 2002.

2 OVERVIEW

2.1 INTRODUCTION

This section describes the extension of the SLE API model in reference [3] for support of the RCF service. Extensions are needed for the API components API Service Element and SLE Operations.

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

2.2 PACKAGE RCF SERVICE INSTANCES

2.2.1 GENERAL

The RCF extensions to the component API Service Element are defined by the package RCF Service Instances. Figure 2-1 provides an overview of this package. The diagram includes classes from the package API Service Element specified in reference [3], which provide applicable specifications for the RCF service.

The package adds two service instance classes:

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

    b) RCF 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 [3].

Both classes are able to handle the specific RCF operations.

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

The class RCF SI Provider adds two new interfaces:

    a) `IRCF_SIAdmin` by which the application can set RCF-specific configuration parameters; and

    b) `IRCF_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 [3].

RCF-specific configuration parameters are defined by the internal class RCF Configuration. The class RCF Status Information defines dynamic status parameters maintained by the service instance.

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 RCF SI USER

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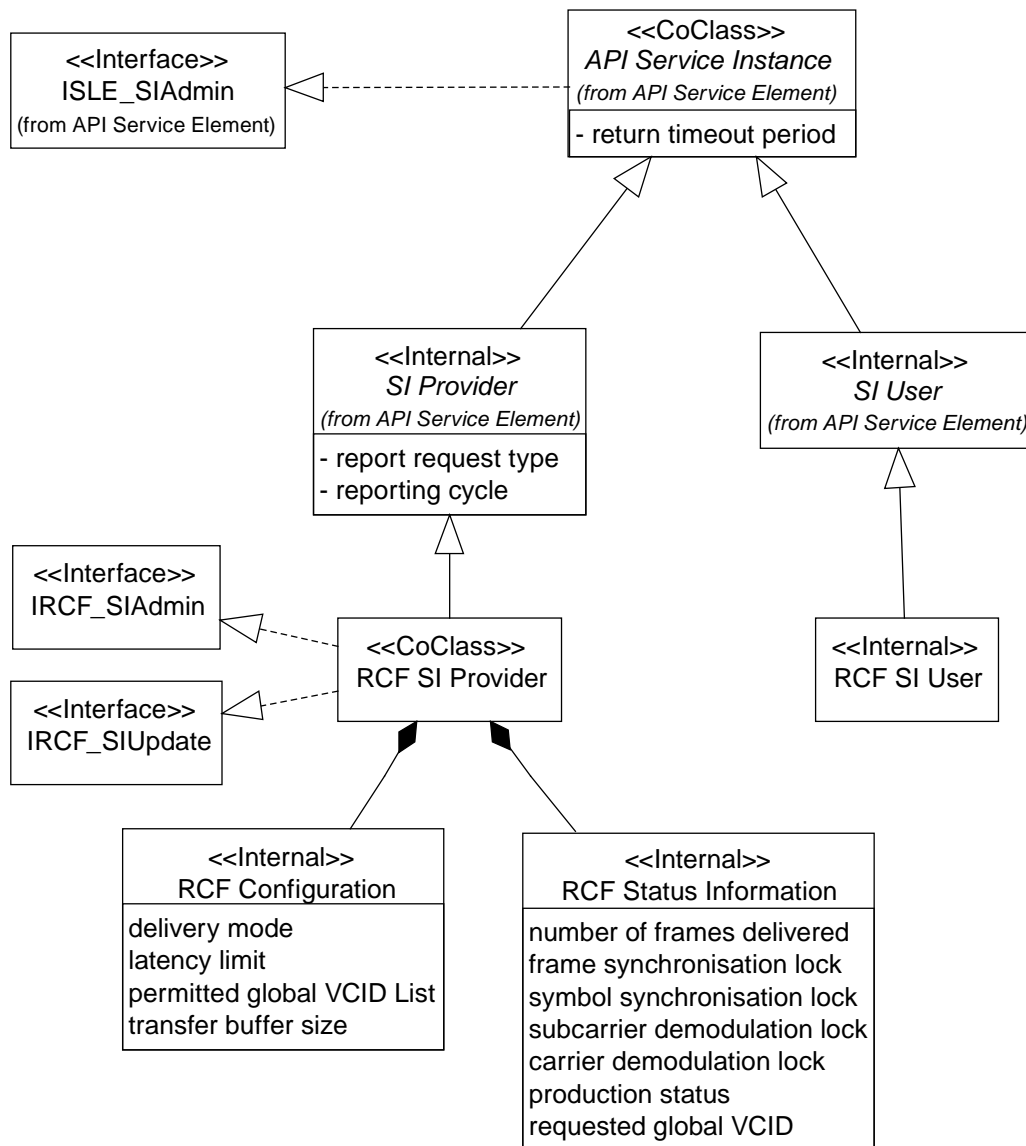

Figure 2-1: RCF Service Instances

2.2.3 COMPONENT CLASS RCF SI PROVIDER

2.2.3.1 General

The class defines a RCF service instance supporting the service provider role. It exports the interfaces `IRCF_SIAdmin` for configuration of the service instance after creation and `IRCF_SIUpdate` for update of dynamic status parameters during operation.

2.2.3.2 Responsibilities

2.2.3.2.1 Service Specific Configuration

The service instance implements the interface `IRCF_SIAdmin` to set the RCF-specific configuration parameters defined by the class RCF 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 [2].

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

2.2.3.2.2 Update of Dynamic Status Parameters

The class implements the interface `IRCF_SIUpdate`. The methods of this interface update status parameters defined by the class RCF Status Information. In order to ensure that the status information is always up to date, all changes of status parameters 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 RCF Status Information from RCF PDUs exchanged between the service provider and the service user. The method used to update each of the parameters is defined in 2.2.5.

The interface `IRCF_SIUpdate` provides read access to all status parameters.

2.2.3.2.3 Handling of the RCF–GET-PARAMETER Operation

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

2.2.3.2.4   Status Reporting

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

2.2.3.2.5   Processing of RCF Protocol Data Units

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

2.2.4   INTERNAL CLASS RCF CONFIGURATION

The class defines the configuration parameters that can be set via the interface `IRCF_SIAdmin`. These parameters are defined by reference [2]. Table 2-1 describes how the service instance uses these parameters.

2.2.5   INTERNAL CLASS RCF STATUS INFORMATION

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

Table 2-1: RCF Configuration Parameters

Parameter	Used for
`delivery-mode`	handling of the transfer buffer (enables / disables discarding of data) checking of PDUs RCF–GET–PARAMETER returns
`latency-limit`	handling of the transfer buffer in the timely online and complete online delivery modes RCF–GET–PARAMETER returns
`permitted-global-VCID-set`	RCF–GET–PARAMETER returns checking of RCF-START invocations
`transfer-buffer-size`	handling of the transfer buffer RCF–GET–PARAMETER returns

Table 2-2: RCF Sta*t*us Information

Parameter	Update	Used for
`number-of-frames-delivered`	count of RCF–TRANSFER–DATA invocations transmitted	status reports
`frame-sync-lock-status`	set by a method of `IRCF_SIUpdate`	status reports
`symbol-sync-lock-status`	set by a method of `IRCF_SIUpdate`	status reports
`subcarrier-lock-status`	set by a method of `IRCF_SIUpdate`	status reports
`carrier-lock-status`	set by a method of `IRCF_SIUpdate`	status reports
`production-status`	set by a method of `IRCF_SIUpdate`	status reports
`requested-global-VCID`	extracted from RCF-START-return with a positive result	RCF-GET-PARAMETER

2.3   PACKAGE RCF OPERATIONS

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

   a) RCF–START;

   b) RCF–TRANSFER–DATA;

   c) RCF–SYNC–NOTIFY;

   d) RCF–STATUS–REPORT;

   e) RCF–GET–PARAMETER.

For other operations the API uses the common operation objects defined in reference [3].

Table 2-3 maps RCF operations to operation object interfaces.

Table 2-3: Mapping of RCF Operations to Operation Object Interfaces

RCF Operation	Operation Object Interface	Defined in Package
RCF–BIND	`ISLE_Bind`	SLE Operations
RCF–UNBIND	`ISLE_Unbind`	SLE Operations
RCF–START	`IRCF_Start`	RCF Operations
RCF–STOP	`ISLE_Stop`	SLE Operations
RCF–TRANSFER–DATA	`IRCF_TransferData`	RCF Operations
RCF–SYNC–NOTIFY	`IRCF_SyncNotify`	RCF Operations
[TRANSFER-BUFFER] (see note)	`ISLE_TransferBuffer`	SLE Operations
RCF–SCHEDULE–STATUS–REPORT	`ISLE_ScheduleStatusReport`	SLE Operations
RCF–STATUS–REPORT	`IRCF_StatusReport`	RCF Operations
RCF–GET–PARAMETER	`IRCF_GetParameter`	RCF Operations
RCF–PEER–ABORT	`ISLE_PeerAbort`	SLE Operations

NOTE – TRANSFER-BUFFER is a pseudo-operation used to handle the transfer buffer defined in reference [2].

<<Inheritance>>

<<Interface>>
ISLE_ConfirmedOperation
(from SLE Operations)

<<Interface>>
ISLE_Operation
(from SLE Operations)

<<Inheritance>>

<<Inheritance>>

<<Inheritance>>

<<Inheritance>>

<<Interface>>
ISLE_PeerAbort
(from SLE Operations)

<<Interface>>
ISLE_TransferBuffer
(from SLE Operations)

<<Inheritance>>

<<Interface>>
IRCF_StatusReport

<<Interface>>
IRCF_SyncNotify

<<Interface>>
IRCF_TransferData

<<Inheritance>>

<<Inheritance>>

<<Interface>>
ISLE_Unbind
(from SLE Operations)

<<Inheritance>>

<<Inheritance>>

<<Interface>>
ISLE_Bind
(from SLE Operations)

<<Inheritance>>

<<Interface>>
IRCF_GetParameter

<<Inheritance>>

<<Interface>>
ISLE_Stop
(from SLE Operations)

<<Interface>>
IRCF_Start

<<Interface>>
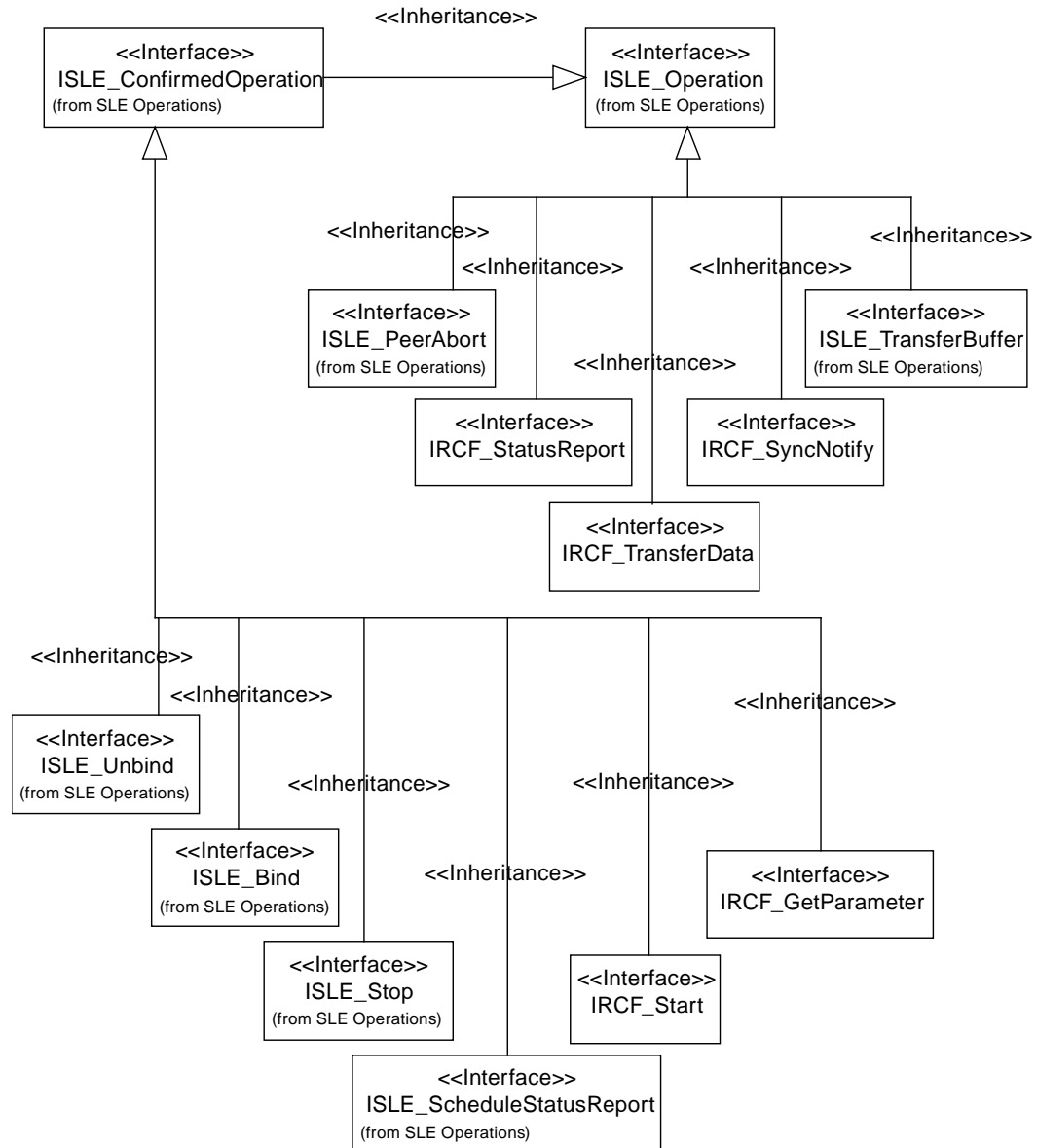ISLE_ScheduleStatusReport
(from SLE Operations)

Figure 2-2: RCF Operation Object Interfaces

2.4 SECURITY ASPECTS OF THE SLE RCF TRANSFER SERVICE

2.4.1 SECURITY BACKGROUND/INTRODUCTION

The SLE transfer services explicitly provide authentication and access control. Additional security capabilities, if required, are levied on the underlying communication services that support the SLE transfer services. The SLE transfer services are defined as layered application services operating over underlying communication services that must meet certain requirements but which are otherwise unspecified. Selection of the underlying communication services over which real SLE implementations connect is based on the requirements of the communicating parties and/or the availability of CCSDS-standard communication technology profiles and proxy specifications. Different underlying communication technology profiles are intended to address not only different performance requirements but also different security requirements. Missions and service providers are expected to select from these technology profiles to acquire the performance and security capabilities appropriate to the mission. Specification of the various underlying communication technologies, and in particular their associated security provisions, are outside the scope of this Recommendation.

The SLE RCF transfer service transfers data that originates on a mission spacecraft. As such, the SLE RCF transfer service has custody of the data for only a portion of the end-to-end data path between mission spacecraft and MDOS. Consequently the ability of an SLE transfer service to secure the transfer of mission spacecraft data is limited to that portion of the end-to-end path that is provided by the SLE transfer service (i.e., the terrestrial link between the MDOS and the ground termination of the space-ground link to the mission spacecraft). End-to-end security must also involve securing the data as it crosses the space-ground link, which can be provided by some combination of securing the mission data itself (e.g., encryption of the mission data within CCSDS space packets) and securing the space-ground link (e.g., encryption of the physical space-ground link). Thus while the SLE RCF transfer service plays a role in the end-to-end security of the data path, it does not control and cannot ensure that end-to-end security. This component perspective is reflected in the security provisions of the SLE transfer services.

2.4.2 STATEMENTS OF SECURITY CONCERNS

This section identifies RCF transfer service support for capabilities that responds to these security concerns in the areas of data privacy, data integrity, authentication, access control, availability of resources, and auditing.

2.4.2.1 Data Privacy (Also Known As Confidentiality)

This SLE RCF transfer service specification does not define explicit data privacy requirements or capabilities to ensure data privacy. Data privacy is expected to be ensured outside of the SLE transfer service layer, by the mission application processes that communicate over the SLE transfer service, in the underlying communication service that

lies under the SLE transfer service, or some combination of both. For example, mission application processes might apply end-to-end encryption to the contents of the CCSDS space link data units carried as data by the SLE transfer service. Alternatively or in addition, the network connection between the SLE entities might be encrypted to provide data privacy in the underlying communication network.

2.4.2.2 Data Integrity

The SLE RCF transfer service defines and enforces a strict sequence of operations that constrain the ability of a third party to inject operation invocations or returns into the transfer service association between a service user and provider (see 4.2.2 in reference [2]). This constrains the ability of a third party to seize control of an active RCF transfer service instance without detection.

The SLE RCF transfer service requires that the underlying communication service transfer data in sequence, completely and with integrity, without duplication, with flow control that notifies the application layer in the event of congestion, and with notification to the application layer in the event that communication between the service user and the service provider is disrupted (see 1.3.1 in reference [2]). No specific mechanisms are identified, as they will be an integral part of the underlying communication service.

2.4.2.3 Authentication

This SLE RCF transfer service specification defines authentication requirements (see 3.1.5 in reference [2]), and defines `initiator-identifier`, `responder-identifier`, `invoker-credentials`, and `performer-credentials` parameters of the service operation invocations and returns that are used to perform SLE transfer service authentication. The procedure by which SLE transfer service operation invocations and returns are authenticated is described in annex F of the Cross Support Service Green Book (reference [C3]). The SLE transfer service authentication capability can be selectively set to authenticate at one of three levels: authenticate every invocation and return, authenticate only the BIND operation invocation and return, or perform no authentication. Depending upon the inherent authentication available from the underlying communication network, the security environment in which the SLE service user and provider are operating, and the security requirements of the spaceflight mission, the SLE transfer service authentication level can be adapted by choosing the SLE operation invocation and returns that shall be authenticated. Furthermore the mechanism used for generating and checking the credentials and thus the level of protection against masquerading (simple or strong authentication) can be selected in accordance with the results of a threat analysis.

2.4.2.4 Access Control

This SLE RCF transfer service specification defines access control requirements (see 3.1.4 in reference [2]), and defines `initiator-identifier` and `responder-identifier`

parameters of the service operation invocations and returns that are used to perform SLE transfer service access control. The procedure by which access to SLE transfer services is controlled is described in annex F of the Cross Support Service Green Book (reference [C3]).

2.4.2.5 Availability of Resources

The SLE transfer services are provided via communication networks that have some limit to the resources available to support those SLE transfer services. If these resources can be diverted from their support of the SLE transfer services (in what is commonly known as "denial of service") then the performance of the SLE transfer services may be curtailed or inhibited. This SLE RCF transfer service specification does not define explicit capabilities to prevent denial of service. Resource availability is expected to be ensured by appropriate capabilities in the underlying communication service. The specific capabilities will be dependent upon the technologies used in the underlying communication service and the security environment in which the transfer service user and provider operate.

2.4.2.6 Auditing

This SLE RCF transfer service specification does not define explicit security auditing requirements or capabilities. Security auditing is expected to be negotiated and implemented bilaterally between the spaceflight mission and the service provider.

2.4.3 POTENTIAL THREATS AND ATTACK SCENARIOS

The SLE RCF transfer service depends on unspecified mechanisms operating above the SLE transfer service (between a mission spacecraft application process and its peer application process on the ground), underneath the SLE transfer service in the underlying communication service, or some combination of both, to ensure data privacy (confidentiality). If no such mechanisms are actually implemented, or the mechanisms selected are inadequate or inappropriate to the network environment in which the mission is operating, an attacker could read the spacecraft telemetry data contained in the RCF protocol data units as they traverse the WAN between service user and service provider.

The SLE RCF transfer service constrains the ability of a third party to seize control of an active SLE transfer service instance, but it does not specify mechanisms that would prevent an attacker from intercepting the protocol data units and replacing the contents of the `data` parameter. The prevention of such a replacement attack depends on unspecified mechanisms operating above the SLE transfer service (between a mission spacecraft application process and its peer application process on the ground), underneath the SLE transfer service in the underlying communication service, in bilaterally-agreed extra capabilities applied to the SLE transfer service (e.g., encryption of the `data` parameter) or some combination of the three. If no such mechanisms are actually implemented, or the mechanisms selected are inadequate or inappropriate to the network environment in which the mission is operating, an attacker could substitute telemetry data without detection.

If the SLE transfer service authentication capability is not used and if authentication is not ensured by the underlying communication service, attackers may somehow obtain valid `initiator-identifier` values and use them to initiate SLE transfer service instances by which they could gain access to spacecraft telemetry data.

The SLE RCF transfer service depends on unspecified mechanisms operating in the underlying communication service to ensure that the supporting network has sufficient resources to provide sufficient support to legitimate users. If no such mechanisms are actually implemented, or the mechanisms selected are inadequate or inappropriate to the network environment in which the mission is operating, an attacker could prevent legitimate users from receiving telemetry from their spacecraft.

If the provider of SLE RCF transfers service provides no security auditing capabilities, or if a user chooses not to employ auditing capabilities that do exist, then attackers may delay or escape detection while stealing or altering telemetry data.

2.4.4   CONSEQUENCES OF NOT APPLYING SECURITY

The consequences of not applying security to the SLE RCF transfer service are possible degradation and loss of ability to receive telemetry from the spacecraft, or the substitution of altered telemetry data.

3   RCF SPECIFIC SPECIFICATIONS FOR API COMPONENTS

3.1   API SERVICE ELEMENT

3.1.1   SERVICE INSTANCE CONFIGURATION

3.1.1.1   The service element shall provide the interface `IRCF_SIAdmin` for configuration of a provider-side service instance after creation.

3.1.1.2   The interface shall provide methods to set the following parameters, which the service element needs for handling of the transfer buffer and delivers to the user in response to a RCF–GET–PARAMETER invocation:

 a) `delivery-mode`;

 b) `transfer-buffer-size`, i.e., the maximum number of RCF–TRANSFER– BUFFER invocations and RCF–SYNC–NOTIFY invocations that can be stored to a transfer buffer PDU submitted to the service user;

 c) `latency-limit`, if the delivery mode is either 'timely online' or 'complete online';

 d) `permitted-global-VCID-set`, i.e., the set of master channels or virtual channels from which the service user may request data.

NOTE –   These parameters are defined in reference [2] for the operation RCF-GET-PARAMETER. Handling of the transfer buffer by the service element is defined in reference [3].

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

 a) the value of the production status at the time the service instance is configured;

 b) the lock status of the frame synchronization process at the time the service instance is configured;

 c) the lock status of the symbol synchronization process at the time the service instance is configured;

 d) the lock status of the sub-carrier demodulation process at the time the service instance is configured;

 e) the lock status of the carrier demodulation process at the time the service instance is configured.

NOTES

1   For the delivery mode 'offline', status reporting is not supported. Therefore, these parameters need not be specified.

2   Further configuration parameters must be set using the interface `ISLE_SIAdmin` specified in reference [3]. These include the parameter `return-timeout-period` required for the RCF-GET-PARAMETER operation.

3.1.1.4 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 [2], the method `ConfigCompleted()` shall return an error.

NOTES

1   As defined in reference [3], the service element 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 perform checks on the limits specified by reference [2].

3.1.1.5   If the delivery mode is 'offline', the service element shall accept the configuration when the parameters defined in 3.1.1.3 have not been specified.

3.1.1.6   Configuration parameters 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.

3.1.1.7   The values of all configuration parameters shall remain unmodified following a RCF-UNBIND or RCF-PEER-ABORT operation and following a protocol-abort.

3.1.1.8   The interface `IRCF_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.

3.1.2 STATUS INFORMATION

3.1.2.1   The service element shall maintain status parameters for every service instance and uses them for generation of status reports and for RCF–GET–PARAMETER returns.

NOTES

1   The parameters are defined in reference [2] for the operations RCF–STATUS–REPORT and RCF–GET–PARAMETER.

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

3.1.2.2 The service element shall update the following status parameters in the methods of the interface `IRCF_SIUpdate` described in 3.1.2.6.

   a) `frame-sync-lock-status;`

   b) `symbol-sync-lock-status;`

   c) `subcarrier-lock-status;`

   d) `carrier-lock-status;` and

   e) `production-status.`

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

3.1.2.3 The service element shall handle the parameter `number-of-frames-delivered` as defined by the following specifications:

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

   b) when a TRANSFER–BUFFER PDU is transmitted to the service user, the parameter shall be incremented by the number of RCF–TRANSFER–DATA invocations in that PDU.

   NOTE –   Frames in a TRANSFER–BUFFER PDU that is discarded shall not be included in the count of frames delivered.

3.1.2.4 The service element shall handle the parameter `requested-global-VCID` as defined by the following specifications:

NOTE –   The parameter `requested-global-VCID` shall be set by a RCF-START invocation and can be requested by a RCF–GET-PARAMETER invocation. It consists of three elements: the spacecraft ID (0 to 1023), the transfer frame version number (0 to 3) and the VC ID (0 to 63). According to reference [2] the VC ID is set to 'any' when a master channel is selected. As this cannot be mapped to C++, the API uses a fourth element which specifies whether the ID refers to a master channel or a virtual channel.

   a) at the time of service instance configuration, the parameter shall be set to NULL;

   NOTE –   Setting the parameter to NULL only implies that a NULL pointer is returned in the method reading the parameter.

   b) if the application transmits a RCF–START return with a positive result, the value of the parameter shall be extracted from the RCF–START invocation;

   c) the parameter shall be reset to NULL following an accepted RCF–STOP invocation, and following RCF–PEER–ABORT and protocol abort;

d) [V1:] if transfer of the parameter is requested by an RCF-GET-PARAMETER invocation and the parameter is NULL, the service element shall return a copy of the first element in the `permitted-global-VCID-set` defined in 3.1.1.2.

NOTE – [V2:] Version 2 of the RCF service foresees returning a NULL value if the service instance is not in the state 'active'.

3.1.2.5 The service element shall provide the interface `IRCF_SIUpdate` for every service instance. This interface must be used by the application to update the status parameters defined in 3.1.2.2.

3.1.2.6 If more than one service instance exists, each service instance must be updated independently.

3.1.2.7 In order to keep the status information up to date and consistent, the methods of the interface `IRCF_SIUpdate` must be invoked for every change, independent of the state of the service instance.

3.1.2.8 The interface `IRCF_SIUpdate` shall provide read access to all status parameters, including those that are derived by other means.

NOTE – In the delivery mode 'offline', status reporting is not supported. Therefore, the application can opt not to update status information in that mode. If status information is not initialized and not updated, retrieval methods shall return the initial parameter values defined in A4.2.

3.1.2.9 The service element shall keep the status parameter `number-of-frames-delivered` up to date for all delivery modes including the delivery mode 'offline'.

3.1.2.10 Status parameters defined in this specification shall not be modified as result of RCF-UNBIND, RCF-PEER-ABORT, or protocol abort.

3.1.3 PROCESSING OF RCF PROTOCOL DATA UNITS

NOTES

1 The service element shall process RCF PDUs according to the general specifications in reference [3]. This section only addresses additional checks and processing steps defined for the RCF service. RCF-specific checks defined in reference [2], but not listed in this section, must be performed by the application.

2 It is noted that 3.1.2 defines further processing requirements for PDUs with respect to update of status information. Annex subsection A3 defines the checks that operation objects perform when the methods `VerifyInvocationArguments()` and `VerifyReturnArguments()` are called. Reference [3] contains specifications defining how the service element handles error codes returned by these methods.

3.1.3.1 RCF START

3.1.3.1.1 When receiving a RCF–START invocation, the service element shall perform the following checks:

a) if the delivery mode is 'offline', the start time must not be null;

b) if the start time is defined and the delivery mode is 'online':

   1) the start time must be equal to or later than the start time of the scheduled provision period of the service instance, and

   2) the start time must be earlier than the stop time of the scheduled provision period;

c) if the delivery mode is 'offline':

   1) the stop time must not be null, and

   2) the stop time must be earlier than current time;

   NOTE – Reference [2] defines an `offline-processing-latency` and requires that the stop time plus the offline processing latency be earlier than current time. If the application makes use of the offline processing latency, the associated checks must be performed by the application.

d) if the stop time is defined and the delivery mode is 'online', the stop time must be earlier than or equal to the stop time of the scheduled provision period;

  NOTE – If the start time and the stop time are defined, the start time must be earlier than the stop time. This check shall be performed by the operation object within the method `VerifyInvocationArguments()` (see 3.2.1).

e) the global VCID must match one of the entries in the permitted global VCID set.

NOTES

1   This check shall only be performed on the provider side for RCF-START invocations received from the service user.

2   The service element shall not check the production status, as this could change before the PDU is processed by the application.

3.1.3.1.2 If any of the checks defined in 3.1.3.1.1 fail, the service element on the provider side shall not forward the PDU to the application but shall respond with a RCF–START return with a negative result and the appropriate diagnostic.

NOTE – As specified in reference [3], the service element shall reject PDUs with errors received from the local application with an appropriate result code.

3.1.3.2 RCF SYNC NOTIFY

When receiving a RCF–SYNC–NOTIFY invocation, the service element on the provider side shall perform the following checks:

a) if the delivery mode is 'offline', the notification type must not be 'loss of frame synchronization', 'production status change', or 'data discarded due to excessive backlog';

b) if the delivery mode is 'timely online', the notification type must not be 'data discarded due to excessive backlog'.

NOTE – This check cannot be performed on the user side, because the service element does not have the required information.

3.1.4 SERVICE INSTANCE SPECIFIC OPERATION FACTORY

For RCF service instances, the interface `ISLE_SIOpFactory` specified in reference [3] shall support creation and configuration of operation objects for the operations specified in 3.2 with exception of the interfaces `IRCF_StatusReport` and `ISLE_TransferBuffer`.

NOTES

1   The initial values of parameters that shall be set for RCF-specific operation objects are defined in annex A.

2   Status reports and the transfer buffer shall be handled by the API Service Element without involvement of the application.

3.2 SLE OPERATIONS

3.2.1   The component SLE Operations shall provide operation objects for the following RCF operations in addition to those specified in reference [3]:

a) RCF–START;

b) RCF–TRANSFER–DATA;

c) RCF–SYNC–NOTIFY;

d) RCF–STATUS–REPORT; and

e) RCF–GET–PARAMETER.

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

3.2.3 The operation factory shall additionally create the following operation objects specified in reference [3] when the requested service type is RCF:

a) SLE–BIND;

b) SLE–UNBIND;

c) SLE–PEER–ABORT;

d) SLE–STOP; and

e) SLE–SCHEDULE–STATUS–REPORT.

3.3   SLE APPLICATION

NOTE –   This subsection summarizes specific obligations of a RCF provider application using the SLE API.

3.3.1   Following creation of a service instance, and setting of the configuration parameters defined in reference [3], the application shall set the configuration parameters defined in 3.1.1 via the interface `IRCF_SIAdmin`.

3.3.2   The application shall update every service instance in the service element with the status information defined in 3.1.2 by invocation of the appropriate methods in the interface `IRCF_SIUpdate`.

3.4   SEQUENCE OF DIAGNOSTIC CODES

3.4.1   GENERAL

3.4.1.1   Reference [2] requires provider applications that do not perform checks in the sequence of the diagnostic codes defined in the Recommended Standard to document the sequence in which checks are actually performed.

3.4.1.2   The specification in 3.1.3.1 does not preserve the sequence of the diagnostic codes defined in reference [2] for the operation RCF–START. This section defines the actual sequence of checks performed by the API Service Element. For the checks that remain to be performed by the provider application, the sequence defined in reference [2] is maintained. Applications applying a different sequence must provide a modified documentation.

3.4.2   SEQUENCE OF RCF START DIAGNOSTIC CODES

3.4.2.1   Codes set by the API Service Element

a) 'duplicate invoke id';

b) 'invalid start time';

   c) 'invalid stop time';

   d) 'missing time value'; and

   e) 'invalid global VC ID'.

3.4.2.2   Codes set by the Application

   a) 'out of service';

   b) 'unable to comply';

   c) 'invalid stop time' (for the delivery mode 'offline' if an offline processing latency is used); and

   d) 'other'.

ANNEX A

RCF SPECIFIC INTERFACES

(Normative)

A1   INTRODUCTION

This annex specifies RCF-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 [3].

The presentation uses the notation and syntax of the C++ programming language as specified in reference [4].

A2   RCF TYPE DEFINITIONS

File   `RCF_Types.h`

The following types have been derived from the ASN.1 module CCSDS-SLE-TRANSFER-SERVICE-RCF-STRUCTURES in reference [2]. The source ASN.1 type is indicated in brackets. For all enumeration types is defined a special value, 'invalid', 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.

Antenna Id Format [AntennaId]

```
typedef enum RCF_AntennaIdFormat
{
  rcfAF_global                      =   0,
  rcfAF_local                       =   1,
  rcfAF_invalid                     =  -1
} RCF_AntennaIdFormat;
```

Reference [2] defines a Local Form (LF) and a Global Form (GF) for the antenna identifier. The local form is a string of octets and the global form is an ASN.1 object identifier.

RCF Get Parameter Diagnostic [DiagnosticRcfGet]

```
typedef enum RCF_GetParameterDiagnostic
{
  rcfGP_unknownParameter            =   0,
  rcfGP_invalid                     =  -1
} RCF_GetParameterDiagnostic;
```

RCF Start Diagnostic [DiagnosticRcfStart]

```
typedef enum RCF_StartDiagnostic
{
  rcfSD_outOfService                =   0,
  rcfSD_unableToComply              =   1,
  rcfSD_invalidStartTime            =   2,
  rcfSD_invalidStopTime             =   3,
  rcfSD_missingTimeValue            =   4,
  rcfSD_invalidGvcId                =   5,
  rcfSD_invalid                     =  -1
} RCF_StartDiagnostic;
```

Channel Type

```
typedef enum RCF_ChannelType
{
  rcfCT_MasterChannel               =   0,
  rcfCT_VirtualChannel              =   1,
  rcfCT_invalid                     =  -1
} RCF_ChannelType;
```

Global VCID [GlobalVcId]

```
typedef struct RCF_Gvcid
{
  RCF_ChannelType    type;
  unsigned long      scid;        /* 0 to 1023 */
  unsigned long      version;     /* 0 to 3    */
  unsigned long      vcid;        /* 0 to 63   */
} RCF_Gvcid;
```

The elements of the structure have been defined as 'long' to avoid padding by the compiler. The member vcId is only defined if type is set to rcfCT_VirtualChannel.

Lock Status [LockStatus]

```
typedef enum RCF_LockStatus
{
  rcfLS_inLock                    =  0,
  rcfLS_outOfLock                 =  1,
  rcfLS_notInUse                  =  2, /*only for
                                           sub-carrier lock */
  rcfLS_unknown                   =  3,
  rcfLS_invalid                   = -1
} RCF_LockStatus;
```

Notification Type [Notification]

```
typedef enum RCF_NotificationType
{
  rcfNT_lossFrameSync             =  0,
  rcfNT_productionStatusChange    =  1,
  rcfNT_excessiveDataBacklog      =  2,
  rcfNT_endOfData                 =  3,
  rcfNT_invalid                   = -1
} RCF_NotificationType;
```

Production Status [RcfProductionStatus]

```
typedef enum RCF_ProductionStatus
{
  rcfPS_running                   =  0,
  rcfPS_interrupted               =  1,
  rcfPS_halted                    =  2,
  rcfPS_invalid                   = -1
} RCF_ProductionStatus;
```

/RCF Parameter Names [RcfGetParameter]

```
typedef enum RCF_ParameterName
{
  rcfPN_bufferSize                =  4,
  rcfPN_deliveryMode              =  6,
  rcfPN_latencyLimit              = 15,
  rcfPN_permittedGvcidSet         = 24,
  rcfPN_reportingCycle            = 26,
  rcfPN_requestedGvcid            = 28,
  rcfPN_returnTimeoutPeriod       = 29,
```

```
  rcfPN_invalid                  = -1
} RCF_ParameterName;
```

Parameters that can be read using a RCF–GET–PARAMETER operation. The parameter name values are taken from the type `ParameterName` in ASN.1 module CCSDS-SLE-TRANSFER-SERVICE-COMMON-TYPES in reference [2].

Delivery Modes

```
typedef enum RCF_DeliveryMode
{
  rcfDM_timelyOnline    = sleDM_rtnTimelyOnline,
  rcfDM_completeOnline  = sleDM_rtnCompleteOnline,
  rcfDM_offline         = sleDM_rtnOffline,
  rcfDM_invalid         = -1
} RCF_DeliveryMode;
```

The delivery modes are defined as a subset of `SLE_DeliveryMode` in reference [3].

A3   RCF OPERATION OBJECTS

A3.1   RCF START OPERATION

Name   IRCF_Start
GUID   {638A73E0-7FE6-11d3-9F15-00104B4F22C0}
Inheritance: IUnknown – ISLE_Operation – ISLE_ConfirmedOperation
File   IRCF_Start.H

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

Synopsis

```
#include <RCF_Types.h>
#include <ISLE_ConfirmedOperation.H>
interface ISLE_Time;
#define IID_IRCF_Start_DEF { 0x638a73e0, 0x7fe6, 0x11d3, \
        { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }
interface IRCF_Start : ISLE_ConfirmedOperation
{
  virtual const ISLE_Time*
    Get_StartTime() const = 0;
  virtual const ISLE_Time*
    Get_StopTime() const = 0;
  virtual const RCF_Gvcid*
    Get_Gvcid() const = 0;
  virtual RCF_StartDiagnostic
    Get_StartDiagnostic() const = 0;
  virtual void
    Set_StartTime( const ISLE_Time& time ) = 0;
  virtual void
    Put_StartTime( ISLE_Time* ptime ) = 0;
  virtual void
    Set_StopTime( const ISLE_Time& time ) = 0;
  virtual void
    Put_StopTime( ISLE_Time* ptime ) = 0;
  virtual void
    Set_Gvcid( const RCF_Gvcid& id ) = 0;
  virtual void
    Put_Gvcid( RCF_Gvcid* pid ) = 0;
  virtual void
    Set_StartDiagnostic( RCF_StartDiagnostic diagnostic ) = 0;
};
```

Methods

const ISLE_Time* Get_StartTime() const;

Returns the reception time of the first frame to be delivered, or NULL if the parameter is not defined.

const ISLE_Time* Get_StopTime() const;

Returns the reception time of the last frame to be delivered, or NULL if the parameter is not defined.

const RCF_Gvcid* Get_Gvcid() const;

Returns the global VCID requested by the service user, or a NULL pointer if the parameter has not been set.

RCF_StartDiagnostic Get_StartDiagnostic() const;

Returns the value of the diagnostic code.

<u>Precondition</u>: the result is negative, and the diagnostic type is set to 'specific'.

void Set_StartTime(const ISLE_Time& time);

Copies the argument to the receive time of the first frame to be delivered.

void Put_StartTime(ISLE_Time* ptime);

Stores the argument as receive time of the first frame to be delivered.

void Set_StopTime(const ISLE_Time& time);

Copies the argument to the receive time of the last frame to be delivered.

void Put_StopTime(ISLE_Time* ptime);

Stores the argument as receive time of the last frame to be delivered.

void Set_Gvcid(const RCF_Gvcid& id);

Copies the elements of the structure passed as argument to the parameter global VCID.

void Put_Gvcid(RCF_Gvcid* pid);

Stores the input argument to the parameter global VCID.

void Set_StartDiagnostic(RCF_StartDiagnostic 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

Parameter	Created directly	Created by Service Instance
start time	NULL	NULL
stop time	NULL	NULL
global VCID	NULL	NULL
START diagnostic	'invalid'	'invalid'

Checking of Invocation Parameters

Parameter	Required condition
start time	if the start and the stop time are used, must be earlier than stop time
stop time	if the start and the stop time are used, must be later than start time
global VCID	must not be NULL
type	must not be 'invalid'

spacecraft identifier	if the version number is 0 (version 1)
	    must be a value on the range 0 to 1023;
	if the version number is 1 (version 2)
	    must be a value in the range 0 to 255;
	otherwise
	    no checks are applied.

version number	must be either 0 or 1

VC ID	if the type is 'virtual channel' AND the version number is 0 (version 1)
	    must be a value in the range 0 to 7;
	if the type is 'virtual channel' AND the version number is 1 (version 2)
	    must be a value in the range 0 to 63;
	otherwise
	    no checks are applied.

NOTE – In the above table, the parameter 'version number' refers to the transfer frame version number, not the version of the RCF service.

Additional Return Codes for `VerifiyInvocationArguments()`

`SLE_E_TIMERANGE` specification of the start and stop times is inconsistent.
`SLE_E_INVALIDID` the global VC ID (spacecraft ID, version number, and VC ID) is invalid.

Checking of Return Parameters

Parameter	Required condition
START diagnostic	must not be 'invalid' if the result is 'negative' and the diagnostic type is 'specific'

A3.2   RCF TRANSFER DATA OPERATION

Name   IRCF_TransferData
GUID   {638A73E1-7FE6-11d3-9F15-00104B4F22C0}
Inheritance: IUnknown – ISLE_Operation
File   IRCF_TransferData.H

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

Synopsis

```
#include <RCF_Types.h>
#include <ISLE_Operation.H>
interface ISLE_Time;
#define IID_IRCF_TransferData_DEF { 0x638a73e1, 0x7fe6, 0x11d3, \
        { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }
interface IRCF_TransferData : ISLE_Operation
{
  virtual const ISLE_Time*
    Get_EarthReceiveTime() const = 0;
  virtual RCF_AntennaIdFormat
    Get_AntennaIdFormat() const = 0;
  virtual const SLE_Octet*
    Get_AntennaIdLF( size_t& size ) const = 0;
  virtual const int*
    Get_AntennaIdGF( int& length ) const = 0;
  virtual char*
    Get_AntennaIdGFString() const = 0;
  virtual int
    Get_DataLinkContinuity() const = 0;
  virtual const SLE_Octet*
    Get_PrivateAnnotation( size_t& size ) const = 0;
  virtual SLE_Octet*
    Remove_PrivateAnnotation( size_t& size ) = 0;
  virtual const SLE_Octet*
    Get_Data( size_t& size ) const = 0;
  virtual SLE_Octet*
    Remove_Data( size_t& size ) = 0;
  virtual void
    Set_EarthReceiveTime( const ISLE_Time& time ) = 0;
  virtual void
    Put_EarthReceiveTime( ISLE_Time* ptime ) = 0;
  virtual void
    Set_AntennaIdLF( const SLE_Octet* id, size_t size ) = 0;
  virtual void
    Set_AntennaIdGF( const int* id, int length ) = 0;
  virtual void
    Set_AntennaIdGFString( const char* id) = 0;
  virtual void
    Set_DataLinkContinuity( int numFrames ) = 0;
  virtual void
    Set_PrivateAnnotation( const SLE_Octet* pannotation,
                           size_t size ) = 0;
  virtual void
    Put_PrivateAnnotation( SLE_Octet* pannotation,
                           size_t size ) = 0;
  virtual void
    Set_Data( const SLE_Octet* pdata, size_t size ) = 0;
```

```
  virtual void
    Put_Data( SLE_Octet* pdata, size_t size ) = 0;
};
```

Methods

virtual const ISLE_Time* Get_EarthReceiveTime() const;

Returns the earth receive time of the frame delivered, if the parameter has been set in the object. Returns NULL otherwise.

RCF_AntennaIdFormat Get_AntennaIdFormat() const;

Returns the format of the antenna identifier (octet string or object identifier) or 'invalid' when the parameter has not been set.

const SLE_Octet* Get_AntennaIdLF(size_t& size) const;

Returns the antenna identifier in the local form, i.e., a string of octets.

Arguments
size   the number of octets in the antenna ID (1 to 16)

Precondition: Get_AntennaIdFormat() returns rcfAF_local.

const int* Get_AntennaIdGF(int& length) const;

Returns the antenna identifier in the global form, i.e., an object identifier as defined by ASN.1. In C++ this is represented as a sequence of integers.

Arguments
length   the number of elements in the antenna ID

Precondition: Get_AntennaIdFormat() returns rcfAF_global.

char* Get_AntennaIdGFString() const;

Returns the antenna ID as a character string formatted as a dot separated list of numbers. The string is allocated on the heap and must be deleted by the client.

Precondition: Get_AntennaIdFormat() returns rcfAF_global.

int Get_DataLinkContinuity() const;

Returns the data link continuity parameter, if the parameter has been set in the object, or –2 if the parameter has not been set. A valid value can be –1, 0, or any positive number.

const SLE_Octet* Get_PrivateAnnotation(size_t& size) const;

Returns a pointer to the private annotation in the object or NULL if the private annotation has not been set.

Arguments
length   the length of the private annotation in bytes

SLE_Octet* Remove_PrivateAnnotation(size_t& size);

Returns the private annotation data and removes them form the object. The memory allocated by the parameter must be released by the client. If the parameter has not been set returns NULL.

Arguments
length   the length of the private annotation in bytes

const SLE_Octet* Get_Data(size_t& size) const;

Returns a pointer to the frame in the object or NULL if the frame has not been inserted.

Arguments
length   the length of the frame in bytes

SLE_Octet* Remove_Data(size_t& size);

Returns the frame and removes it form the object. The memory allocated by the frame must be released by the client. If the parameter has not been set returns NULL.

Arguments
length   the length of the frame in bytes

void Set_EarthReceiveTime(const ISLE_Time& time);

Copies the value of the argument to the earth receive time.

void Put_EarthReceiveTime(ISLE_Time* ptime);

Stores the argument to the parameter earth receive time.

void Set_AntennaIdLF(const SLE_Octet* id, size_t size);

Sets the antenna id format to 'local form' and the antenna id to the value of the argument. The local form of the antenna id is a string of octets.

`void Set_AntennaIdGF( const int* id, int length );`

Sets the antenna id format to 'global form' and the antenna id to the value of the argument. The global form the antenna id is an object identifier as defined by ASN.1, represented as a sequence of integers.

`void Set_AntennaIdGFString( const char* id);`

Sets the antenna id format to 'global form' and the antenna id to the value of the argument. If the argument is badly formatted, the parameter is reset to its initial state, i.e., 'not set'.

Arguments
id   an object identifier formatted as a dot separated list of numbers

`void Set_DataLinkContinuity( int numFrames );`

Sets the parameter data link continuity to the value of the argument.

`void Set_PrivateAnnotation( const SLE_Octet* pannotation, size_t size );`

Copies size bytes from the argument `pannotation` to the parameter private annotation.

Arguments
pannotation   pointer to the private annotation
length   the length of the annotation in bytes

`void Put_PrivateAnnotation( SLE_Octet* pannotation, size_t size );`

Stores the argument `pannotation` to the parameter private annotation.

Arguments
pannotation   pointer to the private annotation
length   the length of the annotation in bytes

`void Set_Data( const SLE_Octet* pdata, size_t size );`

Copies size bytes from the argument `pdata` to the parameter 'data'.

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

`void Put_Data( SLE_Octet* pdata, size_t size );`

Stores the argument `pdata` to the parameter 'data'.

<u>Arguments</u>

`pdata`   pointer to the data

`length`   the length of the data in bytes

Initial Values of Operation Parameters after Creation

Parameter	Created directly	Created by Service Instance
earth receive time	NULL	NULL
antenna id	NULL	NULL
antenna id format	'invalid'	'invalid'
data link continuity	-2	-2
private annotation	NULL	NULL
data	NULL	NULL

Checking of Invocation Parameters

Parameter	Required condition
earth receive time	must not be NULL
antenna id	must not be NULL
data link continuity	must be > -2
data	must not be NULL

A3.3   RCF SYNC NOTIFY OPERATION

Name   IRCF_SyncNotify
GUID   {638A73E2-7FE6-11d3-9F15-00104B4F22C0}
Inheritance: IUnknown – ISLE_Operation
File   IRCF_SyncNotify.H

The interface provides access to the parameters of the unconfirmed operation
RCF-SYNC-NOTIFY.

Synopsis

```
#include <RCF_Types.h>
#include <ISLE_Operation.H>
interface ISLE_Time;
#define IID_IRCF_SyncNotify_DEF { 0x638a73e2, 0x7fe6, 0x11d3, \
          { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }
interface IRCF_SyncNotify : ISLE_Operation
{
  virtual RCF_NotificationType
    Get_NotificationType() const = 0;
  virtual const ISLE_Time*
    Get_LossOfLockTime() const = 0;
  virtual RCF_LockStatus
    Get_CarrierDemodLock() const = 0;
  virtual RCF_LockStatus
    Get_SubCarrierDemodLock() const = 0;
  virtual RCF_LockStatus
    Get_SymbolSyncLock() const = 0;
  virtual RCF_ProductionStatus
    Get_ProductionStatus() const = 0;
  virtual void
    Set_LossOfFrameSync( const ISLE_Time& time,
                         RCF_LockStatus symbolSyncLock,
                         RCF_LockStatus subCarrierDemodLock,
                         RCF_LockStatus carrierDemodLock ) = 0;
  virtual void
    Set_ProductionStatusChange( RCF_ProductionStatus status ) = 0;
  virtual void
    Set_DataDiscarded() = 0;
  virtual void
    Set_EndOfData() = 0;
};
```

Methods

RCF_NotificationType Get_NotificationType() const;

Returns the type of the notification.

const ISLE_Time* Get_LossOfLockTime() const;

Returns the time at which the frame synchronizer lost lock.

<u>Precondition</u>: notification type is 'loss of frame synchronization'.

RCF_LockStatus Get_CarrierDemodLock() const;

Returns the lock status of the carrier demodulation process.

<u>Precondition</u>: notification type is 'loss of frame synchronization'.

RCF_LockStatus Get_SubCarrierDemodLock() const;

Returns the lock status of the sub-carrier demodulation process.

<u>Precondition</u>: notification type is 'loss of frame synchronization'.

RCF_LockStatus Get_SymbolSyncLock() const;

Returns the lock status of the symbol synchronization process.

<u>Precondition</u>: notification type is 'loss of frame synchronization'.

RCF_ProductionStatus Get_ProductionStatus() const;

Returns the production status.

<u>Precondition</u>: notification type is 'production status change'.

```
void Set_LossOfFrameSync( const ISLE_Time& time,
                          RCF_LockStatus symbolSyncLock,
                          RCF_LockStatus subCarrierDemodLock,
                          RCF_LockStatus carrierDemodLock );
```

Sets the notification type to 'loss of frame synchronization' and the notification values as specified by the arguments.

<u>Arguments</u>
time	the time at which the frame synchronizer lost lock
symbolSyncLock	the lock status of the symbol synchronization process
subCarrierDemodLock	the lock status of the sub-carrier demodulation process
carrierDemodLock	the lock status of the carrier demodulation process

void Set_ProductionStatusChange(RCF_ProductionStatus status);

Sets the notification type to 'production status change' and the notification value as defined by the argument.

void Set_DataDiscarded();

Sets the notification type to 'data discarded due to excessive backlog'.

void Set_EndOfData();

Sets the notification type to 'end of data'.

Initial Values of Operation Parameters after Creation

Parameter	Created directly	Created by Service Instance
notification-type	'invalid'	'invalid'
loss of lock time	NULL	NULL
symbol-sync-lock-status	'invalid'	'invalid'
subcarrier-lock-status	'invalid'	'invalid'
carrier-lock-status	'invalid'	'invalid'
production-status	'invalid'	'invalid'

Checking of Invocation Parameters

Parameter	Required condition
notification-type	must not be 'invalid'
loss of lock time	if notification type is 'loss of frame synchronization' must not be NULL
symbol-sync-lock-status	if notification type is 'loss of frame synchronization' must not be 'invalid' or 'not in use'
subcarrier-lock-status	if notification type is 'loss of frame synchronization' must not be 'invalid'
carrier-lock-status	if notification type is 'loss of frame synchronization' must not be 'invalid' or 'not in use'
production-status	if notification type is 'production status change' must not be 'invalid'

A3.4   RCF STATUS REPORT OPERATION

Name   IRCF_StatusReport
GUID   {638A73E3-7FE6-11d3-9F15-00104B4F22C0}
Inheritance: IUnknown – ISLE_Operation
File   IRCF_StatusReport.H

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

Synopsis

```
#include <RCF_Types.h>
#include <ISLE_Operation.H>
#define IID_IRCF_StatusReport_DEF { 0x638a73e3, 0x7fe6, 0x11d3, \
         { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }
interface IRCF_StatusReport : ISLE_Operation
{
  virtual unsigned long
    Get_NumFrames() const = 0;
  virtual RCF_LockStatus
    Get_FrameSyncLock() const = 0;
  virtual RCF_LockStatus
    Get_CarrierDemodLock() const = 0;
  virtual RCF_LockStatus
    Get_SubCarrierDemodLock() const = 0;
  virtual RCF_LockStatus
    Get_SymbolSyncLock() const = 0;
  virtual RCF_ProductionStatus
    Get_ProductionStatus() const = 0;
  virtual void
    Set_NumFrames( unsigned long count ) = 0;
  virtual void
    Set_FrameSyncLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_CarrierDemodLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_SubCarrierDemodLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_SymbolSyncLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_ProductionStatus( RCF_ProductionStatus status ) = 0;
};
```

Methods

unsigned long Get_NumFrames() const;

Returns the total number of frames delivered.

RCF_LockStatus Get_FrameSyncLock() const;

Returns the lock status of the frame synchronization process.

`RCF_LockStatus Get_CarrierDemodLock() const;`

Returns the lock status of the carrier demodulation process.

`RCF_LockStatus Get_SubCarrierDemodLock() const;`

Returns the lock status of the sub-carrier demodulation process.

`RCF_LockStatus Get_SymbolSyncLock() const;`

Returns the lock status of the symbol synchronization process.

`RCF_ProductionStatus Get_ProductionStatus() const;`

Returns the production status.

`void Set_NumFrames( unsigned long count );`

Sets the total number of frames delivered as defined by the argument.

`void Set_FrameSyncLock( RCF_LockStatus status );`

Sets the frame synchronizer lock status as defined by the argument.

`void Set_CarrierDemodLock( RCF_LockStatus status );`

Sets the carrier demodulator lock status as defined by the argument.

`void Set_SubCarrierDemodLock( RCF_LockStatus status );`

Sets the sub-carrier demodulator lock status as defined by the argument.

`void Set_SymbolSyncLock( RCF_LockStatus status );`

Sets the symbol synchronizer lock status as defined by the argument.

`void Set_ProductionStatus( RCF_ProductionStatus status );`

Sets the production status as defined by the argument.

Initial Values of Operation Parameters after Creation

Parameter	Created directly
`number-of-frames`	0
`frame-sync-lock-`	'invalid'

Parameter	Created directly
`status`	
`symbol-sync-lock-status`	'invalid'
`subcarrier-lock-status`	'invalid'
`carrier-lock-status`	'invalid'
`production-status`	'invalid'

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

Checking of Invocation Parameters

Parameter	Required condition
`frame-sync-lock-status`	must not be 'invalid' or 'not in use'
`symbol-sync-lock-status`	must not be 'invalid' or 'not in use'
`subcarrier-lock-status`	must not be 'invalid'
`carrier-lock-status`	must not be 'invalid' or 'not in use'
`production-status`	must not be 'invalid'

A3.5   RCF GET PARAMETER OPERATION

Name   IRCF_GetParameter
GUID   {638A73E4-7FE6-11d3-9F15-00104B4F22C0}
Inheritance: IUnknown – ISLE_Operation – ISLE_ConfirmedOperation
File   IRCF_GetParameter.H

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

Synopsis

```
#include <RCF_Types.h>
#include <ISLE_ConfirmedOperation.H>
#define IID_IRCF_GetParameter_DEF { 0x638a73e4, 0x7fe6, 0x11d3, \
          { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }
interface IRCF_GetParameter : ISLE_ConfirmedOperation
{
  virtual RCF_ParameterName
    Get_RequestedParameter() const = 0;
  virtual RCF_ParameterName
    Get_ReturnedParameter() const = 0;
  virtual RCF_DeliveryMode
    Get_DeliveryMode() const = 0;
  virtual unsigned short
    Get_LatencyLimit() const = 0;
  virtual unsigned long
    Get_TransferBufferSize() const = 0;
  virtual const RCF_Gvcid*
    Get_RequestedGvcid() const = 0;
  virtual const RCF_Gvcid*
    Get_PermittedGvcidSet( size_t& size ) const = 0;
  virtual RCF_Gvcid*
    Remove_PermittedGvcidSet( size_t& size ) = 0;
  virtual unsigned long
    Get_ReportingCycle() const = 0;
  virtual unsigned long
    Get_ReturnTimeoutPeriod() const = 0;
  virtual RCF_GetParameterDiagnostic
    Get_GetParameterDiagnostic() const = 0;
  virtual void
    Set_RequestedParameter( RCF_ParameterName name ) = 0;
  virtual void
    Set_DeliveryMode( RCF_DeliveryMode mode ) = 0;
  virtual void
    Set_LatencyLimit( unsigned short limit ) = 0;
  virtual void
    Set_TransferBufferSize( unsigned long size ) = 0;
  virtual void
    Set_RequestedGvcid( const RCF_Gvcid* id ) = 0;
  virtual void
    Put_RequestedGvcid( RCF_Gvcid* pid ) = 0;
  virtual void
    Set_PermittedGvcidSet( const RCF_Gvcid* idSet,
                           size_t size ) = 0;
  virtual void
    Put_PermittedGvcidSet( RCF_Gvcid* idSet,
```

```
                                     size_t size ) = 0;
  virtual void
    Set_ReportingCycle( unsigned long cycle ) = 0;
  virtual void
    Set_ReturnTimeoutPeriod( unsigned long period ) = 0;
  virtual void
    Set_GetParameterDiagnostic( RCF_GetParameterDiagnostic
                                diagostic ) = 0;
};
```

Methods

RCF_ParameterName Get_RequestedParameter() const;

Returns the identification of the parameter whose value shall be returned.

RCF_ParameterName Get_ReturnedParameter() const;

Returns the identification of the parameter whose value is reported.

RCF_DeliveryMode Get_DeliveryMode() const;

Returns the delivery mode of the service instance.

Precondition: the returned parameter is delivery-mode.

unsigned short Get_LatencyLimit() const;

Returns the latency limit defined by service management. If the delivery mode is 'offline' returns zero.

Precondition: the returned parameter is latency-limit.

unsigned long Get_TransferBufferSize() const;

Returns the size of the transfer buffer as the maximum number of RCF–TRANSFER–DATA invocations and RCF–SYNC–NOTIFY invocations that can be stored in the buffer.

Precondition: the returned parameter is transfer-buffer-size.

const RCF_Gvcid* Get_RequestedGvcid() const;

Returns the requested global VCID if that has been set. Otherwise returns a NULL pointer. This parameter is only meaningful if the VCID has been set by a START operation.

Precondition: the returned parameter is 'requested global VCID'.

`const RCF_Gvcid* Get_PermittedGvcidSet( size_t& size ) const;`

Returns the set of global VCIDs to which the service instance has access. If the parameter has not been set or the set has been removed, returns a NULL pointer.

Precondition: the returned parameter is 'permitted global VCID set'.

`RCF_Gvcid* Remove_PermittedGvcidSet( size_t& size );`

Returns the list of global VCIDs to which the service instance has access and removes the list from the object. If the parameter has not been set or the list has been removed, returns a NULL pointer.

Precondition: the returned parameter is 'permitted global VCID set'.

`unsigned long Get_ReportingCycle() 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'.

`RCF_GetParameterDiagnostic Get_GetParameterDiagnostic() const;`

Returns the diagnostic code.

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

`void Set_RequestedParameter( RCF_ParameterName name );`

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

`void Set_DeliveryMode( RCF_DeliveryMode mode );`

Sets the returned parameter name to 'delivery mode' and the value as defined by the argument.

`void Set_LatencyLimit( unsigned short limit );`

Sets the returned parameter name to 'latency limit' and the value as defined by the argument.

void Set_TransferBufferSize(unsigned long size);

Sets the returned parameter name to 'transfer buffer size' and the value as defined by the argument.

void Set_RequestedGvcid(const RCF_Gvcid* id);

Sets the returned parameter name to 'requested global VCID' and copies its value from the argument.

void Put_RequestedGvcid(RCF_Gvcid* pid);

Sets the returned parameter name to 'requested global VCID' and stores the argument as the value of this parameter.

void Set_PermittedGvcidSet(const RCF_Gvcid* idSet, size_t size);

Sets the returned parameter name to 'permitted global VCID set' and copies its value from the argument.

void Put_PermittedGvcidSet(RCF_Gvcid* idSet, size_t size);

Sets the returned parameter name to 'permitted global VCID set' and stores the argument as the value of this parameter.

void Set_ReportingCycle(unsigned long cycle);

Sets the returned parameter name to 'reporting cycle' and the value as defined by the argument.

void Set_ReturnTimeoutPeriod(unsigned long period);

Sets the returned parameter name to 'return timeout period' and the value as defined by the argument.

void Set_GetParameterDiagnostic(RCF_GetParameterDiagnostic diagostic);

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

Parameter	Created directly	Created by Service Instance
requested parameter	'invalid'	'invalid'
returned parameter	'invalid'	'invalid'
`delivery-mode`	'invalid'	'invalid'
`latency-limit`	0	0
`transfer-buffer-size`	0	0
`requested-global-VCID`	NULL	NULL
`permitted-global-VCID-set`	NULL	NULL
`reporting-cycle`	0	0
`return-timeout-period`	0	0
GET PARAMETER diagnostic	'invalid'	'invalid'

Checking of Invocation Parameters

Parameter	Required condition
requested parameter	must not be 'invalid'

Checking of Return Parameters

Parameter	Required condition
returned parameter	must be the same as 'requested parameter'
`delivery-mode`	If the returned parameter is 'delivery mode' must not be 'invalid'
`transfer-buffer-size`	If the returned parameter is 'transfer buffer size' must not be 0
`requested-global-VCID`	[V1:] if the returned parameter is 'requested global VCID' must not be NULL and must have the following structure [V2:] if the returned parameter is 'requested global VCID' must either be NULL or must have the following structure
Type	must not be 'invalid'

spacecraft ID	if the version number is 0 (version 1)
	      must be a value on the range 0 to 1023;
	if the version number is 1 (version 2)
	      must be a value in the range 0 to 255;
	otherwise
	      no checks are applied.

version number	must be either 0 or 1

VCID	if the type is 'virtual channel' AND the version number is 0 (version 1)
	      must be a value in the range 0 to 7
	if the type is 'virtual channel' AND the version number is 1 (version 2)
	      must be a value in the range 0 to 63
	otherwise
	      no checks are applied

`permitted-global-VCID-set`	if the returned parameter is 'permitted global VCID set' must not be NULL
`return-timeout-period`	If the returned parameter is 'return timeout period' must not be 0
GET PARAMETER diagnostic	must not be 'invalid' if the result is 'negative' and the diagnostic type is 'specific'

NOTE – In the above table, the parameter 'version number' refers to the transfer frame version number, not the version of the RCF service.

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 above only need to be performed when the return is received by the service user.

A4   RCF SERVICE INSTANCE INTERFACES

A4.1   SERVICE INSTANCE CONFIGURATION

Name   IRCF_SIAdmin
GUID   {638A73E5-7FE6-11d3-9F15-00104B4F22C0}
Inheritance: IUnknown
File   IRCF_SIAdmin.H

The interface provides write and read access to the RCF-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 required parameters have been set and returns an error when the configuration is not complete.

Configuration parameters must not be set after successful return of the method ConfigCompleted(). The effect of invoking these methods at a later stage is undefined.

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

Synopsis

```
#include <RCF_Types.h>
#include <SLE_SCM.H>
#define IID_IRCF_SIAdmin_DEF { 0x638a73e5, 0x7fe6, 0x11d3, \
          { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }
interface IRCF_SIAdmin : IUnknown
{
  virtual void
    Set_DeliveryMode( RCF_DeliveryMode mode ) = 0;
  virtual void
    Set_LatencyLimit( unsigned short limit ) = 0;
  virtual void
    Set_TransferBufferSize( unsigned long size ) = 0;
  virtual void
    Set_PermittedGvcidSet( const RCF_Gvcid* idSet,
                           size_t size ) = 0;
  virtual void
    Set_InitialProductionStatus( RCF_ProductionStatus status ) = 0;
  virtual void
    Set_InitialFrameSyncLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_InitialCarrierDemodLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_InitialSubCarrierDemodLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_InitialSymbolSyncLock( RCF_LockStatus status ) = 0;
  virtual RCF_DeliveryMode
    Get_DeliveryMode() const = 0;
  virtual unsigned short
    Get_LatencyLimit() const = 0;
  virtual unsigned long
```

```
    Get_TransferBufferSize() const = 0;
  virtual const RCF_Gvcid*
    Get_PermittedGvcidSet( size_t& size ) const = 0;
};
```

Methods

void Set_DeliveryMode(RCF_DeliveryMode mode);

Sets the delivery mode of the service instance.

void Set_LatencyLimit(unsigned short limit);

Sets the latency limit in seconds for transmission of the transfer buffer. If the delivery mode is offline, the parameter need not be set.

void Set_TransferBufferSize(unsigned long size);

Sets the maximum number of RCF–TRANSFER–DATA invocations and RCF–SYNC–NOTIFY invocations that shall be stored in one transfer buffer PDU.

void Set_PermittedGvcidSet(const RCF_Gvcid* idSet, size_t size);

Sets the set of global VCIDs to which the service instance has access. This set must not be empty and all members must be valid global VCIDs.

void Set_InitialProductionStatus(RCF_ProductionStatus status);

Sets the value of the production status at the time of configuration. The parameter is used to initialize status report parameters. If the delivery mode is 'offline', this parameter need not be set.

void Set_InitialFrameSyncLock(RCF_LockStatus status);

Sets the lock status of the frame synchronization process at the time of configuration. The parameter is used to initialize status report parameters. If the delivery mode is 'offline', this parameter need not be set.

void Set_InitialCarrierDemodLock(RCF_LockStatus status);

Sets the lock status of the carrier demodulation process at the time of configuration. The parameter is used to initialize status report parameters. If the delivery mode is 'offline', this parameter need not be set.

void Set_InitialSubCarrierDemodLock(RCF_LockStatus status);

Sets the lock status of the sub-carrier demodulation process at the time of configuration. The parameter is used to initialize status report parameters. If the delivery mode is 'offline', this parameter need not be set.

void Set_InitialSymbolSyncLock(RCF_LockStatus status);

Sets the lock status of the symbol synchronization process at the time of configuration. The parameter is used to initialize status report parameters. If the delivery mode is 'offline', this parameter need not be set.

RCF_DeliveryMode Get_DeliveryMode() const;

Returns the value of the parameter delivery-mode.

unsigned short Get_LatencyLimit() const;

Returns the value of the parameter latency-limit.

unsigned long Get_TransferBufferSize() const;

Returns the value of the parameter transfer-buffer-size.

const RCF_Gvcid* Get_PermittedGvcidSet(size_t& size) const;

Returns the set of global VCIDs to which the service instance has access.

A4.2 UPDATE OF SERVICE INSTANCE PARAMETERS

Name   IRCF_SIUpdate
GUID   {638A73E6-7FE6-11d3-9F15-00104B4F22C0}
Inheritance: IUnknown
File   IRCF_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 the information changes, independent of the state of the service instance.

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 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.

In the delivery mode 'offline', status reporting is not supported. Therefore configuration parameters used to initialize the status report need not be supplied and the status information need not be updated. If the initial values and updates are not supplied, the retrieval methods return the values defined at the end of this section. Values accumulated by the service element are kept up to date for all delivery modes, including the mode 'offline'.

Synopsis

```
#include <RCF_Types.h>
#include <SLE_SCM.H>
#define IID_IRCF_SIUpdate_DEF { 0x638a73e6, 0x7fe6, 0x11d3, \
          { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }
interface IRCF_SIUpdate : IUnknown
{
  virtual void
    Set_ProductionStatus( RCF_ProductionStatus status ) = 0;
  virtual void
    Set_FrameSyncLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_CarrierDemodLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_SubCarrierDemodLock( RCF_LockStatus status ) = 0;
  virtual void
    Set_SymbolSyncLock( RCF_LockStatus status ) = 0;
  virtual RCF_ProductionStatus
    Get_ProductionStatus() const = 0;
  virtual RCF_LockStatus
    Get_FrameSyncLock() const = 0;
  virtual RCF_LockStatus
    Get_CarrierDemodLock() const = 0;
  virtual RCF_LockStatus
    Get_SubCarrierDemodLock() const = 0;
  virtual RCF_LockStatus
    Get_SymbolSyncLock() const = 0;
  virtual unsigned long
    Get_NumFrames() const = 0;
  virtual RCF_Gvcid*
    Get_RequestedGvcid() const = 0;
};
```

Methods

`void Set_ProductionStatus( RCF_ProductionStatus status );`

The method must be called whenever the production status changes to set the new value.

`void Set_FrameSyncLock( RCF_LockStatus status );`

The method must be called whenever the lock status of the frame synchronization process changes to set the new value.

`void Set_CarrierDemodLock( RCF_LockStatus status );`

The method must be called whenever the lock status of the carrier demodulation process changes to set the new value.

`void Set_SubCarrierDemodLock( RCF_LockStatus status );`

The method must be called whenever the lock status of the sub-carrier demodulation process changes to set the new value.

`void Set_SymbolSyncLock( RCF_LockStatus status );`

The method must be called whenever the lock status of the symbol synchronization process changes to set the new value.

`RCF_ProductionStatus Get_ProductionStatus() const;`

Returns the value of the production status.

`RCF_LockStatus Get_FrameSyncLock() const;`

Returns the lock status of the frame synchronization process.

`RCF_LockStatus Get_CarrierDemodLock() const;`

Returns the lock status of the carrier demodulation process.

`RCF_LockStatus Get_SubCarrierDemodLock() const;`

Returns the lock status of the sub-carrier demodulation process.

`RCF_LockStatus Get_SymbolSyncLock() const;`

Returns the lock status of the symbol synchronization process.

unsigned long Get_NumFrames() const;

Returns the total number of frames delivered by the service instance. In the delivery mode timely online this number can be smaller than the number of frames passed to the service element because data might have been discarded because of excessive backlog.

RCF_Gvcid* Get_RequestedGvcid() const;

Returns a copy of the global VCID requested by the service user, or a NULL pointer if the service instance is not in the state 'active'. If a non-NULL pointer is returned, the client must release the memory allocated by the global VCID.

Initial Parameter Values

Parameter	Value
production-status	initial production status set via the interface `IRCF_SIAdmin`, in the delivery mode 'offline' set to 'invalid' if not set via `IRCF_SIAdmin`
frame-sync-lock-status	initial frame synchronizer lock set via the interface `IRCF_SIAdmin`, in the delivery mode 'offline' set to 'unknown' if not set via `IRCF_SIAdmin`
symbol-sync-lock-status	initial symbol synchronizer lock set via the interface `IRCF_SIAdmin`, in the delivery mode 'offline' set to 'unknown' if not set via `IRCF_SIAdmin`
subcarrier-lock-status	initial sub-carrier demodulator lock set via the interface IRCF_SIAdmin, in the delivery mode 'offline' set to 'unknown' if not set via IRCF_SIAdmin
carrier-lock-status	initial carrier demodulator lock set via the interface `IRCF_SIAdmin`, in the delivery mode 'offline' set to 'unknown' if not set via `IRCF_SIAdmin`
number-of-frames-delivered	0
requested-global-VCID	NULL (if NULL, RCF-GET-PARAMETER returns the first element of the 'permitted global VCID set')

ANNEX B

ACRONYMS

(Informative)

This annex expands the acronyms used throughout this Recommended Practice.

API	Application Program Interface
CCSDS	Consultative Committee for Space Data Systems
GUID	Globally Unique Identifier
GVCID	Global Virtual Channel Identifier
ID	Identifier
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MC	Master Channel
OMG	Object Management Group
PDU	Protocol Data Unit
RCF	Return Channel Frames
SI	Service Instance
SLE	Space Link Extension
UML	Unified Modeling Language
VC	Virtual Channel

ANNEX C

INFORMATIVE REFERENCES

(Informative)

[C1] *Procedures Manual for the Consultative Committee for Space Data Systems*. CCSDS A00.0-Y-9. Yellow Book. Issue 9. Washington, D.C.: CCSDS, November 2003.

[C2] *Space Link Extension – Return Channel Frames Service Specification*, Draft Recommendation for Space Data System Standards, CCSDS 911.2 R1.7, Red Book, Issue 1.7, September 1999.

[C3] *Cross Support Concept — Part 1: Space Link Extension Services*. Report Concerning Space Data System Standards, CCSDS 910.3-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, March 2006.

[C4] *Space Link Extension—Internet Protocol for Transfer Services*. Recommendation for Space Data System Standards, CCSDS 913.1-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2008.

[C5] *Space Link Extension—Application Program Interface for Transfer Services— Summary of Concept and Rationale*. Report Concerning Space Data System Standards, CCSDS 914.1-G-1. Green Book. Issue 1. Washington, D.C.: CCSDS, January 2006.

[C6] *Space Link Extension—Application Program Interface for Transfer Services— Application Programmer's Guide*. Report Concerning Space Data System Standards, CCSDS 914.2-G-2. Green Book. Issue 2. Washington, D.C.: CCSDS, October 2008.

[C7] *The COM/DCOM Reference*. COM/DCOM Product Documentation, AX-01. San Francisco: The Open Group, 1999.
<http://www.opengroup.org/products/publications/catalog/ax01.htm>

[C8] *Unified Modeling Language (UML)*. Version 1.5, formal/2003-03-01. Needham, MA: Object Management Group, March 2003.
<http://www.omg.org/technology/documents/modeling_spec_catalog.htm>