



**CCSDS**

The Consultative Committee for Space Data Systems

---

## Recommendation for Space Data System Practices

# **SPACE LINK EXTENSION— APPLICATION PROGRAM INTERFACE FOR RETURN OPERATIONAL CONTROL FIELDS SERVICE**

**RECOMMENDED PRACTICE**

**CCSDS 915.5-M-2**

**MAGENTA BOOK**

**September 2015**

## **Recommendation for Space Data System Practices**

**SPACE LINK EXTENSION—  
APPLICATION PROGRAM  
INTERFACE FOR RETURN  
OPERATIONAL CONTROL  
FIELDS SERVICE**

**RECOMMENDED PRACTICE**

**CCSDS 915.5-M-2**

**MAGENTA BOOK**  
**September 2015**

## AUTHORITY

Issue:	Recommended Practice, Issue 2
Date:	September 2015
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 *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4), and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the e-mail address below.

This document is published and maintained by:

CCSDS Secretariat  
National Aeronautics and Space Administration  
Washington, DC, USA  
E-mail: [secretariat@mailman.ccsds.org](mailto:secretariat@mailman.ccsds.org)

## 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**.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CCSDS has processes for identifying patent issues and for securing from the patent holder agreement that all licensing policies are reasonable and non-discriminatory. However, CCSDS does not have a patent law staff, and CCSDS shall not be held responsible for identifying any or all such patent rights.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Standard is therefore subject to CCSDS document management and change control procedures, which are defined in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). 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 sent to the CCSDS Secretariat at the e-mail 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.
- 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 (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.
- UK Space Agency/United Kingdom.

### Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSP0)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- 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.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- 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.
- Ministry of Communications (MOC)/Israel.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- South African National Space Agency (SANSA)/Republic of South Africa.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

**DOCUMENT CONTROL**

<b>Document</b>	<b>Title</b>	<b>Date</b>	<b>Status</b>
CCSDS 915.2-M-1	Space Link Extension—Application Program Interface for Return Channel Frames Service, Recommended Practice, Issue 1	October 2008	Original issue, superseded
CCSDS 915.5-M-2	Space Link Extension—Application Program Interface for Return Operational Control Fields Service, Recommended Practice, Issue 2	September 2015	Current issue: – updates text to accommodate changes in current version of SLE service specification; – differentiates applicability by SLE service specification version; – updates references.

NOTE – Substantive changes from the previous issue are marked with change bars in the inside margin.

## CONTENTS

<u>Section</u>	<u>Page</u>
<b>1 INTRODUCTION.....</b>	<b>1-1</b>
1.1 PURPOSE.....	1-1
1.2 SCOPE.....	1-1
1.3 APPLICABILITY.....	1-2
1.4 RATIONALE.....	1-2
1.5 DOCUMENT STRUCTURE .....	1-3
1.6 DEFINITIONS, NOMENCLATURE, AND CONVENTIONS .....	1-5
1.7 REFERENCES .....	1-8
<b>2 OVERVIEW .....</b>	<b>2-1</b>
2.1 INTRODUCTION .....	2-1
2.2 PACKAGE ROCF SERVICE INSTANCES.....	2-1
2.3 PACKAGE ROCF OPERATIONS .....	2-5
2.4 SECURITY ASPECTS OF THE SLE ROCF TRANSFER SERVICE .....	2-7
<b>3 ROCF SPECIFIC SPECIFICATIONS FOR API COMPONENTS.....</b>	<b>3-1</b>
3.1 API SERVICE ELEMENT.....	3-1
3.2 SLE OPERATIONS .....	3-9
3.3 SLE APPLICATION .....	3-9
3.4 SEQUENCE OF DIAGNOSTIC CODES.....	3-10
<b>ANNEX A ROCF SPECIFIC INTERFACES (NORMATIVE).....</b>	<b>A-1</b>
<b>ANNEX B ACRONYMS (INFORMATIVE).....</b>	<b>B-1</b>
<b>ANNEX C INFORMATIVE REFERENCES (INFORMATIVE) .....</b>	<b>C-1</b>

### Figure

1-1 SLE Services and SLE API Documentation.....	1-4
2-1 ROCF Service Instances .....	2-2
2-2 ROCF Operation Object Interfaces .....	2-6

### Table

2-1 ROCF Configuration Parameters.....	2-4
2-2 ROCF Status Information .....	2-5
2-3 Mapping of ROCF Operations to Operation Object Interfaces.....	2-7



# 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 Operational Control Fields (ROCF) service defined in reference [2].

## 1.2 SCOPE

This specification defines extensions to the SLE API in terms of:

- a) the ROCF-specific functionality provided by API components;
- b) the ROCF-specific interfaces provided by API components; and
- c) the externally visible behavior associated with the ROCF 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 specification defines only interfaces and behavior that must be provided by implementations supporting the Return Operational Control Fields service in addition to the specification in reference [3].

### 1.3 APPLICABILITY

The Application Program Interface specified in this document supports two generations of Return Operational Control Fields service, namely:

- a) Generation 2 identified by the version number 1 in the BIND operation, as specified by reference [C8];
- b) Generation 3 identified by the version number 4 in the BIND operation, as specified by reference [2].

#### NOTES

- 1 The use of the term ‘Generation’ follows the definition in the API Core Specification (reference [3]) where it is used to classify all SLE Transfer Services.
- 2 Support for Generation 1 is not valid in this book because the ROCF initial issue was written as part of Generation 2 of the RAF, RCF and CLTU specific version 1 recommendations.

Support for Generation 2 of the service is included for backward compatibility purposes for a limited time and may not be continued in future versions of this specification.

Provisions within this Recommended Practice that are specific for a specific generation are marked as:

- [G2:] for provisions specific to Generation 2;
- [G3:] for provisions specific to Generation 3.

Provisions that apply to all generations are not marked.

### 1.4 RATIONALE

This Recommended Practice specifies the mapping of the ROCF 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 ROCF service and describes the API model extension including support for the ROCF service defined in reference [2];
- 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 ROCF service;
- annex B lists all acronyms used within this document;
- annex C lists informative references.

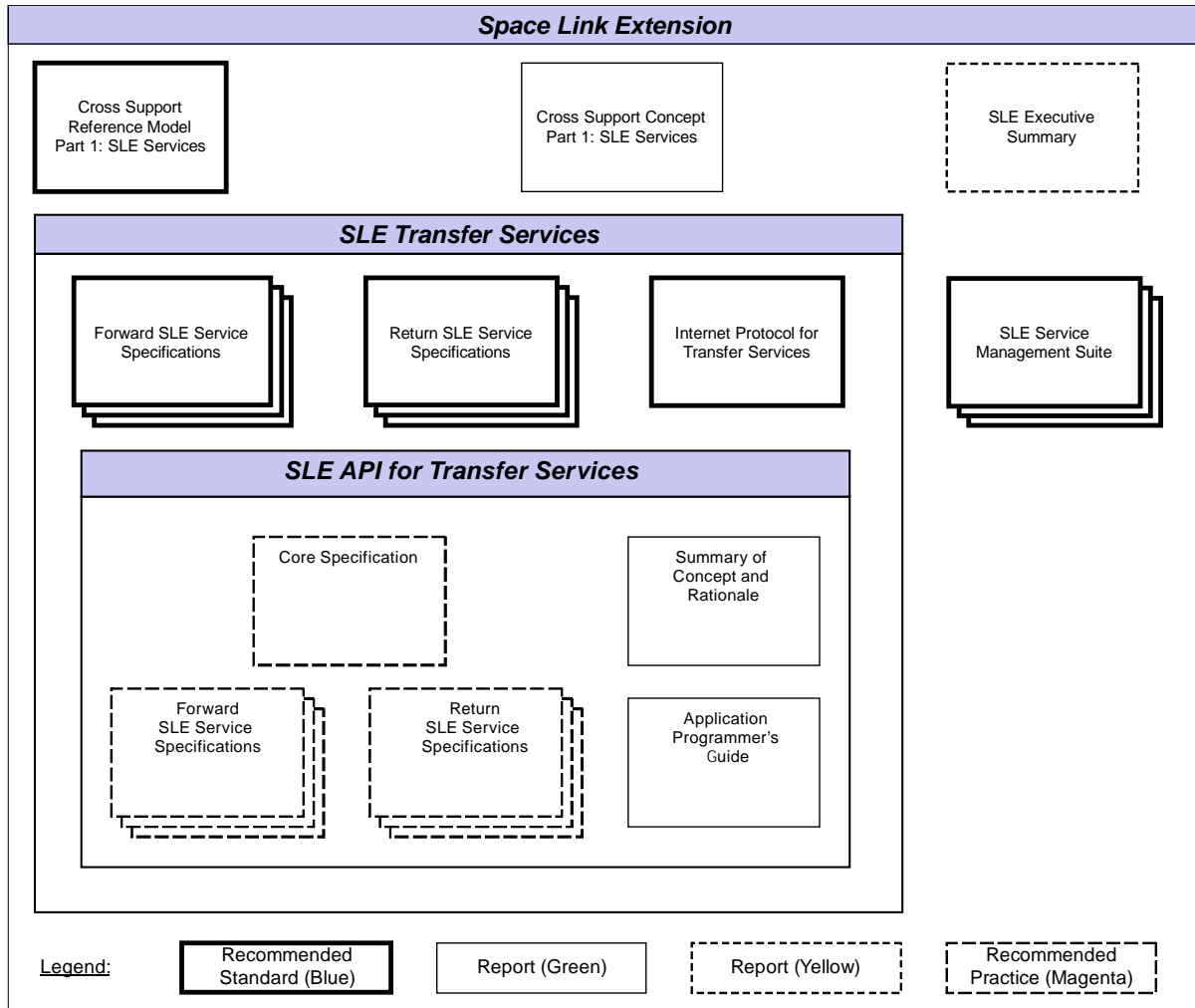
### **1.5.2 SLE SERVICE DOCUMENTATION TREE**

The SLE suite of Recommended Standards is based on the cross support model defined in the SLE Reference Model (reference [1]). The SLE services 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**

- 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.
- Cross Support Concept—Part 1: Space Link Extension Services*, a Report introducing the concepts of cross support and the SLE services.
- 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.
- Forward SLE Service Specifications*, a set of Recommended Standards that provide specifications of all forward link SLE services.
- 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 Operational Control Fields service (ROCF 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 ROCF 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 [C7]:

- 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 Programming Interface;
- b) Component.

### **1.6.2 NOMENCLATURE**

#### **1.6.2.1 Normative Text**

The following conventions apply for the normative specifications in this Recommended Standard:

- 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.

NOTE – These conventions do not imply constraints on diction in text that is clearly informative in nature.

#### **1.6.2.2 Informative Text**

In the normative sections of this document, informative text is set off from the normative specifications either in notes or under one of the following subsection headings:

- Overview;
- Background;

- Rationale;
- Discussion.

### 1.6.3 CONVENTIONS

This document applies the conventions defined in reference [3].

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

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

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

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

- [1] *Cross Support Reference Model—Part 1: Space Link Extension Services*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 910.4-B-2. Washington, D.C.: CCSDS, October 2005.
- [2] *Space Link Extension—Return Operational Control Fields Service Specification*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 911.5-B-2. Washington, D.C.: CCSDS, January 2010.
- [3] *Space Link Extension—Application Program Interface for Transfer Services—Core Specification*. Issue 2. Recommendation for Space Data System Practices (Magenta Book), CCSDS 914.0-M-2. Washington, D.C.: CCSDS, September 2015.
- [4] *Programming Languages—C++*. 3rd ed. International Standard, ISO/IEC 14882:2011. Geneva: ISO, 2011.
- [5] *Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. 4th ed. International Standard, ISO/IEC 8824-1:2008. Geneva: ISO, 2008.



## 2 OVERVIEW

### 2.1 INTRODUCTION

This section describes the extension of the SLE API model in reference [3] for support of the ROCF 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 ROCF-specific protocol data units.

### 2.2 PACKAGE ROCF SERVICE INSTANCES

#### 2.2.1 GENERAL

The ROCF extensions to the component API Service Element are defined by the package ROCF 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 ROCF service.

The package adds two service instance classes:

- a) ROCF SI User, supporting the service user role; and
- b) ROCF 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 ROCF operations.

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

The class ROCF SI Provider adds two new interfaces:

- a) IROCF\_SIAdmin by which the application can set ROCF-specific configuration parameters; and
- b) IROCF\_SIUUpdate 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].

ROCF-specific configuration parameters are defined by the internal class ROCF Configuration. The class ROCF 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 ROCF SI USER

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

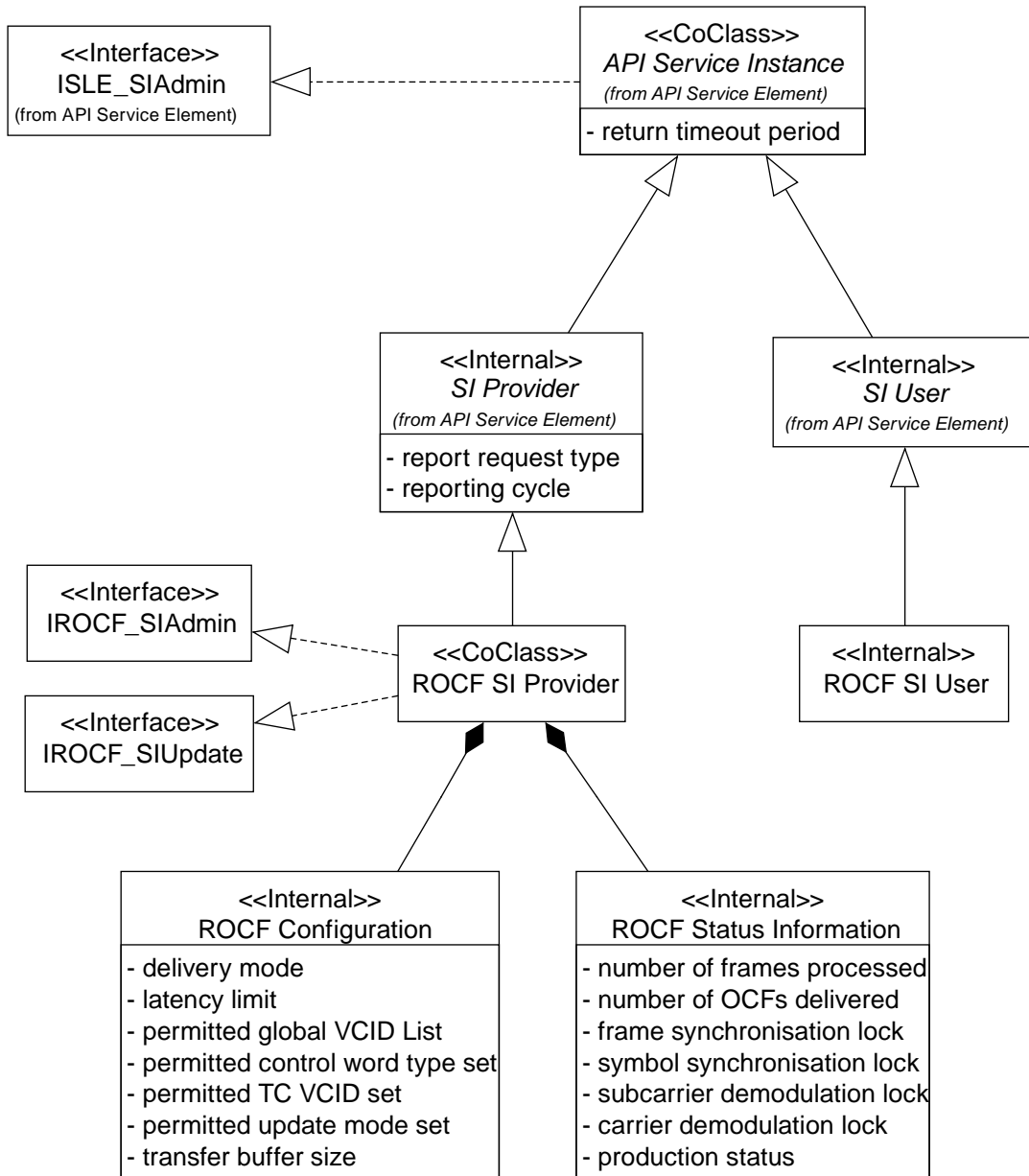


Figure 2-1: ROCF Service Instances

## 2.2.3 COMPONENT CLASS ROCF SI PROVIDER

### 2.2.3.1 General

The class defines a ROCF service instance supporting the service provider role. It exports the interfaces `IROCF_SIAAdmin` for configuration of the service instance after creation and `IROCF_SIUUpdate` 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 `IROCF_SIAAdmin` to set the ROCF-specific configuration parameters defined by the class `ROCF 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_SIAAdmin`) have been set, the method `ISLE_SIAAdmin::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 `IROCF_SIAAdmin` provides read access to the configuration parameters.

#### 2.2.3.2.2 Update of Dynamic Status Parameters

The class implements the interface `IROCF_SIUUpdate`. The methods of this interface update status parameters defined by the class `ROCF 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 `ROCF Status Information` from `ROCF 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 `IROCF_SIUUpdate` provides read access to all status parameters.

#### 2.2.3.2.3 Handling of the ROCF-GET-PARAMETER Operation

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

#### 2.2.3.2.4 Status Reporting

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

#### 2.2.3.2.5 Processing of ROCF Protocol Data Units

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

### 2.2.4 INTERNAL CLASS ROCF CONFIGURATION

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

**Table 2-1: ROCF Configuration Parameters**

Parameter	Used for
delivery-mode	handling of the transfer buffer (enables / disables discarding of data) checking of PDUs ROCF-GET-PARAMETER returns
latency-limit	handling of the transfer buffer in the timely online and complete online delivery modes ROCF-GET-PARAMETER returns
permitted-global-VCID-list	ROCF-GET-PARAMETER returns checking of ROCF-START invocations
permitted-control-word-type-set	ROCF-GET-PARAMETER returns checking of ROCF-START invocations
permitted-TC-VCID-set	ROCF-GET-PARAMETER returns checking of ROCF-START invocations
permitted-update-mode-set	ROCF-GET-PARAMETER returns checking of ROCF-START invocations
transfer-buffer-size	handling of the transfer buffer ROCF-GET-PARAMETER returns

### 2.2.5 INTERNAL CLASS ROCF 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 2-2: ROCF Status Information**

Parameter	Update	Used for
number-of-frames-processed	count of telemetry frames processed for OCF extraction set by a method of IROCF_SIUpdate	status reports
number-of-ocfs-delivered	count of ROCF-TRANSFER-DATA invocations transmitted	status reports
frame-sync-lock-status	set by a method of IROCF_SIUpdate	status reports
symbol-sync-lock-status	set by a method of IROCF_SIUpdate	status reports
subcarrier-lock-status	set by a method of IROCF_SIUpdate	status reports
carrier-lock-status	set by a method of IROCF_SIUpdate	status reports
production-status	set by a method of IROCF_SIUpdate	status reports
requested-global-VCID	extracted from ROCF-START-return with a positive result	GET-PARAMETER
requested-control-word-type	extracted from ROCF-START-return with a positive result	GET-PARAMETER
requested-TC-VCID	extracted from ROCF-START-return with a positive result	GET-PARAMETER
requested-update-mode	extracted from ROCF-START-return with a positive result	GET-PARAMETER

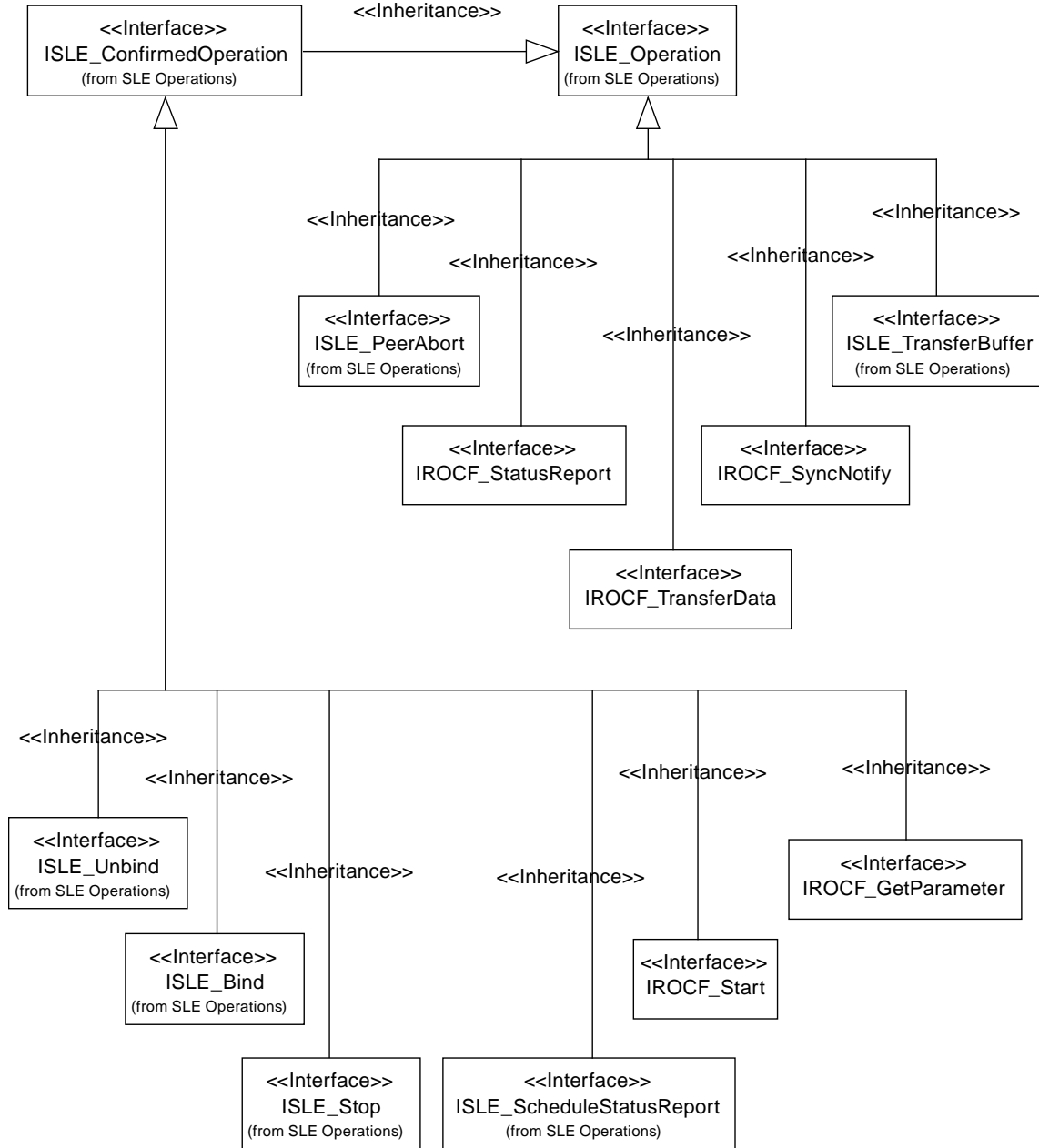
### 2.3 PACKAGE ROCF OPERATIONS

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

- a) ROCF-START;
- b) ROCF-TRANSFER-DATA;
- c) ROCF-SYNC-NOTIFY;

- d) ROCF-STATUS-REPORT; and
- e) ROCF-GET-PARAMETER.

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



**Figure 2-2: ROCF Operation Object Interfaces**

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

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

ROCF Operation	Operation Object Interface	Defined in Package
ROCF-BIND	ISLE_Bind	SLE Operations
ROCF-UNBIND	ISLE_Unbind	SLE Operations
ROCF-START	IROCF_Start	ROCF Operations
ROCF-STOP	ISLE_Stop	SLE Operations
ROCF-TRANSFER-DATA	IROCF_TransferData	ROCF Operations
ROCF-SYNC-NOTIFY	IROCF_SyncNotify	ROCF Operations
[TRANSFER-BUFFER] (see note)	ISLE_TransferBuffer	SLE Operations
ROCF-SCHEDULE-STATUS-REPORT	ISLE_ScheduleStatusReport	SLE Operations
ROCF-STATUS-REPORT	IROCF_StatusReport	ROCF Operations
ROCF-GET-PARAMETER	IROCF_GetParameter	ROCF Operations
ROCF-PEER-ABORT	ISLE_PeerAbort	SLE Operations

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

## 2.4 SECURITY ASPECTS OF THE SLE ROCF 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 ROCF transfer service transfers data that originates on a mission spacecraft. As such, the SLE ROCF 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 ROCF 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**

### **2.4.2.1 Overview**

This subsection identifies ROCF 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.

### **2.4.2.2 Data Privacy (also known as Confidentiality)**

This SLE ROCF 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.

### **2.4.2.3 Data Integrity**

The SLE ROCF 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 ROCF transfer service instance without detection.

The SLE ROCF 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.

#### **2.4.2.4 Authentication**

This SLE ROCF 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 [C2]). 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 invocations 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.

#### **2.4.2.5 Access Control**

This SLE ROCF 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 [C2]).

#### **2.4.2.6 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 ROCF 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.

### 2.4.2.7 Auditing

This SLE ROCF 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 ROCF 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 Operational Control Field (OCF) data contained in the ROCF protocol data units as they traverse the WAN between service user and service provider.

NOTE – In the case of the ROCF transfer service, being able to protect the confidentiality of the OCF data at the mission application level is unlikely because the most common payload of the OCF is the Communications Link Control Word (CLCW). The CLCW is specified as part of the CCSDS-standard Communications Operation Procedure (COP), which has no provision for protecting the confidentiality of the CLCW. The OCF may also be used directly by mission-unique applications, and in such cases end-to-end confidentiality mechanisms may be applied to the contents of the OCF. However, such mission-unique applications are few in comparison to the usage of the OCF to transfer CLCWs. So in most cases the confidentiality of the OCF data will need to depend solely on the underlying communication service.

The SLE ROCF 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 OCF data without detection. The most likely use of such an attack would be to substitute CLCWs to corrupt the operation of the COP, resulting in degradation or loss of commanding ability.

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 OCF data.

The SLE ROCF 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 OCF data from their spacecraft, and (when the OCFs carry CLCWs) inhibit the operation of the COP.

If the provider of SLE ROCF 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, altering, or preventing delivery of OCF data.

#### **2.4.4 CONSEQUENCES OF NOT APPLYING SECURITY**

The consequences of not applying security to the SLE ROCF transfer service are possible degradation and loss of ability to receive OCF from the spacecraft, the substitution of altered OCF data, and the degradation or loss of commanding ability due to the corruption of COP operation.

### 3 ROCF 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 `IROCF_SIAAdmin` 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 `ROCF-GET-PARAMETER` invocation:

- a) `delivery-mode`;
- b) `transfer-buffer-size`, i.e., the maximum number of `ROCF-TRANSFER-BUFFER` invocations and `ROCF-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-list`, i.e., the list of master channels or virtual channels from which the service user may request data;
- e) `permitted-control-word-type-set`, i.e., the set of control word types from which the service user may request data;
- f) `permitted-TC-VCID-set`, i.e., the set of TC VCIDs for which the service user may request data;
- g) `permitted-update-mode-set`, i.e., the set update modes for which the service user may request data.

NOTE – These parameters are defined in reference [2] for the operation `ROCF-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_SIAAdmin` specified in reference [3]. These include the parameter `return-timeout-period` required for the `ROCF-GET-PARAMETER` operation.

**3.1.1.4** All configuration parameters must be set before the method `ConfigCompleted()` of the interface `ISLE_SIAAdmin` 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_SIAAdmin`. 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 `ROCF-UNBIND` or `ROCF-PEER-ABORT` operation and following a protocol-abort.

**3.1.1.8** The interface `IROCF_SIAAdmin` 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 ROCF-GET-PARAMETER returns.

#### NOTES

- 1 The parameters are defined in reference [2] for the operations ROCF-STATUS-REPORT and ROCF-GET-PARAMETER.
- 2 Handling of the parameter `reporting-cycle` defined for the ROCF-GET-PARAMETER operation is specified in reference [3].

**3.1.2.2** The service element shall update the following status parameters in the methods of the interface IROCF\_SIUpdate described in 3.1.2.10.

- 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-processed` as defined by the following specifications:

- a) the parameter shall be set to zero if the service instance is configured;
- b) the parameter shall be set to the value as provided by the application on the IROCF\_SIUpdate interface.

**3.1.2.4** The service element shall handle the parameter `number-of-ocfs-delivered` as defined by the following specifications:

- a) the parameter shall be set to zero if the service instance is configured;
- b) if a TRANSFER-BUFFER PDU is transmitted to the service user, the parameter shall be incremented by the number of ROCF-TRANSFER-DATA invocations in that PDU.

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

**3.1.2.5** 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 `ROCF-START` invocation and can be requested by a `ROCF-GET-PARAMETER` invocation. It consists of three elements: the spacecraft ID (0 to 1023), the version number (0 to 1) and the VC ID (0 to 63). According to reference [2] the VC ID is set to ‘any’ if 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 `ROCF-START` return with a positive result, the value of the parameter shall be extracted from the `ROCF-START` invocation;

c) the parameter shall be reset to NULL following an accepted `ROCF-STOP` invocation, and following `ROCF-PEER-ABORT` and protocol abort.

**3.1.2.6** The service element shall handle the parameter `requested-control-word-type` as defined by the following specifications:

NOTE – The parameter `requested-control-word-type` shall be set by a `ROCF-START` invocation and can be requested by a `ROCF-GET-PARAMETER` invocation.

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

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

c) the parameter shall be reset to ‘invalid’ following an accepted `ROCF-STOP` invocation, and following `ROCF-PEER-ABORT` and protocol abort.

**3.1.2.7** The service element shall handle the parameter `requested-TC-VCID` as defined by the following specifications:

NOTE – The parameter `requested-TC-VCID` shall be set by the previous `ROCF-START` invocation and can be requested by a `ROCF-GET-PARAMETER` invocation.

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

NOTE – Setting the parameter to NULL only implies that the method `Get_TcVcidUsed()` returns FALSE.

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

NOTE – The parameter shall be set to NULL in the ROCF-START invocation if the `control-word-type` is not 'clw', or if the `control-word-type` is 'clw' and the OCF from all TC VCIDs shall be transmitted.

- c) the parameter shall be reset to NULL following an accepted ROCF-STOP invocation, and following ROCF-PEER-ABORT and protocol abort.

**3.1.2.8** The service element shall handle the parameter `requested-update-mode` as defined by the following specifications:

NOTE – The parameter `requested-update-mode` shall be set by a ROCF-START invocation and can be requested by a ROCF-GET-PARAMETER invocation.

- a) at the time of service instance configuration, the parameter shall be set to 'invalid';
- b) if the application transmits a ROCF-START return with a positive result, the value of the parameter shall be extracted from the ROCF-START invocation;
- c) the parameter shall be reset to 'invalid' following an accepted ROCF-STOP invocation, and following ROCF-PEER-ABORT and protocol abort.

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

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

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

**3.1.2.12** The interface `IROCF_SIUUpdate` 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.

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

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



### 3.1.3 PROCESSING OF ROCF PROTOCOL DATA UNITS

#### NOTES

- 1 The service element shall process ROCF PDUs according to the general specifications in reference [3]. This section only addresses additional checks and processing steps defined for the ROCF service. ROCF-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 ROCF START

**3.1.3.1.1** When receiving a ROCF-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 list;

NOTES

- 1 This check shall only be performed on the provider side for ROCF-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.
- f) the control word type must match one of the entries in the permitted control word type set;

NOTE – This check shall only be performed on the provider side for ROCF-START invocations received from the service user.

- g) in case the TC VCID is set and the control word type is ‘clw’, the TC VCID must match one of the entries in the permitted TC VCID set;

NOTES

- 1 If the TC VCID is not set and the control word type is ‘clw’, the CLCW for all TC VCs shall be requested.
  - 2 This check shall only be performed on the provider side for ROCF-START invocations received from the service user.
- h) in case the control word type is not ‘clw’, the TC VCID must not be present;

NOTE – This check shall only be performed on the provider side for ROCF-START invocations received from the service user.

- i) the update mode must match one of the entries in the permitted update mode set.

NOTE – This check shall only be performed on the provider side for ROCF-START invocations received from the service user.

**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 responds with a ROCF-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 ROCF SYNC NOTIFY

When receiving a ROCF–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 ROCF 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 `IROCF_StatusReport` and `ISLE_TransferBuffer`.

#### NOTES

- 1 The initial values of parameters that shall be set for ROCF-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 ROCF operations in addition to those specified in reference [3]:

- a) ROCF-START;
- b) ROCF-TRANSFER-DATA;
- c) ROCF-SYNC-NOTIFY;
- d) ROCF-STATUS-REPORT; and
- e) ROCF-GET-PARAMETER.

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

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

- 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 section summarizes specific obligations of a ROCF 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 IROCF\_SIAAdmin.

**3.3.2** The application shall update the 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 IROCF\_SIUpdate.

## 3.4 SEQUENCE OF DIAGNOSTIC CODES

### 3.4.1 OVERVIEW

**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 ROCF-START. This subsection 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 ROCF 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';
- e) 'invalid global VC ID';
- f) 'invalid control word type';
- g) 'invalid tc-vcid'; and
- h) 'invalid update mode'.

#### 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**

### **ROCF SPECIFIC INTERFACES**

#### **(NORMATIVE)**

##### **A1 INTRODUCTION**

This annex specifies ROCF-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 ROCF TYPE DEFINITIONS

**File**            ROCF\_Types.h

The following types have been derived from the ASN.1 module CCSDS-SLE-TRANSFER-SERVICE-ROCF-STRUCTURES in reference [2]. The source ASN.1 type is indicated in brackets. For all enumeration types a special value ‘invalid’ is defined, which is returned if the associated value in the operation object has not yet been set, or is not applicable in case of a choice.

### Antenna Id Format [AntennaId]

```
typedef enum ROCF_AntennaIdFormat
{
    rocfAF_global           = 0,
    rocfAF_local           = 1,
    rocfAF_invalid         = -1
} ROCF_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.

### ROCF Get Parameter Diagnostic [DiagnosticRocfGet]

```
typedef enum ROCF_GetParameterDiagnostic
{
    rocfGP_unknownParameter = 0,
    rocfGP_invalid         = -1
} ROCF_GetParameterDiagnostic;
```

### ROCF Start Diagnostic [DiagnosticRocfStart]

```
typedef enum ROCF_StartDiagnostic
{
    rocfSD_outOfService      = 0,
    rocfSD_unableToComply   = 1,
    rocfSD_invalidStartTime = 2,
    rocfSD_invalidStopTime  = 3,
    rocfSD_missingTimeValue = 4,
    rocfSD_invalidGvcId     = 5,
    rocfSD_invalidControlWordType = 6,
    rocfSD_invalidTcVcid    = 7,
    rocfSD_invalidUpdateMode = 8,
    rocfSD_invalid         = -1
} ROCF_StartDiagnostic;
```

### Channel Type

```
typedef enum ROCF_ChannelType
{
    rocfCT_MasterChannel    = 0,
    rocfCT_VirtualChannel   = 1,
    rocfCT_invalid         = -1
} ROCF_ChannelType;
```

**Global VCID [GlobalVcId]**

```
typedef struct ROCF_Gvcid
{
    ROCF_ChannelType    type;
    unsigned long       scid;           /* 0 to 1023 */
    unsigned long       version;       /* 0 to 3 */
    unsigned long       vcid;          /* 0 to 63 */
} ROCF_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 rocfcT\_VirtualChannel.

**Lock Status [LockStatus]**

```
typedef enum ROCF_LockStatus
{
    rocflS_inLock           = 0,
    rocflS_outOfLock        = 1,
    rocflS_notInUse         = 2, /* only for
                                sub-carrier lock */
    rocflS_unknown         = 3,
    rocflS_invalid          = -1
} ROCF_LockStatus;
```

**Notification Type [Notification]**

```
typedef enum ROCF_NotificationType
{
    rocfnT_lossFrameSync    = 0,
    rocfnT_productionStatusChange = 1,
    rocfnT_excessiveDataBacklog = 2,
    rocfnT_endOfData        = 3,
    rocfnT_invalid          = -1
} ROCF_NotificationType;
```

**Production Status [RocfProductionStatus]**

```
typedef enum ROCF_ProductionStatus
{
    rocfps_running          = 0,
    rocfps_interrupted      = 1,
    rocfps_halted           = 2,
    rocfps_invalid          = -1
} ROCF_ProductionStatus;
```

**ROCF Parameter Names [RocfGetParameter]**

```
typedef enum ROCF_ParameterName
{
    rocfpN_bufferSize       = 4,
    rocfpN_deliveryMode     = 6,
    rocfpN_latencyLimit     = 15,
    rocfpN_permittedGvcidSet = 24,
    rocfpN_permittedControlWordTypeSet = 101,
    rocfpN_permittedTcVcidSet = 102,
    rocfpN_permittedUpdateModeSet = 103,
```



```

rocFPN_reportingCycle           = 26,
rocFPN_requestedGvcid          = 28,
rocFPN_requestedControlWordType = 104,
rocFPN_requestedTcVcid         = 105,
rocFPN_requestedUpdateMode     = 106,
rocFPN_returnTimeoutPeriod     = 29,
rocFPN_invalid                 = -1
} ROCF_ParameterName;

```

Parameters that can be read using a ROCF-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 ROCF_DeliveryMode
{
    rocfdm_timelyOnline      = sleDM_rtnTimelyOnline,
    rocfdm_completeOnline   = sleDM_rtnCompleteOnline,
    rocfdm_offline          = sleDM_rtnOffline,
    rocfdm_invalid          = -1
} ROCF_DeliveryMode;

```

The delivery modes are defined as a subset of the SLE\_DeliveryMode in reference [4].

### Control Word Type

```

typedef enum ROCF_ControlWordType
{
    rocfcwt_allControlWords = 0,
    rocfcwt_clw             = 1,
    rocfcwt_notClw         = 2,
    rocfcwt_invalid        = -1
} ROCF_ControlWordType;

```

### Update Mode

```

typedef enum ROCF_UpdateMode
{
    rocfum_continuous      = 0,
    rocfum_changeBased     = 1,
    rocfum_invalid         = -1
} ROCF_UpdateMode;

```

### Virtual Channel Id

```

typedef unsigned long ROCF_TcVcid;    /* 0 to 63 */

```

## A3 ROCF OPERATION OBJECTS

### A3.1 ROCF START OPERATION

**Name** IROCF\_Start  
**GUID** {9B10BFF7-1402-4dd2-A754-001281366835}  
**Inheritance:** Iunknown - ISLE\_Operation - ISLE\_ConfirmedOperation  
**File** IROCF\_Start.H

/\* The interface provides access to the parameters of the confirmed operation ROCF- START. \*/

#### Synopsis

```
#include <ROCF_Types.h>
#include <ISLE_ConfirmedOperation.H>
interface ISLE_Time;

#define IID_IROCF_Start_DEF { 0x9b10bff7, 0x1402, 0x4dd2, \
    { 0xa7, 0x54, 0x0, 0x12, 0x81, 0x36, 0x68, 0x35 } }

interface IROCF_Start : ISLE_ConfirmedOperation
{
    virtual const ISLE_Time*
        Get_StartTime() const = 0;
    virtual const ISLE_Time*
        Get_StopTime() const = 0;
    virtual const ROCF_Gvcid*
        Get_Gvcid() const = 0;
    virtual ROCF_ControlWordType
        Get_ControlWordType() const = 0;
    virtual bool
        Get_TcVcidUsed() const = 0;
    virtual ROCF_TcVcid
        Get_TcVcid() const = 0;
    virtual ROCF_UpdateMode
        Get_UpdateMode() const = 0;
    virtual ROCF_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 ROCF_Gvcid& id ) = 0;
    virtual void
        Put_Gvcid( ROCF_Gvcid* pid ) = 0;
    virtual void
        Set_ControlWordType ( ROCF_ControlWordType type ) = 0;
    virtual void
        Set_TcVcid( ROCF_TcVcid id ) = 0;
    virtual void
        Set_UpdateMode( ROCF_UpdateMode mode ) = 0;
    virtual void
        Set_StartDiagnostic( ROCF_StartDiagnostic diagnostic ) = 0;
};
```

**Methods**

```
const ISLE_Time* Get_StartTime() const;
```

Returns the reception time of the first frame for which the OCF shall be delivered, or NULL if the parameter is not defined.

```
const ISLE_Time* Get_StopTime() const;
```

Returns the reception time of the last frame for which the OCF shall be delivered, or NULL if the parameter is not defined.

```
const ROCF_Gvcid* Get_Gvcid() const;
```

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

```
ROCF_ControlWordType Get_ControlWordType() const;
```

Returns the control word type requested by the service user, or 'invalid' if the parameter is not defined.

```
bool Get_TcVcidUsed() const;
```

Returns TRUE if a Tc Vcid is specified and FALSE otherwise.

```
ROCF_TcVcid Get_TcVcid() const;
```

Returns the Tc Vcid for which the provider shall deliver the OCFs.

Precondition: Get\_TcVcidUsed( ) returns TRUE.

```
ROCF_UpdateMode Get_UpdateMode() const;
```

Returns the update mode, which the provider shall apply for the delivery of OCFs.

```
ROCF_StartDiagnostic Get_StartDiagnostic() const;
```

Returns the value of the diagnostic code.

Precondition: 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 ROCF_Gvcid& id );
```

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

```
void Put_Gvcid( ROCF_Gvcid* pid );
```

Stores the input argument to the parameter global VCID.

```
void Set_ControlWordType( ROCF_ControlWordType type )
```

Sets the control word type requested by the service user.

```
void Set_TcVcid( ROCF_TcVcid id )
```

Sets the Tc Vcid for which the provider shall deliver the OCFs. When this method has been called, `Get_TcVcidUsed()` returns TRUE.

```
void Set_UpdateMode( ROCF_UpdateMode mode )
```

Sets the update mode to be applied by the provider for the delivery of OCFs.

```
void Set_StartDiagnostic( ROCF_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
control-word-type	'invalid'	'invalid'
TcVcId	(not used)	(not used)
update-mode	'invalid'	'invalid'
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
control-word-type	must not be 'invalid'
TC VCID	if TC VCID is set and the control word type is 'clcw' must be a value in the range 0 to 63 if the control word type is not 'clcw' TC VCID must not be used otherwise no checks are applied
update-mode	must not be 'invalid'

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

**Additional Return Codes for `VerifyInvocationArguments()`**

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 ROCF TRANSFER DATA OPERATION

**Name** IROCF\_TransferData  
**GUID** {AC88BB53-0C6A-43b3-BD06-90E88D19ACE7}  
**Inheritance:** IUnknown - ISLE\_Operation  
**File** IROCF\_TransferData.H

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

### Synopsis

```

#include <ROCF_Types.h>
#include <ISLE_Operation.H>
interface ISLE_Time;

#define IID_IROCF_TransferData_DEF { 0xac88bb53, 0xc6a, 0x43b3, \
    { 0xbd, 0x6, 0x90, 0xe8, 0x8d, 0x19, 0xac, 0xe7 } }

interface IROCF_TransferData : ISLE_Operation
{
    virtual const ISLE_Time*
        Get_EarthReceiveTime() const = 0;
    virtual ROCF_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() const = 0;
    virtual SLE_Octet*
        Remove_Data() = 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 ) = 0;
    virtual void
        Put_Data( SLE_Octet* pdata ) = 0;
};

```

## Methods

```
virtual const ISLE_Time* Get_EarthReceiveTime() const;
```

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

```
ROCF_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 rocfAF\_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 rocfAF\_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 rocfAF\_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 from 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() const;
```

Returns a pointer to the 4 bytes OCF in the object or `NULL` if the OCF has not been inserted.

```
SLE_Octet* Remove_Data();
```

Returns the 4 bytes OCF and removes it from the object. The memory allocated by the frame must be released by the client. If the parameter has not been set returns `NULL`.

```
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 );
```

Copies 4 bytes OCF data from the argument `pdata` to the parameter ‘data’.

Arguments

`pdata` pointer to the data (4 bytes)

```
void Put_Data( SLE_Octet* pdata );
```

Stores the 4 bytes OCF data argument `pdata` to the parameter ‘data’.

Arguments

`pdata` pointer to the data (4 bytes)

**Initial Values of Operation Parameters after Creation**

<b>Parameter</b>	<b>Created directly</b>	<b>Created by Service Instance</b>
earth-receive-time	NULL	NULL
antenna-id	NULL	NULL
antenna-id-format	'invalid'	'invalid'
data-link-continuity	-2	-2
frame-quality	'invalid'	'invalid'
private-annotation	NULL	NULL
data	NULL	NULL

**Checking of Invocation Parameters**

<b>Parameter</b>	<b>Required condition</b>
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 ROCF SYNC NOTIFY OPERATION

**Name** IROCF\_SyncNotify  
**GUID** {E6BFDADA-ABD9-45fa-99A8-38CEC93F0755}  
**Inheritance:** IUnknown - ISLE\_Operation  
**File** IROCF\_SyncNotify.H

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

#### Synopsis

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

#define IID_IROCF_SyncNotify_DEF { 0xe6bfdada, 0xabd9, 0x45fa, \
    { 0x99, 0xa8, 0x38, 0xce, 0xc9, 0x3f, 0x7, 0x55 } }

interface IROCF_SyncNotify : ISLE_Operation
{
    virtual ROCF_NotificationType
        Get_NotificationType() const = 0;
    virtual const ISLE_Time*
        Get_LossOfLockTime() const = 0;
    virtual ROCF_LockStatus
        Get_CarrierDemodLock() const = 0;
    virtual ROCF_LockStatus
        Get_SubCarrierDemodLock() const = 0;
    virtual ROCF_LockStatus
        Get_SymbolSyncLock() const = 0;
    virtual ROCF_ProductionStatus
        Get_ProductionStatus() const = 0;
    virtual void
        Set_LossOfFrameSync( const ISLE_Time& time,
                            ROCF_LockStatus symbolSyncLock,
                            ROCF_LockStatus subCarrierDemodLock,
                            ROCF_LockStatus carrierDemodLock ) = 0;

    virtual void
        Set_ProductionStatusChange( ROCF_ProductionStatus status ) = 0;
    virtual void
        Set_DataDiscarded() = 0;
    virtual void
        Set_EndOfData() = 0;
};
```

#### Methods

**ROCF\_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.

Precondition: Notification type is ‘loss of frame synchronization’.

```
ROCF_LockStatus Get_CarrierDemodLock() const;
```

Returns the lock status of the carrier demodulation process.

Precondition: Notification type is ‘loss of frame synchronization’.

```
ROCF_LockStatus Get_SubCarrierDemodLock() const;
```

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

Precondition: Notification type is ‘loss of frame synchronization’.

```
ROCF_LockStatus Get_SymbolSyncLock() const;
```

Returns the lock status of the symbol synchronization process.

Precondition: Notification type is ‘loss of frame synchronization’.

```
ROCF_ProductionStatus Get_ProductionStatus() const;
```

Returns the production status.

Precondition: notification type is ‘production status change’.

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

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

#### Arguments

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( ROCF_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 ROCF STATUS REPORT OPERATION

**Name** IROCF\_StatusReport  
**GUID** {8ACBB4CC-92C0-474c-8FF8-3BE34B1D15E5}  
**Inheritance:** IUnknown - ISLE\_Operation  
**File** IROCF\_StatusReport.H

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

#### Synopsis

```
#include <ROCF_Types.h>
#include <ISLE_Operation.H>

#define IID_IROCF_StatusReport_DEF { 0x8acbb4cc, 0x92c0, 0x474c, \
    {0x8f, 0xf8, 0x3b, 0xe3, 0x4b, 0x1d, 0x15, 0xe5 } }

interface IROCF_StatusReport : ISLE_Operation
{
    virtual unsigned long
        Get_NumFramesProcessed() const = 0;
    virtual unsigned long
        Get_NumOcfDelivered() const = 0;
    virtual ROCF_LockStatus
        Get_FrameSyncLock() const = 0;
    virtual ROCF_LockStatus
        Get_CarrierDemodLock() const = 0;
    virtual ROCF_LockStatus
        Get_SubCarrierDemodLock() const = 0;
    virtual ROCF_LockStatus
        Get_SymbolSyncLock() const = 0;
    virtual ROCF_ProductionStatus
        Get_ProductionStatus() const = 0;
    virtual void
        Set_NumFramesProcessed( unsigned long count ) = 0;
    virtual void
        Set_NumOcfDelivered( unsigned long count ) = 0;
    virtual void
        Set_FrameSyncLock( ROCF_LockStatus status ) = 0;
    virtual void
        Set_CarrierDemodLock( ROCF_LockStatus status ) = 0;
    virtual void
        Set_SubCarrierDemodLock( ROCF_LockStatus status ) = 0;
    virtual void
        Set_SymbolSyncLock( ROCF_LockStatus status ) = 0;
    virtual void
        Set_ProductionStatus( ROCF_ProductionStatus status ) = 0;
};
```

#### Methods

**unsigned long Get\_NumFramesProcessed() const;**

Returns the total number of telemetry frames that have been processed for extracting OCFs.

```
unsigned long Get_NumOcfDelivered() const;
```

Returns the total number of OCFs delivered to the user.

```
ROCF_LockStatus Get_FrameSyncLock() const;
```

Returns the lock status of the frame synchronization process.

```
ROCF_LockStatus Get_CarrierDemodLock() const;
```

Returns the lock status of the carrier demodulation process.

```
ROCF_LockStatus Get_SubCarrierDemodLock() const;
```

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

```
ROCF_LockStatus Get_SymbolSyncLock() const;
```

Returns the lock status of the symbol synchronization process.

```
ROCF_ProductionStatus Get_ProductionStatus() const;
```

Returns the production status.

```
void Set_NumFramesProcessed( unsigned long count );
```

Sets the total number of telemetry frames that have been processed for extracting OCFs.

```
void Set_NumOcfDelivered( unsigned long count );
```

Sets the total number of OCFs delivered to the user.

```
void Set_FrameSyncLock( ROCF_LockStatus status );
```

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

```
void Set_CarrierDemodLock( ROCF_LockStatus status );
```

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

```
void Set_SubCarrierDemodLock( ROCF_LockStatus status );
```

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



```
void Set_SymbolSyncLock( ROCF_LockStatus status );
```

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

```
void Set_ProductionStatus( ROCF_ProductionStatus status );
```

Sets the production status as defined by the argument.

### Initial Values of Operation Parameters after Creation

Parameter	Created directly
number-of-frames-processed	0
number-of-ocfs-delivered	0
frame-sync-lock-status	'invalid'
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 ROCF GET PARAMETER OPERATION

**Name** IROCF\_GetParameter  
**GUID** {7B015634-CA29-4464-B420-4047E5FCA4E8}  
**Inheritance:** IUnknown - ISLE\_Operation - ISLE\_ConfirmedOperation  
**File** IROCF\_GetParameter.H

This interface provides access to the parameters of the confirmed operation ROCF-GET-PARAMETER.

#### Synopsis

```
#include <ROCF_Types.h>
#include <ISLE_ConfirmedOperation.H>

#define IID_IROCF_GetParameter_DEF { 0x7b015634, 0xca29, 0x4464, \
    { 0xb4, 0x20, 0x40, 0x47, 0xe5, 0xfc, 0xa4, 0xe8 } }

interface IROCF_GetParameter : ISLE_ConfirmedOperation
{
    virtual ROCF_ParameterName
        Get_RequestedParameter() const = 0;
    virtual ROCF_ParameterName
        Get_ReturnedParameter() const = 0;
    virtual ROCF_DeliveryMode
        Get_DeliveryMode() const = 0;
    virtual unsigned short
        Get_LatencyLimit() const = 0;
    virtual unsigned long
        Get_TransferBufferSize() const = 0;
    virtual const ROCF_Gvcid*
        Get_RequestedGvcid() const = 0;
    virtual const ROCF_Gvcid*
        Get_PermittedGvcidSet( size_t& size ) const = 0;
    virtual ROCF_Gvcid*
        Remove_PermittedGvcidSet( size_t& size ) = 0;
    virtual ROCF_ControlWordType
        Get_RequestedControlWordType() const = 0;
    virtual const ROCF_ControlWordType*
        Get_PermittedControlWordTypeSet( size_t& size ) const = 0;
    virtual ROCF_ControlWordType*
        Remove_PermittedControlWordTypeSet( size_t& size ) = 0;
    virtual bool
        Get_TcVcidUsed() const = 0;
    virtual ROCF_TcVcid
        Get_RequestedTcVcid() const = 0;
    virtual const ROCF_TcVcid*
        Get_PermittedTcVcidSet( size_t& size ) const = 0;
    virtual ROCF_TcVcid*
        Remove_PermittedTcVcidSet( size_t& size ) = 0;
    virtual ROCF_UpdateMode
        Get_RequestedUpdateMode() const = 0;
    virtual const ROCF_UpdateMode*
        Get_PermittedUpdateModeSet( size_t& size ) const = 0;
    virtual ROCF_UpdateMode*
        Remove_PermittedUpdateModeSet( size_t& size ) = 0;
    virtual unsigned long
```

```

    Get_ReportingCycle() const = 0;
virtual unsigned long
    Get_ReturnTimeoutPeriod() const = 0;
virtual ROCF_GetParameterDiagnostic
    Get_GetParameterDiagnostic() const = 0;
virtual void
    Set_RequestedParameter( ROCF_ParameterName name ) = 0;
virtual void
    Set_DeliveryMode( ROCF_DeliveryMode mode ) = 0;
virtual void
    Set_LatencyLimit( unsigned short limit ) = 0;
virtual void
    Set_TransferBufferSize( unsigned long size ) = 0;
virtual void
    Set_RequestedGvcid( const ROCF_Gvcid* id ) = 0;
virtual void
    Put_RequestedGvcid( ROCF_Gvcid* pid ) = 0;
virtual void
    Set_PermittedGvcidSet( const ROCF_Gvcid* idSet,
                          size_t size ) = 0;
virtual void
    Put_PermittedGvcidSet( ROCF_Gvcid* idSet,
                          size_t size ) = 0;
virtual void
    Set_RequestedControlWordType( ROCF_ControlWordType type ) = 0;
virtual void
    Set_PermittedControlWordTypeSet( const ROCF_ControlWordType* typeSet,
                                     size_t& size ) = 0;
virtual void
    Put_PermittedControlWordTypeSet( ROCF_ControlWordType* typeSet,
                                     size_t& size ) = 0;
virtual void
    Set_RequestedTcVcid( ROCF_TcVcid id ) = 0;
virtual void
    Set_PermittedTcVcidSet( const ROCF_TcVcid* idSet,
                          size_t& size ) = 0;
virtual void
    Put_PermittedTcVcidSet( ROCF_TcVcid* idSet,
                          size_t& size ) = 0;
virtual void
    Set_RequestedUpdateMode( ROCF_UpdateMode mode ) = 0;
virtual void
    Set_PermittedUpdateModeSet( const ROCF_UpdateMode* modeSet,
                                size_t& size ) = 0;
virtual void
    Put_PermittedUpdateModeSet( ROCF_UpdateMode* modeSet,
                                size_t& size ) = 0;
virtual void
    Set_ReportingCycle( unsigned long cycle ) = 0;
virtual void
    Set_ReturnTimeoutPeriod( unsigned long period ) = 0;
virtual void
    Set_GetParameterDiagnostic( ROCF_GetParameterDiagnostic
                               diagnostic ) = 0;
};

```

**Methods**

```
ROCF_ParameterName Get_RequestedParameter() const;
```

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

```
ROCF_ParameterName Get_ReturnedParameter() const;
```

Returns the identification of the parameter whose value is reported.

```
ROCF_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 `ROCF-TRANSFER-DATA` invocations and `ROCF-SYNC-NOTIFY` invocations that can be stored in the buffer.

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

```
const ROCF_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 ROCF_Gvcid* Get_PermittedGvcidSet( size_t& size ) const;
```

Returns the list of global VCIDs to which the service instance has access. 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-list`.

```
ROCF_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-list.

```
ROCF_ControlWordType Get_RequestedControlWordType() const;
```

Returns the control word type requested by the user.

Precondition: the returned parameter is control-word-type.

```
const ROCF_ControlWordType*
  Get_PermittedControlWordTypeSet( size_t& size ) const;
```

Returns the set of control word types to which the service instance has access. If the parameter has not been set, or the control word type set has been removed, returns a NULL pointer.

Precondition: the returned parameter is permitted-control-word-type-set.

```
ROCF_ControlWordType* Remove_PermittedControlWordTypeSet( size_t&
size );
```

Returns the set of control word types to which the service instance has access and removes the set from the object. If the parameter has not been set or the permitted control word type set has been removed, returns a NULL pointer.

Precondition: the returned parameter is permitted-control-word-type-set.

```
bool Get_TcVcidUsed() const;
```

Returns TRUE if a Tc Vcid is specified and FALSE otherwise.

```
ROCF_TcVcid Get_RequestedTcVcid() const;
```

Returns the TcVcid requested by the user. This function shall only be called when Get\_TcVcidUsed() returns TRUE, otherwise the returned value is undefined.

Precondition: the returned parameter is 'requested TcVcid'.

```
const ROCF_TcVcid* Get_PermittedTcVcidSet( size_t& size ) const;
```

Returns the set of TcVcids to which the service instance has access or NULL if either the permitted-control-word-type is not 'clw' or the CLCW for any TC VC shall be returned. If the parameter has not been set, or the TcVcid set has been removed, returns a NULL pointer.

Precondition: the returned parameter is ‘permitted TcVcid set’.

```
ROCF_TcVcid* Remove_PermittedTcVcidSet( size_t& size );
```

Returns the set of TcVcids to which the service instance has access, or NULL if either the permitted-control-word-type is not ‘clcw’ or the CLCW for any TC VC shall be returned, and removes the set from the object. If the parameter has not been set or the permitted TcVcid set has been removed, returns a NULL pointer.

Precondition: the returned parameter is ‘permitted TcVcid set’.

```
ROCF_UpdateMode Get_RequestedUpdateMode() const;
```

Returns the update mode requested by the user.

Precondition: the returned parameter is requested-update-mode.

```
const ROCF_UpdateMode* Get_PermittedUpdateModeSet( size_t& size )
const;
```

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

Precondition: the returned parameter is permitted-update-mode-set.

```
ROCF_UpdateMode* Remove_PermittedUpdateModeSet( size_t& size )
```

Returns the set of update modes to which the service instance has access and removes the set from the object. If the parameter has not been set or the permitted update mode set has been removed, returns a NULL pointer.

Precondition: the returned parameter is permitted-update-mode-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.

```
ROCF_GetParameterDiagnostic Get_GetParameterDiagnostic() const;
```

Returns the diagnostic code.

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

```
void Set_RequestedParameter( ROCF_ParameterName name );
```

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

```
void Set_DeliveryMode( ROCF_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 ROCF_Gvcid* id );
```

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

```
void Put_RequestedGvcid( ROCF_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 ROCF_Gvcid* idSet, size_t size );
```

Sets the returned parameter name to `permitted-global-VCID-list` and copies its value from the argument.

```
void Put_PermittedGvcidSet( ROCF_Gvcid* idSet, size_t size );
```

Sets the returned parameter name to `permitted-global-VCID-list` and stores the argument as the value of this parameter.

```
void Set_RequestedControlWordType( ROCF_ControlWordType type )
```

Sets the returned parameter name to `requested-control-word-type` and the value as defined by the argument.

```
void Set_PermittedControlWordTypeSet( const ROCF_ControlWordType*  
                                     typeSet, size_t size );
```

Sets the returned parameter name to `permitted-control-word-type` and copies its value from the argument.

```
void Put_PermittedControlWordTypeSet( ROCF_ControlWordType* typeSet,  
                                     size_t size );
```

Sets the returned parameter name to `permitted-control-word-type-set` and stores the argument as the value of this parameter.

```
void Set_RequestedTcVcid( ROCF_TcVcid id );
```

Sets the returned parameter name to `requested-TcVcid` and the value as defined by the argument. If this method has been called, `Get_TcVcidUsed()` returns TRUE.

```
void Set_PermittedTcVcidSet( const ROCF_TcVcid* idSet, size_t size );
```

Sets the returned parameter name to `permitted-TcVcid-set` and copies its value from the argument. The parameter `idSet` may be set to NULL to indicate either that the permitted-control-word-type is not 'clw' or that the CLCW from any TC VC shall be delivered.

### Arguments

`idSet` array of TCVC identifiers where each identifier must be in the range 0 to 63, or NULL

`size` number of identifiers in the array

```
void Put_PermittedTcVcidSet( ROCF_TcVcid* idSet, size_t size );
```

Sets the returned parameter name to `permitted-TcVcid-set` and stores the argument as the value of this parameter. The parameter `idSet` may be set to NULL to indicate either that the permitted-control-word-type is not 'clw' or that the CLCW from any TC VC shall be delivered.



Arguments

`idSet` array of TCVC identifiers where each identifier must be in the range 0 to 63, or NULL

`size` number of identifiers in the array

```
void Set_RequestedUpdateMode( ROCF_UpdateMode mode );
```

Sets the returned parameter name to `requested-update-mode` and the value as defined by the argument.

```
void Set_PermittedUpdateModeSet( const ROCF_UpdateMode* modeSet,
                                size_t size );
```

Sets the returned parameter name to `permitted-update-mode` and copies its value from the argument.

```
void Put_PermittedUpdateModeSet( ROCF_UpdateMode* modeSet, size_t size );
```

Sets the returned parameter name to `requested-update-mode-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( ROCF_GetParameterDiagnostic diagnostic
                                );
```

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

**Initial Values of Operation Parameters after Creation**

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-list	NULL	NULL
requested-control-word-type	'invalid'	'invalid'
permitted-control-word-type-set	NULL	NULL
requested-TC-VCID	(not used)	(not used)
permitted-TC-VCID-set	NULL	NULL
requested-update-mode	'invalid'	'invalid'
permitted-update-mode-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 <code>delivery-mode</code> must not be 'invalid'
transfer-buffer-size	If the returned parameter is <code>transfer-buffer-size</code> must not be 0
requested-global-VCID	if the returned parameter is <code>requested-global-VCID</code> must either be NULL or must have the following structure
type	must not be 'invalid'

Parameter	Required condition
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-list	if the returned parameter is permitted-global-VCID-list must not be NULL
permitted-control-word-type-set	if the returned parameter is permitted-control-word-type-set must not be NULL
permitted-TC-VCID-set	if not NULL, each element in the list must be in the range 0 to 63; if the returned parameter is 'permitted-TC-VCID-list', a NULL value indicates either that the CLCW for any TC VCID shall be reported, or that the control word type is not 'clcw'.
permitted-update-mode-set	if the returned parameter is permitted-update-mode-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 ROCF 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 ROCF SERVICE INSTANCE INTERFACES

### A4.1 SERVICE INSTANCE CONFIGURATION

**Name** IROCF\_SIAdmin  
**GUID** {54D0A215-52D9-490e-9BF8-BAEC667F5E45}  
**Inheritance:** IUnknown  
**File** IROCF\_SIAdmin.H

The interface provides write and read access to the ROCF-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 shall check 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 <ROCF_Types.h>
#include <SLE_SCM.H>

#define IID_IROCF_SIAdmin_DEF { 0x54d0a215, 0x52d9, 0x490e, \
    { 0x9b, 0xf8, 0xba, 0xec, 0x66, 0x7f, 0x5e, 0x45 } }

interface IROCF_SIAdmin : IUnknown
{
    virtual void
        Set_DeliveryMode( ROCF_DeliveryMode mode ) = 0;
    virtual void
        Set_LatencyLimit( unsigned short limit ) = 0;
    virtual void
        Set_TransferBufferSize( unsigned long size ) = 0;
    virtual void
        Set_PermittedGvcidSet( const ROCF_Gvcid* idSet,
                               size_t size ) = 0;
    virtual void
        Set_PermittedControlWordTypeSet( const ROCF_ControlWordType* typeSet,
                                           size_t size ) = 0;
    virtual void
        Set_PermittedTcVcidSet( const ROCF_TcVcid* idSet,
                                 size_t size ) = 0;
    virtual void
        Set_PermittedUpdateModeSet( const ROCF_UpdateMode* modeSet,
                                     size_t size ) = 0;
    virtual void
        Set_InitialProductionStatus( ROCF_ProductionStatus status ) = 0;
    virtual void
        Set_InitialFrameSyncLock( ROCF_LockStatus status ) = 0;
```

```

virtual void
    Set_InitialCarrierDemodLock( ROCF_LockStatus status ) = 0;
virtual void
    Set_InitialSubCarrierDemodLock( ROCF_LockStatus status ) = 0;
virtual void
    Set_InitialSymbolSyncLock( ROCF_LockStatus status ) = 0;
virtual ROCF_DeliveryMode
    Get_DeliveryMode() const = 0;
virtual unsigned short
    Get_LatencyLimit() const = 0;
virtual unsigned long
    Get_TransferBufferSize() const = 0;
virtual const ROCF_Gvcid*
    Get_PermittedGvcidSet( size_t& size ) const = 0;
virtual const ROCF_ControlWordType*
    Get_PermittedControlWordTypeSet( size_t& size ) const = 0;
virtual const ROCF_TcVcid*
    Get_PermittedTcVcidSet( size_t& size ) const = 0;
virtual const ROCF_UpdateMode*
    Get_PermittedUpdateModeSet( size_t& size ) const = 0;
};

```

## Methods

```
void Set_DeliveryMode( ROCF_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 ROCF-TRANSFER-DATA invocations and ROCF-SYNC-NOTIFY invocations that shall be stored in one transfer buffer PDU.

```
void Set_PermittedGvcidSet( const ROCF_Gvcid* idSet, size_t size );
```

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

```
void Set_InitialProductionStatus( ROCF_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( ROCF_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( ROCF_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( ROCF_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_InitialSymbolsSyncLock( ROCF_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.

```
ROCF_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 ROCF_Gvcid* Get_PermittedGvcidSet( size_t& size ) const;
```

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

## A4.2 UPDATE OF SERVICE INSTANCE PARAMETERS

**Name** IROCF\_SIUpdate  
**GUID** {638A73E6-7FE6-11d3-9F15-00104B4F22C0}  
**Inheritance:** IUnknown  
**File** IROCF\_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 <ROCF_Types.h>
#include <SLE_SCM.H>

#define IID_IROCF_SIUpdate_DEF { 0x638a73e6, 0x7fe6, 0x11d3, \
    { 0x9f, 0x15, 0x0, 0x10, 0x4b, 0x4f, 0x22, 0xc0 } }

interface IROCF_SIUpdate : IUnknown
{
    virtual void
        Set_ProductionStatus( ROCF_ProductionStatus status ) = 0;
    virtual void
        Set_FrameSyncLock( ROCF_LockStatus status ) = 0;
    virtual void
        Set_CarrierDemodLock( ROCF_LockStatus status ) = 0;
    virtual void
        Set_SubCarrierDemodLock( ROCF_LockStatus status ) = 0;
    virtual void
        Set_SymbolSyncLock( ROCF_LockStatus status ) = 0;
    virtual ROCF_ProductionStatus
        Get_ProductionStatus() const = 0;
    virtual ROCF_LockStatus
        Get_FrameSyncLock() const = 0;
    virtual ROCF_LockStatus
        Get_CarrierDemodLock() const = 0;
    virtual ROCF_LockStatus
        Get_SubCarrierDemodLock() const = 0;
    virtual ROCF_LockStatus
        Get_SymbolSyncLock() const = 0;
    virtual unsigned long
```

```

    Get_NumFrames() const = 0;
    virtual ROCF_Gvcid*
        Get_RequestedGvcid() const = 0;
};

```

## Methods

```
void Set_NumFramesProcessed( unsigned long count );
```

Sets the total number of telemetry frames that have been processed for extracting OCFs, since the start of the service instance provision period.

```
void Set_ProductionStatus( ROCF_ProductionStatus status );
```

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

```
void Set_FrameSyncLock( ROCF_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( ROCF_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( ROCF_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( ROCF_LockStatus status );
```

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

```
ROCF_ProductionStatus Get_ProductionStatus() const;
```

Returns the value of the production status.

```
ROCF_LockStatus Get_FrameSyncLock() const;
```

Returns the lock status of the frame synchronization process.

```
ROCF_LockStatus Get_CarrierDemodLock() const;
```

Returns the lock status of the carrier demodulation process.



```
ROCF_LockStatus Get_SubCarrierDemodLock() const;
```

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

```
ROCF_LockStatus Get_SymbolSyncLock() const;
```

Returns the lock status of the symbol synchronization process.

```
unsigned long Get_NumFramesProcessed() const;
```

Returns the total number of telemetry frames that have been processed for extracting OCFs, since the start of the service instance provision period.

```
unsigned long Get_NumOcfDelivered() const;
```

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

```
ROCF_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.

```
ROCF_ControlWordType Get_RequestedControlWordType() const;
```

Returns the control word type requested by the user, or 'invalid' if the service instance is not in the state 'active'.

```
bool Get_TcVcidUsed() const;
```

Returns TRUE if a Tc Vcid has been specified in the previous ROCF-START invocation, and FALSE otherwise.

```
ROCF_TcVcid Get_RequestedTcVcid() const;
```

Returns the TcVcid requested by the user. This function shall only be called when `Get_TcVcidUsed()` returns TRUE, otherwise the returned value is undefined.

```
ROCF_UpdateMode Get_RequestedUpdateMode() const;
```

Returns the update mode requested by the user, or 'invalid' if the service instance is not in the state 'active'.

**Initial Parameter Values**

<b>Parameter</b>	<b>Value</b>
production-status	initial production status set via the interface IROCF_SIAdmin, in the delivery mode 'offline' set to 'invalid' if not set via IROCF_SIAdmin
frame-synchronizer-lock	initial frame synchronizer lock set via the interface IROCF_SIAdmin, in the delivery mode 'offline' set to 'unknown' if not set via IROCF_SIAdmin
symbol-synchronizer-lock	initial symbol synchronizer lock set via the interface IROCF_SIAdmin, in the delivery mode 'offline' set to 'unknown' if not set via IROCF_SIAdmin
subcarrier-demodulator-lock	initial sub-carrier demodulator lock set via the interface IROCF_SIAdmin, in the delivery mode 'offline' set to 'unknown' if not set via IROCF_SIAdmin
carrier-demodulator-lock	'initial carrier demodulator lock set via the interface IROCF_SIAdmin, in the delivery mode 'offline' set to 'unknown' if not set via IROCF_SIAdmin
number-of-frames-processed	0
number-of-ocfs-delivered	0
requested-global-VCID	NULL
requested-control-word-type	'invalid'
requested-TC-VCID	(not used)
requested-update-mode	'invalid'

## **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
GVCID	Global Virtual Channel Identifier
MC	Master Channel
PDU	Protocol Data Unit
ROCF	Return Operational Control Fields
SLE	Space Link Extension
UML	Unified Modeling Language
VC	Virtual Channel

## ANNEX C

## INFORMATIVE REFERENCES

## (INFORMATIVE)

- [C1] *Organization and Processes for the Consultative Committee for Space Data Systems*. Issue 4. CCSDS Record (Yellow Book), CCSDS A02.1-Y-4. Washington, D.C.: CCSDS, April 2014.
- [C2] *Cross Support Concept—Part 1: Space Link Extension Services*. Issue 3. Report Concerning Space Data System Standards (Green Book), CCSDS 910.3-G-3. Washington, D.C.: CCSDS, March 2006.
- [C3] *Space Link Extension—Internet Protocol for Transfer Services*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 913.1-B-2. Washington, D.C.: CCSDS, September 2015.
- [C4] *Space Link Extension—Application Program Interface for Transfer Services—Summary of Concept and Rationale*. Issue 1. Report Concerning Space Data System Standards (Green Book), CCSDS 914.1-G-1. Washington, D.C.: CCSDS, January 2006.
- [C5] *Space Link Extension—Application Program Interface for Transfer Services—Application Programmer's Guide*. Issue 2. Report Concerning Space Data System Standards (Green Book), CCSDS 914.2-G-2. Washington, D.C.: CCSDS, October 2008.
- [C6] *The COM/DCOM Reference*. COM/DCOM Product Documentation, AX-01. San Francisco: The Open Group, 1999.
- [C7] *Unified Modeling Language (UML)*. Version 2.4.1. Needham, Massachusetts: Object Management Group, August 2011.
- [C8] *Space Link Extension—Return Operational Control Fields Service Specification*. Issue 1-S. Recommendation for Space Data System Standards (Historical), CCSDS 911.5-B-1-S. Washington, D.C.: CCSDS, (December 2004) January 2010.