INTERNATIONAL STANDARD

ISO
20242-2

First edition
2010-09-01

Industrial automation systems and integration — Service interface for testing applications —

Part 2:
Resource management service interface

Systèmes d'automatisation industrielle et intégration — Interface de service pour contrôler les applications —

Partie 2: Interface de service pour la gestion de ressource

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 20242-2 was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 5, *Architecture, communications and integration frameworks*.

ISO 20242 consists of the following parts, under the general title *Industrial automation systems and integration — Service interface for testing applications*:

— *Part 1: Overview*

— *Part 2: Resource management service interface*

The following parts are planned:

— *Part 3: Virtual device service interface*

— *Part 4: Device capability profile template*

— *Part 5: Application program service interface*

— *Part 6: Conformance test methods, criteria and reports*

Introduction

The motivation for ISO 20242 stems from international automotive industries and their suppliers to facilitate the integration of automation and measurement devices, and other peripheral components for this purpose, into computer-based applications. It defines rules for the construction of device drivers and their behaviour in the context of an automation application, or a measurement application, or an automation and measurement application.

The main goal of ISO 20242 is to provide users with:

— independence from the computer operating system;

— independence from the device connection technology (device interface/network);

— independence from device suppliers;

— the ability to certify device drivers with connected devices and their behaviour in the context of a given computer platform;

— independence from the technological device development in the future.

ISO 20242 will not force the development of new device families or the use of special interface technologies (networks). It encapsulates a device and its communication interface to make it compatible with other devices of that kind for a given application.

Industrial automation systems and integration — Service interface for testing applications —

Part 2:
Resource management service interface

1 Scope

This part of ISO 20242 defines a service interface that provides a generic service access point for managing and operating the resources supported by the operating system of a computer and its peripherals, including special hardware on plug-in boards that are used in computer-assisted testing applications. The resource management service interface is intended to be implemented in a manner that offers the exposed services of a computing platform adapter to be generic and independent of the operating system and its communication interfaces.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 20242-1, *Industrial automation systems and integration — Service interface for testing applications — Part 1: Overview*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 20242-1 and the following apply.

3.1
device driver
software module providing an ISO 20242-specified interface with service functions to call a platform adapter to access physical devices

3.2
platform adapter
software module providing a resource management service interface as defined in this part of ISO 20242, which encapsulates the computer platform, including the operating system, the hardware and its peripherals

4 Symbols and abbreviated terms

CNF, Cnf   Confirm (service primitive)

IND, Ind   Indication (service primitive)

REQ, Req   Request (service primitive)

RMS   Resource Management Services

RMSI   Resource Management Service Interface

RSP, Rsp   Response (service primitive)

SAP   Service Access Point

5   Conventions for service definitions and procedures

5.1   General

This part of ISO 20242 uses the descriptive conventions given in ISO/IEC 10731.

The interface between the user of RMS and the provider of RMS is described by service primitives that convey parameters. Since data transmission aspects are outside the scope of ISO 20242, only the request and confirm primitives are used to describe events occurring at the RMS service provider. Indication and response primitives are used to handle events occurring at the RMS service provider. The service model, service primitives and sequence diagrams are abstract descriptions; they do not represent a specification for implementation.

Annex A contains rules for example implementations.

5.2   Parameters

Service primitives, used to represent service user/provider interactions (see ISO/IEC 10731), convey parameters that indicate information used and exchanged in these interactions.

This part of ISO 20242 uses a tabular format to describe the component parameters of the RMS primitives, as shown in Table 1. The parameters that apply to each group of RMS primitives are set out in tables throughout the remainder of this part of ISO 20242. Each table consists of three columns, where the first column contains the name of the service parameter, the second column contains the input parameters of either the request or indication primitives, and the third column contains the output parameters of either the confirm or response primitives.

One parameter (or part of it) is listed in each row of each table. Under the appropriate service primitive columns, the following codes are used to specify the type of usage of the parameter on the primitive and parameter direction specified in the column:

a)   M: parameter is mandatory for the primitive;

b)   C: parameter is conditional upon other parameters or upon RMS capabilities;

c)   S: parameter is a selected item;

d)   (blank): parameter is not conveyed by the RMS user or the RMS provider.

Table 1 — Tabular format for service primitive parameters

Parameter name	REQ or IND	CNF or RSP
Argument	M	
Parameter 1	M	
Parameter 2	C	
Result (+)		S
Parameter 3		M
Parameter 4		C
Result (-)		S
Parameter 5		M

5.3   Service procedures

5.3.1   RMS confirmed services

An RMS user submits a request primitive to the RMSI. It is implied that the service access point (SAP) exists. The corresponding service processing entity delivers a confirmation primitive to the user after all necessary interactions are finished or an error occurred.

5.3.2   RMS event handling

The user creates a service access point (SAP) at RMSI for handling events. An event is signalled with an indication primitive at this access point. The user of RMSI issues a response primitive after all necessary interactions are finished or an error occurred (see Figure 1).

Figure 1 — Handling local events with RMS
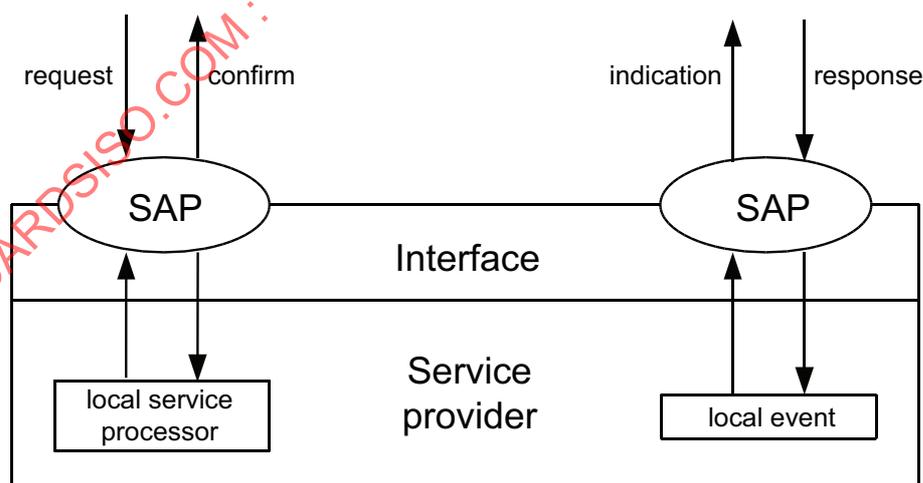
5.4   Service primitives and state diagrams

If needed, UML state diagrams are used to describe the behaviour of RMS. In such diagrams, only the service name is used to describe a state transition where no explicit state between request and confirm primitives is necessary [see Figure 2 b)]. Otherwise an extra state of processing the service is denoted [see Figure 2 a)].

A)

B)
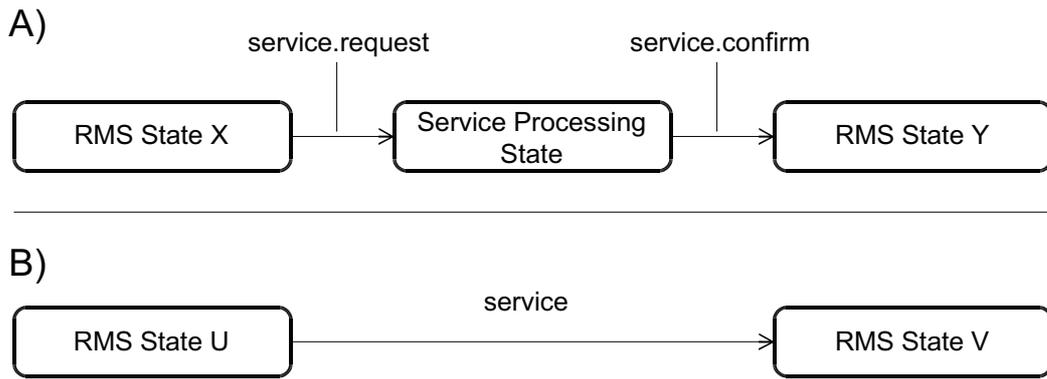
Figure 2 — State transitions caused by services

6 Resource Management Services

6.1 Overview

The RMSI shall provide generic management support services, generic operating support services, and generic input/output services.

The input/output services access another subjacent layer providing extended services. Extended services are introduced to describe the structure of loadable resources for different kinds of periphery interfaces (see Figure 3).

NOTE 1   ISO 20242 does not define the methods for integrating entities with extended services into the RMS provider, as that will depend on the computer operating system and the programming language used for implementing service providers. However, the extended services need to be described to enable the extension of input/output services for different peripheral interfaces without changing the RMS provider. See Annex A for an implementation example.

NOTE 2   There are additional cascading methods described in Annex B for using the RMSI in more complex structures of device and equipment integration.

Figure 3 — Service users and providers at the RMSI

6.2   List of services

6.2.1   Generic management support services

Generic management support services are used for handling the access to other services and for initiating (and loading, if necessary) extended service providers. Table 2 gives an overview of these services.

Table 2 — Generic management support services

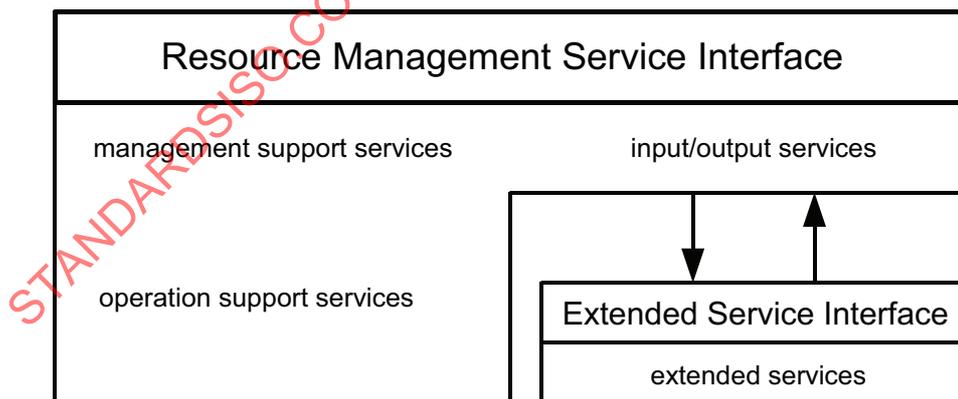| Service	Name for identification	Remarks
Get Service Reference	getFuncAddress	Get the reference for a service by identifier (name and/or number) and version number.
Initiate Periphery Interface Type	io_initiate	Get the identifier for a specified interface type and load an extended service provider for this type (if necessary).
Conclude Periphery Interface Type	io_conclude	Release a type identifier and close the extended service provider for this type (if existing).

6.2.2   Generic input/output services

Generic input/output services are used for communication with real devices and for configuration and control of peripheral interfaces. Table 3 gives an overview of these services.

Table 3 — Generic input/output services

Service	Name for identification	Remarks
Open Periphery Interface Channel	io_open	Open a peripheral interface for data transmission and configure the interface.
Reconfigure Periphery Interface Channel	io_config	Change the configuration of an interface without closing it, e.g. change transmission parameters.
Read Data	io_read	Fetch received data at a peripheral interface.
Write Data	io_write	Deliver data to a peripheral interface for transmitting.
Execute Operation	io_execute	Execute an operation belonging to a peripheral interface channel. This is comparable to handling both read and write data with one service (data exchange).
Cancel Communication	io_cancel	Cancel a Read Data, Write Data or Execute Operation service and prepare the interface for new requests.
Get Periphery Interface Channel Status	io_stat	Investigate the status of a peripheral interface.
Clear Read Buffer	io_clear	Delete the contents of the input buffer of a peripheral interface.
Close Peripheral Interface Channel	io_close	Close a peripheral interface.
Signal Event	io_event	Indicating a local event and responding to the event source.

Generic input/output services are transferred to corresponding extended services (see Table 4) if an extended service provider is loaded for the specified type of interface.

6.2.3 Extended services

Extended services are not visible to the user of the RMSI; they are defined in this part of ISO 20242 to enable a hierarchical modular structure of RMS implementation by using extended service providers. These extended services are substantially the same as the generic input/output services of RMS.

Table 4 — Extended services for peripheral interfaces

Service	Name for identification	Remarks
Initiate Extended Interface Type	ext_initiate	Set the identifier for a specified peripheral interface type.
Conclude Extended Interface Type	ext_conclude	Release the type identifier of ext_initiate.
Open Extended Interface	ext_open	Open a peripheral interface for data transmission and configure the interface.
Reconfigure Extended Interface	ext_config	Change the configuration of a peripheral interface without closing it, e.g. change transmission parameters.
Read Extended Interface Data	ext_read	Fetch received data at a peripheral interface.
Write Extended Interface Data	ext_write	Deliver data to a peripheral interface for transmitting.
Execute Extended Interface Operation	ext_execute	Execute an operation belonging to a peripheral interface. This is comparable to handling read and write data with one service (data exchange).
Cancel Extended Communication	ext_cancel	Cancel a Read Extended Interface Data, Write Extended Interface Data or Execute Extended Interface Operation service and prepare the interface for new requests.
Get Extended Interface Status	ext_stat	Investigate the status of an interface.
Clear Extended Interface Read Buffer	ext_clear	Delete the contents of the input buffer of a peripheral interface.
Close Extended Interface	ext_close	Close a peripheral interface.
Signal Extended Event	ext_event	Indicating an extended event and awaiting a response.

6.2.4 Operating support services

Operating support services (see Table 5) provide access to memory, timer control, semaphores and other resources of the computer operating system.

Table 5 — Operating support services

Service	Name for identification	Remarks
Allocate Memory	os_allocate	Allocate coherent data space of specified size.
Reallocate Memory	os_reallocate	Change size of allocated data space.
Free Memory	os_free	Release allocated data space.
Get Time	os_time	Investigate the local time.
Get Process Time	os_clock	Investigate the CPU-time for a process.
Wait	os_delay	Temporize a specified amount of time.
Create Timer	os_settimer	Create and start a timer.
Signal Timer Event	os_timerEvent	Indicating that a timer elapsed and awaiting a response.
Remove Timer	os_killtimer	Stop and remove a timer.
Create Light Process Timer	os_setLPtimer	Create and start a light process timer; resolution and accuracy depend on the light process.
Signal Light Process Timer Event	os_LPtimerEvent	Indicating that a light process timer elapsed and awaiting a response.
Remove Light Process Timer	os_killLPtimer	Stop and delete a light process timer.
Identify Light Process	os_getLPnumber	Identify the actual light process.
Create Counted Semaphore	os_createSem	Create a counted semaphore to control multiple concurrent use of resources.
Wait for Counted Semaphore	os_waitSem	Wait for a free access to a protected resource.
Release Counted Semaphore	os_releaseSem	Release the access to a protected resource.
Delete Counted Semaphore	os_deleteSem	Delete a counted semaphore.
Create Private Semaphore	os_createMutex	Create a private semaphore to control access to resources by different light processes with mutual exclusion.
Wait for Private Semaphore	os_waitMutex	Wait for a free access to a protected resource.
Release Private Semaphore	os_releaseMutex	Release the access to a protected resource.
Delete Private Semaphore	os_deleteMutex	Delete a private semaphore.
Open Debug Log	os_openDebug	Open a text log for debug messages.
Write Debug Message	os_writeDebug	Send message to text log.
Close Debug Log	os_closeDebug	Close a text log.

6.3   Management support services

6.3.1   Get Service Reference service

6.3.1.1   Service overview

The Get Service Reference service is used to get a reference for other version-dependent resource management services. This service is requested by the RMS user for each resource management service that is needed for an application.

6.3.1.2 Service parameter structure

The service parameters for the Get Service Reference service are shown in Table 6.

Table 6 — Get Service Reference parameter structure

Parameter name	Req	Cnf
Argument	M	
Service identifier (name)	M	
Proposed version number	M	
Result (+)		S
Service reference		M
Result (-)		S

6.3.1.3 Service parameters

6.3.1.3.1 Argument

The argument contains the parameters of the service request.

6.3.1.3.2 Service identifier

This parameter identifies the service for which the reference is requested.

6.3.1.3.3 Proposed version number

This parameter specifies the version which the RMS user requests for this service.

6.3.1.3.4 Result (+)

This selection type parameter indicates that the service request succeeded.

6.3.1.3.5 Service reference

This parameter contains a reference to identify the service of the proposed version number.

6.3.1.3.6 Result (-)

This selection type parameter indicates that the service request failed.

6.3.1.4 Service procedure

If a service of the specified name and with the specified version number is available, a reference to it is created and submitted to the requester.

6.3.2   Initiate Peripheral Interface Type service

6.3.2.1   Service overview

This service requests the availability of an interface with the specified type name. If a name for an extended service provider is specified with this request, the extended provider will be loaded and the availability of the specified interface is requested at this provider.

6.3.2.2   Service parameter structure

The service parameters for this service are shown in Table 7.

Table 7 — Initiate Peripheral Interface Type parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface type name	M	
Extended services provider name	C	
Result (+)		S
Interface type identifier		M
Result (-)		S
Error		M

6.3.2.3   Service parameters

6.3.2.3.1   Argument

The argument contains the parameters of the service request.

6.3.2.3.2   Interface type name

This parameter contains the name of the interface type.

6.3.2.3.3   Extended services provider name

This conditional parameter, if specified, contains the name of an extended services provider that handles the input/output services for this type of interface.

6.3.2.3.4   Result (+)

This selection type parameter indicates that the service request succeeded.

6.3.2.3.5   Interface type identifier

This parameter contains a number identifying this interface type for other service requests.

6.3.2.3.6 Result (-)

This selection type parameter indicates that the service request failed.

6.3.2.3.7 Error

This parameter indicates that one of the following conditions exists:

—   unknown or unavailable interface type;

—   unknown, unloadable, or unusable extended service provider;

—   interface type already initialized;

—   memory error occurred while loading the extended service provider;

—   hardware error detected.

6.3.2.4 Service procedure

This service checks the availability of an interface with the specified type. If a name for an extended service provider is given, the applicable provider is loaded, if not present, and checked for the specified type of interface.

6.3.3 Conclude Peripheral Interface Type service

6.3.3.1 Service overview

This service concludes a peripheral interface which has been initiated before.

6.3.3.2 Service parameter structure

The service parameters for this service are shown in Table 8.

Table 8 — Conclude Peripheral Interface Type parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface type identifier	M	
Result (+)		S
Result (-)		S
Error		M

6.3.3.3 Service parameters

6.3.3.3.1 Argument

The argument contains the parameters of the service request.

6.3.3.3.2   Interface type identifier

This parameter contains the identifier of the type which was the result of the service Initiate Periphery Interface Type.

6.3.3.3.3   Result (+)

This selection type parameter indicates that the service request succeeded.

6.3.3.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.3.3.3.5   Error

This parameter indicates one of the following conditions exists:

⎯ unknown or unavailable interface type identifier;

⎯ extended services provider could not be released;

⎯ memory error occurred while releasing the extended service provider;

⎯ hardware error was detected.

6.3.3.4   Service procedure

This service checks if this type of peripheral interface can be released. If this type is served by an associated extended service provider, the service also checks to see if the associated extended service provider can be released. An extended service provider can only be released if all contained peripheral interfaces of all types are released.

6.4   Input/output services

6.4.1   Open Peripheral Interface Channel service

6.4.1.1   Service overview

This service is used to open a communication channel for a peripheral interface of specified type and configures the channel for the user's needs.

6.4.1.2   Service parameter structure

The service parameters for this service are shown in Table 9.

Table 9 — Open Peripheral Interface Channel parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface type identifier	M	
Interface channel name	C	
List of configuration parameters	M	
Confirmed services access point reference	M	
Event services access point reference	M	
Result (+)		S
Interface channel identifier		M
Result (-)		S
Error		M

6.4.1.3   Service parameters

6.4.1.3.1   Argument

The argument contains the parameters of the service request.

6.4.1.3.2   Interface type identifier

This parameter identifies the type of the peripheral interface.

6.4.1.3.3   Interface channel name

This parameter identifies a communication channel of the specified type. It may be omitted, if channels are instantiated by RMS and there is no assignment of unique peripheral connectors necessary (e.g. bus systems).

6.4.1.3.4   List of configuration parameters

The list of configuration parameters, which is outside the scope of ISO 20242, depends on the type of the selected peripheral interface. A selected peripheral interface specifies the necessary parameters for its configuration.

If there are configuration parameters valid for all channels of the specified type, only the first use of this service after initiating the interface type will set these parameters.

6.4.1.3.5   Confirmed services access point reference

This parameter is the reference of a special service access point which is used for all confirmed services belonging to this channel.

6.4.1.3.6   Event services access point reference

This parameter is the reference of a special service access point which shall be used for all event handling services belonging to this channel.

6.4.1.3.7   Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.1.3.8   Interface channel identifier

This parameter is the identifier for this channel and will be used for channel specific service requests.

6.4.1.3.9   Result (-)

This selection type parameter indicates that the service request failed.

6.4.1.3.10   Error

This parameter indicates one of the following conditions exists:

— interface of that type unavailable;

— channel of that name unavailable;

— channel of that name is already opened;

— missing or invalid channel name;

— missing or invalid reference for confirmed services access point;

— missing or invalid reference for event services access point;

— memory error occurred;

— hardware error detected;

— channel configuration timeout;

— insufficient resources to open this channel;

— unspecified parameter error;

— specified parameter error.

6.4.1.4   Service procedure

This service checks all parameters and, if there is no error, it tries to open the specified communication channel. If the channel is ready for use, a channel identifier is returned; otherwise an error is returned.

6.4.2   Reconfigure Peripheral Interface Channel service

6.4.2.1   Service overview

This service is used to reconfigure an open communication channel for a peripheral interface.

6.4.2.2   Service parameter structure

The service parameters for this service are shown in Table 10.

Table 10 — Reconfigure Peripheral Interface Channel parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
List of configuration parameters	M	
Confirmed services access point reference	M	
Event services access point reference	M	
Result (+)		S
Interface channel identifier		M
Result (-)		S
Error		M

6.4.2.3   Service parameters

6.4.2.3.1   Argument

The argument contains the parameters of the service request.

6.4.2.3.2   Interface channel identifier

This parameter identifies the channel of the peripheral interface.

6.4.2.3.3   List of configuration parameters

The list of configuration parameters, which is outside the scope of ISO 20242, depends on the type of the selected peripheral interface. A selected peripheral interface specifies the necessary parameters for its configuration.

If there are configuration parameters valid for all channels of this type of interface, these parameters will only be set if the interface channel identifier is that of the first opened channel after initiating the interface type.

6.4.2.3.4   Confirmed services access point reference

This parameter is the reference of a special service access point which shall be used for all confirmed services belonging to this channel.

6.4.2.3.5   Event services access point reference

This parameter is the reference of a special service access point which shall be used for all event handling services belonging to this channel.

6.4.2.3.6   Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.2.3.7 Interface channel identifier

This parameter is the identifier for this channel and will be used for channel specific service requests.

6.4.2.3.8 Result (-)

This selection type parameter indicates that the service request failed.

6.4.2.3.9 Error

This parameter indicates that one of the following conditions exists:

— channel unavailable;

— channel busy;

— missing or invalid reference for confirmed services access point;

— missing or invalid reference for event services access point;

— memory error occurred;

— hardware error detected;

— channel configuration timeout;

— insufficient resources to configure the channel;

— unspecified parameter error;

— specified parameter error.

6.4.2.4 Service procedure

This service checks if there is any transmission activity with this channel and produces a Result (-) with a "channel busy" error in that case. Otherwise, all parameters are checked and, if there is no error, the specified communication channel will be newly configured. If the channel is ready for use, a Result (+) is delivered; otherwise, a Result (-) with an error is delivered.

6.4.3 Read Data service

6.4.3.1 Service overview

This service is used to receive data via a peripheral interface channel.

6.4.3.2 Service parameter structure

The service parameters for this service are shown in Table 11.

Table 11 — Read Data parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
Data receiving process handle	M	
Maximum length of received data	M	
Maximum process duration time	M	
Result (+)		S
Data receiving process handle		M
Received data length		M
Received data		M
Result (-)		S
Data receiving process handle		M
Received data length		M
Received data		M
Error		M

6.4.3.3   Service parameters

6.4.3.3.1   Argument

The argument contains the parameters of the service request.

6.4.3.3.2   Interface channel identifier

This parameter identifies the channel of the peripheral interface expecting receive data.

6.4.3.3.3   Data receiving process handle

This parameter is a user defined identifier for the data receiving process.

6.4.3.3.4   Maximum length of received data

The maximum length of received data is the maximum number of octets contained in a received protocol data unit.

6.4.3.3.5   Maximum process duration time

This parameter contains the maximum duration time of data receiving in milliseconds.

6.4.3.3.6   Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.3.3.7   Data receiving process handle

This parameter is a copy of the user defined process handle delivered with the service request.

6.4.3.3.8   Received data length

The number of octets contained in a received protocol data unit.

6.4.3.3.9   Received data

This parameter returns the received protocol data unit consisting of one or more octets. The number of octets is always less than or equal to the maximum data length specified with the request.

6.4.3.3.10   Result (-)

This selection type parameter indicates that the service request failed.

6.4.3.3.11   Data receiving process handle

This parameter is a copy of the user defined process handle delivered with the service request.

6.4.3.3.12   Received data length

The number of octets received until the error occurred and receiving process stopped.

6.4.3.3.13   Received data

This parameter returns the received data consisting of zero or more octets.

6.4.3.3.14   Error

This parameter indicates that one of the following conditions exists:

— channel unavailable;

— receiving process of this channel is busy;

— memory error occurred;

— hardware error detected;

— allocation of communication buffer failed;

— invalid specified maximum data length;

— data receiving timeout;

— receiving process cancelled by user.

6.4.3.4   Service procedure

If a channel is available and no receiving process is active for it, this service starts a data receiving process that finishes either when a complete protocol data unit is received, when the maximum data length is reached, when a timeout occurs, or when an error occurs. In the case of a timeout or an error, the receiver buffer contains the octets received at that time.

6.4.4 Write Data service

6.4.4.1 Service overview

This service is used to transmit data via a peripheral interface channel.

6.4.4.2 Service parameter structure

The service parameters for this service are shown in Table 12.

Table 12 — Write Data parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
Data transmission process handle	M	
Length of data being transmitted	M	
Data to be transmitted	M	
Maximum process duration time	M	
Result (+)		S
Data transmission process handle		M
Result (-)		S
Data transmission process handle		M
Transmitted data length		M
Error		M

6.4.4.3 Service parameters

6.4.4.3.1 Argument

The argument contains the parameters of the service request.

6.4.4.3.2 Interface channel identifier

This parameter identifies the channel of the peripheral interface to transmit data.

6.4.4.3.3 Data transmission process handle

This parameter is a user defined identifier for this data transmission process.

6.4.4.3.4 Length of data being transmitted

This parameter specifies the number of octets which have to be transmitted.

6.4.4.3.5   Data to be transmitted

This parameter contains one or more octets which have to be transmitted.

6.4.4.3.6   Maximum process duration time

This parameter contains the maximum duration time of data transmission in milliseconds.

6.4.4.3.7   Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.4.3.8   Data transmission process handle

This parameter is a copy of the user defined process handle delivered with the service request.

6.4.4.3.9   Result (-)

This selection type parameter indicates that the service request failed.

6.4.4.3.10   Data transmission process handle

This parameter is a copy of the user defined process handle delivered with the service request.

6.4.4.3.11   Transmitted data length

The number of octets transmitted until the error occurred and transmission process stopped.

6.4.4.3.12   Error

This parameter indicates one of the following conditions exists:

— channel unavailable;

— transmission process of this channel is busy;

— memory error occurred;

— hardware error detected;

— allocation of communication buffer failed;

— invalid specified maximum data length;

— data transmission timeout;

— transmission process cancelled by user.

6.4.4.4   Service procedure

If the channel is available and no transmission process is active for it, this service starts a data transmission process that finishes either when the specified number of octets is transmitted, when a timeout occurs, or when an error occurs. In the case of a timeout or error, the parameter "transmitted data length" returns the number of transmitted octets at the time the timeout or error occurred.

6.4.5   Execute Operation service

6.4.5.1   Service overview

This service is used to execute an operation of this peripheral interface channel.

6.4.5.2   Service parameter structure

The service parameters for this service are shown in Table 13.

Table 13 — Execute Operation parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
Execute operation process handle	M	
Operation identifier	M	
List of operation input parameters	M	
Maximum process duration time	M	
Result (+)		S
Execute operation process handle		M
List of operation output parameters		M
Result (-)		S
Execute operation process handle		M
Error		M

6.4.5.3   Service parameters

6.4.5.3.1   Argument

The argument contains the parameters of the service request.

6.4.5.3.2   Interface channel identifier

This parameter identifies the channel of the peripheral interface comprising the operation.

6.4.5.3.3   Execute operation process handle

This parameter is a user defined identifier for this execution process.

6.4.5.3.4   Operation identifier

This parameter is the identifier for the operation to be executed.

6.4.5.3.5 List of operation input parameters

This parameter list depends on the kind of operation to be executed. Definition of the operations and their arguments are outside the scope of ISO 20242.

6.4.5.3.6 Maximum process duration time

This parameter contains the maximum duration time of executing the operation in milliseconds.

6.4.5.3.7 Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.5.3.8 Execute operation process handle

This parameter is a copy of the user defined process handle delivered with the service request.

6.4.5.3.9 List of operation output parameters

This parameter list depends on the kind of operation to be executed. Defining operations and their arguments is not within the scope of this part of ISO 20242.

6.4.5.3.10 Result (-)

This selection type parameter indicates that the service request failed.

6.4.5.3.11 Execute operation process handle

This parameter is a copy of the user defined process handle delivered with the service request.

6.4.5.3.12 Error

This parameter indicates that one of the following conditions exists:

— channel unavailable;

— operation not found, identifier not valid;

— execution of operation still in progress;

— memory error occurred;

— hardware error detected;

— input parameter list not valid;

— executing operation timeout;

— execution cancelled by user.

6.4.5.4 Service procedure

If the channel is available and the specified operation is not in progress, the operation will be started. If an error occurs or if it is cancelled by the RMS user, the operation is stopped.

6.4.6   Cancel Communication Process service

6.4.6.1   Service overview

This service is used to cancel a pending receiving or transmission process or the execution of an operation.

6.4.6.2   Service parameter structure

The service parameters for this service are shown in Table 14.

Table 14 — Cancel Communication Process parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
Communication process handle	M	
Result (+)		S
Result (-)		S
Error		M

6.4.6.3   Service parameters

6.4.6.3.1   Argument

The argument contains the parameters of the service request.

6.4.6.3.2   Interface channel identifier

This parameter identifies the communicating channel of the peripheral interface.

6.4.6.3.3   Communication process handle

This parameter is the identifier for the communication process to be cancelled. This identifier was defined by the user when starting the service which is now to be cancelled.

6.4.6.3.4   Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.6.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.4.6.3.6   Error

This parameter indicates that one of the following conditions exists:

— channel not found;

— communication process handle not open;

— cancelling communication not possible;

— memory error occurred;

— hardware error detected.

6.4.6.4 Service procedure

This service is used to cancel a pending communication process. If the service is successful, a new communication process can be started. A cancelled communication process produces a Result (-) and a "cancelled by user" error.

6.4.7 Get Periphery Interface Channel Status service

6.4.7.1 Service overview

This service is used to investigate the status of a receiving or transmission process or of the execution of an operation.

6.4.7.2 Service parameter structure

The service parameters for this service are shown in Table 15.

Table 15 — Get Periphery Interface Channel Status parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
Communication process handle	M	
Result (+)		S
Status		M
Progress		M
Result (-)		S
Error		M

6.4.7.3 Service parameters

6.4.7.3.1 Argument

The argument contains the parameters of the service request.

6.4.7.3.2 Interface channel identifier

This parameter identifies the communicating channel of the peripheral interface.

6.4.7.3.3   Communication process handle

This parameter is the identifier for the communication process to be checked. The identifier is defined by the user when starting the applicable service.

6.4.7.3.4   Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.7.3.5   Status

This parameter identifies the status of the peripheral interface channel.

6.4.7.3.6   Progress

This parameter describes the progress of the communication process. For data read or write processes it is the number of actual received or transmitted characters.

6.4.7.3.7   Result (-)

This selection type parameter indicates that the service request failed.

6.4.7.3.8   Error

This parameter indicates that one of the following conditions exists:

— channel not found;

— communication process handle not open;

— memory error occurred;

— hardware error detected.

6.4.7.4   Service procedure

This service checks if the specified communication process is pending and then ascertains its status.

6.4.8   Clear Read Buffer service

6.4.8.1   Service overview

This service is used to clear the read buffer of a peripheral interface channel.

6.4.8.2   Service parameter structure

The service parameters for this service are shown in Table 16.

Table 16 — Clear Read Buffer parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
Result (+)		S
Result (-)		S
Error		M

6.4.8.3 Service parameters

6.4.8.3.1 Argument

The argument contains the parameters of the service request.

6.4.8.3.2 Interface channel identifier

This parameter identifies the communicating channel of the peripheral interface.

6.4.8.3.3 Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.8.3.4 Result (-)

This selection type parameter indicates that the service request failed.

6.4.8.3.5 Error

This parameter indicates that one of the following conditions exists:

— channel not found;

— memory error occurred;

— hardware error detected;

— receiving process is busy.

6.4.8.4 Service procedure

This service checks whether there is a receiving process active and, if not, it clears the read buffer.

6.4.9 Close Peripheral Interface Channel service

6.4.9.1 Overview

This service is used to close a peripheral interface channel.

6.4.9.2   Service parameter structure

The service parameters for this service are shown in Table 17.

Table 17 — Close Peripheral Interface Channel parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface channel identifier	M	
Result (+)		S
Result (-)		S
Error		M

6.4.9.3   Service parameters

6.4.9.3.1   Argument

The argument contains the parameters of the service request.

6.4.9.3.2   Interface channel identifier

This parameter identifies the communicating channel of the peripheral interface.

6.4.9.3.3   Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.9.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.4.9.3.5   Error

This parameter indicates that one of the following conditions exists:

⎯ channel not found;

⎯ memory error occurred;

⎯ hardware error detected;

⎯ communication process is busy.

6.4.9.4   Service procedure

This service checks whether there is a communication process open and, if not, it closes the peripheral interface channel.

6.4.10 Signal Event service

6.4.10.1 Service Overview

This service is used to signal local events.

6.4.10.2 Service parameter structure

The service parameters for this service are shown in Table 18.

Table 18 — Signal Event parameter structure

Parameter name	Ind	Rsp
Argument	M	
Interface channel identifier	M	
Event identifier	M	
Event message	M	
Result (+)		S
Status		M
Result (-)		S
Error		M

6.4.10.3 Service parameters

6.4.10.3.1 Argument

The argument contains the parameters of the service request.

6.4.10.3.2 Interface channel identifier

This parameter identifies the communicating channel of the peripheral interface. If the value is zero, the event is not specific for a single channel.

6.4.10.3.3 Event identifier

This parameter is the identifier for the event.

6.4.10.3.4 Event message

This parameter returns an event message, which is an event specific data structure.

NOTE   Defining event identifiers or the data events carry is not within the scope of this part of ISO 20242. This will be application specific.

6.4.10.3.5 Result (+)

This selection type parameter indicates that the service request succeeded.

6.4.10.3.6 Status

This parameter identifies the status of the event handler among the following choices:

— Event handling finished

— Event handling in progress

6.4.10.3.7 Result (-)

This selection type parameter indicates that the service request failed.

6.4.10.3.8 Error

This parameter shall indicate an error among the following choices:

— Event handling temporarily not possible

— Event with that identifier still in progress

— Handling of the specified event not possible

6.4.10.4 Service procedure

This service first checks if it is possible to handle this event. Then a handler is started accordingly and status "Event handling in progress" is returned or the handler is called and finishes its task and status "Event handling finished" is returned. If the event cannot be handled, an error is returned.

6.5 Extended services

6.5.1 Initiate Extended Interface Type

6.5.1.1 Service overview

The service request is forwarded by service Initiate Peripheral Interface Type (see 6.3.2) after installing an extended service provider.

6.5.1.2 Service parameter structure

The service parameters for this service are shown in Table 19.

Table 19 — Initiate Extended Interface Type parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface type name	M	
Interface type identifier	M	
Result (+)		S
Result (-)		S
Error		M

6.5.1.3   Service parameters

6.5.1.3.1   Argument

The argument contains the parameters of the service request.

6.5.1.3.2   Interface type name

This parameter contains the name of the interface type.

6.5.1.3.3   Interface type identifier

This parameter contains a number identifying this interface type for other service requests.

6.5.1.3.4   Result (+)

This selection type parameter indicates that the service request succeeded.

6.5.1.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.5.1.3.6   Error

This parameter shall indicate an error among the following choices:

— Unknown or not available interface type

— This interface type is already initialized

— A hardware error was detected

6.5.1.4   Service procedure

This service checks the availability of an interface with the specified type name. If this interface exists, it is prepared for use.

6.5.2   Conclude Extended Interface Type

Procedure and parameters of this service are identical to service Conclude Peripheral Interface Type as described in 6.3.3.

6.5.3   Open Extended Interface Channel

6.5.3.1   Service overview

This service is used to open a communication channel for an extended interface of specified type and configures the channel for the user's needs.

6.5.3.2   Service parameter structure

The service parameters for this service are shown in Table 20.

Table 20 — Open Extended Interface Channel parameter structure

Parameter name	Req	Cnf
Argument	M	
Interface type identifier	M	
Interface channel identifier	M	
Interface channel name	M	
List of configuration parameters	M	
Confirmed services access point reference	M	
Event services access point reference	M	
Result (+)		S
Result (-)		S
Error		M

6.5.3.3   Service parameters

6.5.3.3.1   Argument

The argument contains the parameters of the service request.

6.5.3.3.2   Interface type identifier

This parameter identifies the type of the peripheral interface.

6.5.3.3.3   Interface channel identifier

This parameter is an assigned identifier for the channel of the peripheral interface.

6.5.3.3.4   Interface channel name

This parameter identifies a communication channel of the specified type.

6.5.3.3.5   List of configuration parameters

The list of configuration parameters depends on the type of the selected peripheral interface. The specification of this list is not within the scope of this part of ISO 20242. The necessary parameters may be found at descriptions for the specified peripheral interfaces.

6.5.3.3.6   Confirmed services access point reference

This parameter is the reference of a special service access point which shall be used for all confirmed services belonging to this channel.

6.5.3.3.7   Event services access point reference

This parameter is the reference of a special service access point which shall be used for all event handling services belonging to this channel.

6.5.3.3.8   Result (+)

This selection type parameter indicates that the service request succeeded.

6.5.3.3.9   Result (-)

This selection type parameter indicates that the service request failed.

6.5.3.3.10   Error

This parameter shall indicate an error among the following choices:

⎯ Interface of that type is not available

⎯ Channel of that name is not available

⎯ Channel is already opened

⎯ Missing or invalid channel name

⎯ Missing or invalid reference for confirmed services access point

⎯ Missing or invalid reference for event services access point

⎯ A memory error occurred

⎯ A hardware error was detected

⎯ Timeout for channel configuration

⎯ Not enough resources to open this channel

⎯ Unspecified parameter error

⎯ Specified parameter error

⎯ Any other error

6.5.3.4   Service procedure

This service checks all parameters and, if there is no error, it tries to open the specified communication channel. If the channel is not ready for use, an error is returned.

6.5.4   Reconfigure Extended Interface Channel

Procedure and parameters of this service are identical to service Reconfigure Peripheral Interface Channel as described in 6.4.2.

6.5.5   Read Extended Interface Data

Procedure and parameters of this service are identical to service Read Data as described in 6.4.3.

6.5.6   Write Extended Interface Data

Procedure and parameters of this service are identical to service Write Data as described in 6.4.4.

6.5.7 Execute Extended Interface Operation

Procedure and parameters of this service are identical to service Execute Operation as described in 6.4.5.

6.5.8 Cancel Extended Communication Process

Procedure and parameters of this service are identical to service Cancel Communication Process as described in 6.4.6.

6.5.9 Get Extended Interface Channel Status

Procedure and parameters of this service are identical to service Get Periphery Interface Channel Status as described in 6.4.7.

6.5.10 Clear Extended Interface Read Buffer

Procedure and parameters of this service are identical to service Clear Read Buffer as described in 6.4.8.

6.5.11 Close Extended Interface Channel

Procedure and parameters of this service are identical to service Close Peripheral Interface Buffer as described in 6.4.9.

6.5.12 Signal Extended Event

Procedure and parameters of this service are identical to service Signal Event as described in 6.4.10.

6.6 Operating Support Services

6.6.1 Allocate Memory

6.6.1.1 Service overview

This service is used to get a reference for coherent data space of specified size.

6.6.1.2 Service parameter structure

The service parameters for this service are shown in Table 21.

Table 21 — Allocate Memory parameter structure

Parameter name	Req	Cnf
Argument	M	
Number of octets in data space	M	
Result (+)		S
Allocated data space reference		M
Result (-)		S

6.6.1.3   Service parameters

6.6.1.3.1   Argument

The argument contains the parameters of the service request.

6.6.1.3.2   Number of octets in data space

This parameter specifies the size of the requested data space in octets (unit of eight bits).

6.6.1.3.3   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.1.3.4   Allocated data space reference

This parameter contains a reference to identify the requested data space.

6.6.1.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.6.1.4   Service procedure

If a coherent data area of the specified size is available, a reference to it is created. Otherwise the request fails.

6.6.2   Reallocate Memory

6.6.2.1   Service overview

This service is used to change the size of a previously allocated data space.

6.6.2.2   Service parameter structure

The service parameters for this service are shown in Table 22.

Table 22 — Reallocate Memory parameter structure

Parameter name	Req	Cnf
Argument	M	
Reference to allocated data space	M	
Number of octets in reallocated data space	M	
Result (+)		S
Reallocated data space reference		M
Result (-)		S

6.6.2.3   Service parameters

6.6.2.3.1   Argument

The argument contains the parameters of the service request.

6.6.2.3.2   Reference to allocated data space

This parameter is the reference to the data space whose size has to be changed. It shall be the result of either an Allocate Memory or Reallocate Memory service.

6.6.2.3.3   Number of octets in reallocated data space

This parameter specifies the new size of the requested data space in octets (unit of eight bits).

6.6.2.3.4   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.2.3.5   Service reference

This parameter contains a reference to identify the requested data space.

6.6.2.3.6   Result (-)

This selection type parameter indicates that the service request failed.

6.6.2.4   Service procedure

If data space is available to change the specified data space, a reference to the new data space is created. The contents of the data space shall be unchanged up to the shorter of the old and new sizes. If it is not possible to change the size of the data space, the request fails and the old data space stays valid.

6.6.3   Free Memory

6.6.3.1   Service overview

This service is used to get a reference for coherent data space of specified size.

6.6.3.2   Service parameter structure

The service parameters for this service are shown in Table 23.

Table 23 — Free Memory parameter structure

Parameter name	Req	Cnf
Argument	M	
Reference to allocated data space	M	
Result (+)		S
Result (-)		S
Error		M

6.6.3.3   Service parameters

6.6.3.3.1   Argument

The argument contains the parameters of the service request.

6.6.3.3.2   Reference to allocated data space

This parameter is the reference to the data space which has to be released. It has to be the result of either an Allocate Memory or Reallocate Memory service.

6.6.3.3.3   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.3.3.4   Service reference

This parameter contains a reference to identify the requested data space.

6.6.3.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.6.3.3.6   Error

This parameter shall indicate an error among the following choices:

— Invalid reference to allocated data space

— Any other error

6.6.3.4   Service procedure

If the specified reference is valid for an allocated data space, the space is released and will be free for any other allocation. Otherwise the old data space stays valid and an error is returned.

6.6.4   Get Time

6.6.4.1   Service overview

This service is used to investigate the time in different formats with respect to Universal Coordinated Time (UTC).

6.6.4.2   Service parameter structure

The service parameters for this service are shown in Table 24.

Table 24 — Get Time parameter structure

Parameter name	Req	Cnf
Argument	M	
Required time format	M	
Result (+)		S
List of time describing elements		M
Result (-)		S
Error		M

6.6.4.3   Service parameters

6.6.4.3.1   Argument

The argument contains the parameters of the service request.

6.6.4.3.2   Required time format

This parameter specifies the required time format among the following choices:

— UNIX time: the number of seconds since January 1, 1970, 00:00 o'clock UTC, units for seconds and microseconds since the last second;

— Absolute time: the Universal Coordinated Time units for year, month, day of month, hour, minute, second, millisecond, microsecond and nanosecond with an additional element containing the difference to the local time in seconds.

6.6.4.3.3   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.4.3.4   List of time describing elements

This parameter contains the elements of a time structure according to the required time format.

6.6.4.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.6.4.3.6   Error

This parameter shall indicate an error among the following choices:

— Time could not be investigated

— Any other error

6.6.4.4   Service procedure

The Universal Coordinated Time is verified by means of RMS. If that is not possible, an error is returned.

6.6.5   Get Process Time

6.6.5.1   Service overview

This service tells the value of a microseconds counter in RMS. The start time of the counter is not determined.

6.6.5.2   Service parameter structure

The service parameters for this service are shown in Table 25.

Table 25 — Get Process Time parameter structure

Parameter name	Req	Cnf
Result (+)		S
Local Process Time		M
Result (-)		S
Error		M

6.6.5.3   Service parameters

6.6.5.3.1   Argument

This service has no parameters.

6.6.5.3.2   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.5.3.3   Local Process Time

This parameter contains the value of a microseconds counter in RMS.

6.6.5.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.6.5.3.5   Error

This parameter shall indicate an error among the following choices:

— Local Process Time could not be investigated

— Any other error

6.6.5.4   Service procedure

RMS shall contain a microseconds counter. This counter is started with the availability of RMS, but the start value of the counter is not determined. This service investigates the actual value of the counter.

6.6.6 Wait

6.6.6.1 Service overview

This service creates a delay for a specified amount of time.

6.6.6.2 Service parameter structure

The service parameters for this service are shown in Table 26.

Table 26 — Wait parameter structure

Parameter name	Req	Cnf
Argument	M	
Waiting time	M	
Result (+)		S
Result (-)		S
Error		M

6.6.6.3 Service parameters

6.6.6.3.1 Argument

The argument contains the parameters of the service request.

6.6.6.3.2 Waiting time

This parameter specifies the amount of delay time in milliseconds.

6.6.6.3.3 Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.6.3.4 Result (-)

This selection type parameter indicates that the service request failed.

6.6.6.3.5 Error

This parameter shall indicate an error among the following choices:

— Temporizing failed, service finished immediately

— Any other error

6.6.6.4 Service procedure

If resources to determine time are available, this service waits a specified amount of time. Otherwise an error is returned.

6.6.7   Create Timer

6.6.7.1   Service overview

This service creates and starts a timer in RMS and installs a service access point to handle the timer event.

6.6.7.2   Service parameter structure

The service parameters for this service are shown in Table 27.

Table 27 — Create Timer parameter structure

Parameter name	Req	Cnf
Argument	M	
Timer access point reference	M	
Timer process handle	M	
Duration time	M	
Number of events	M	
Result (+)		S
Timer identifier		M
Result (-)		S
Error		M

6.6.7.3   Service parameters

6.6.7.3.1   Argument

The argument contains the parameters of the service request.

6.6.7.3.2   Timer access point reference

This parameter is the reference of a special service access point which shall be used to handle events belonging to this timer.

6.6.7.3.3   Timer process handle

This parameter is a handle defined by the user to identify events of this timer. The event carries this handle.

6.6.7.3.4   Duration time

This parameter is the time elapsing until a timer event is signalled.

6.6.7.3.5   Number of events

Timers may work repetitively. This parameter assigns the number of repetitions. The value zero denotes a repetitive timer which only stops if the timer is removed.

6.6.7.3.6   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.7.3.7   Timer identifier

This parameter is the identification for the created timer. The identifier is needed to remove the timer.

6.6.7.3.8   Result (-)

This selection type parameter indicates that the service request failed.

6.6.7.3.9   Error

This parameter shall indicate an error among the following choices:

— Resources for timers exhausted

— Any other error

6.6.7.4   Service procedure

If corresponding resources are available, a timer is created and started. After the given duration time is exhausted, a timer event is signalled. If the specified number of events is greater than one, the timer is started again. If the number of signalled events matches the specified number of events, the timer is removed and timer resources are available for another timer. If the specified number of events is zero, the timer runs repetitively and may only be stopped by Remove Timer service.

The accuracy of the timer depends on the resources of RMS and should be the best possible.

6.6.8   Signal Timer Event

6.6.8.1   Service overview

This service signals the end of a timing period and issues an indication primitive.

6.6.8.2   Service parameter structure

The service parameters for this service are shown in Table 28.

Table 28 — Signal Timer Event parameter structure

Parameter name	Ind	Rsp
Argument	M	
Timer process handle	M	
Timer status	M	
Result (+)		S
Result (-)		S
Error code		C

6.6.8.3   Service parameters

6.6.8.3.1   Argument

The argument contains the parameters of the service request.

6.6.8.3.2   Timer process handle

This parameter is a handle defined by the user to identify the timer which signals this event. This handle is a parameter of service Create Timer.

6.6.8.3.3   Timer status

This parameter denotes if a former event of this timer is still pending. An event is pending until a response primitive is received.

6.6.8.3.4   Result (+)

This parameter indicates that the service indication succeeded and the event is no longer pending.

6.6.8.3.5   Result (-)

This parameter indicates that the service indication cannot be received or processed and the event is not expected.

6.6.8.3.6   Error code

This parameter denotes the reason for the error condition.

6.6.8.4   Service procedure

If the duration time of a timer is exhausted, an event is signalled to the user with an indication primitive. It is checked, if a former event is still pending and status of timer is defined accordingly.

6.6.9   Remove Timer

6.6.9.1   Service overview

This service stops a running timer in RMS, removes it and frees all linked resources.

6.6.9.2   Service parameter structure

The service parameters for this service are shown in Table 29.

Table 29 — Remove Timer parameter structure

Parameter name	Req	Cnf
Argument	M	
Timer identifier	M	
Result (+)		S
Result (-)		S
Error		M

6.6.9.3   Service parameters

6.6.9.3.1   Argument

The argument contains the parameters of the service request.

6.6.9.3.2   Timer identifier

This parameter is an identifier for the timer returned by service Create Timer.

6.6.9.3.3   Result (+)

This parameter indicates that the service request succeeded and the timer is removed.

6.6.9.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.6.9.3.5   Error

This parameter shall indicate an error among the following choices:

— Timer identifier not valid

— Operation failed, timer event busy

— Any other error

6.6.9.4   Service procedure

If no event of the addressed timer is pending, the timer is stopped and removed. Otherwise an error is returned.

6.6.10 Create Light Process Timer

6.6.10.1   Service overview

This service creates and starts a light process timer in RMS and installs a service access point to handle the timer event.

6.6.10.2   Service parameter structure

The service parameters for this service are shown in Table 30.

6.6.10.3.9 Error

This parameter shall indicate an error among the following choices:

— Resources for light processes exhausted

— Any other error

6.6.10.4 Service procedure

If corresponding resources are available, a new light process is created and an associated timer is started. After the given duration time is exhausted, a timer event is signalled. If the specified number of events is greater than one, the timer is started again. If the number of signalled events matches the specified number of events, the timer is removed and the light process is closed. If the specified number of events is zero, the timer runs repetitively and may only be stopped by a Remove Timer service.

The accuracy of the timer depends on the scheduling of the light process.

6.6.11 Signal Light Process Timer Event

6.6.11.1 Service overview

The timing period is exhausted and the light process associated with this timer issues an indication primitive.

6.6.11.2 Service parameter structure

The service parameters for this service are shown in Table 31.

Table 31 — Signal Light Process Timer Event parameter structure

Parameter name	Ind	Rsp
Argument	M	
Timer process handle	M	
Timer status	C	
Result (+)		M

6.6.11.3 Service parameters

6.6.11.3.1 Argument

The argument contains the parameters of the service request.

6.6.11.3.2 Timer process handle

This parameter is a handle defined by the user to identify the timer which signals this event. This handle is a parameter of Create Light Process Timer service.

6.6.11.3.3 Timer status

If the light process is able to control absolute timing, this parameter is present and denotes if processing of a former event of this timer exceeded specified timer duration.

6.6.11.3.4   Result (+)

This parameter indicates that the service indication succeeded and the event is no longer pending.

6.6.11.4   Service procedure

If the duration time of a timer is exhausted, an event is signalled to the user with an indication primitive. If possible, it is checked whether a former event handling exceeded the specified timer period.

6.6.12 Remove Light Process Timer

6.6.12.1   Service overview

This service stops the light process service parameter which is associated with this timer and frees all linked resources.

6.6.12.2   Service parameter structure

The service parameters for this service are shown in Table 32.

Table 32 — Remove Light Process Timer table

Parameter name	Req	Cnf
Argument	M	
Timer identifier	M	
Result (+)		S
Result (-)		S
Error		M

6.6.12.3   Service parameters

6.6.12.3.1   Argument

The argument contains the parameters of the service request.

6.6.12.3.2   Timer identifier

This parameter is an identifier for the timer returned by Create Timer service.

6.6.12.3.3   Result (+)

This parameter indicates that the service request succeeded and the timer is removed.

6.6.12.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.6.12.3.5 Error

This parameter shall indicate an error among the following choices:

— Timer identifier not valid

— Operation failed, timer event busy

— Any other error

6.6.12.4 Service procedure

If no event of the addressed timer is pending, the light process is stopped and linked resources are freed. Otherwise an error is returned.

6.6.13 Identify Light Process

6.6.13.1 Service overview

This service identifies the light process of the requesting entity.

6.6.13.2 Service parameter structure

The service parameters for this service are shown in Table 33.

Table 33 — Identify Light Process parameter structure

Parameter name	Req	Cnf
Result (+)		S
Light process identifier		M
Result (-)		S
Error		M

6.6.13.3 Service parameters

6.6.13.3.1 Argument

This service request has no parameters.

6.6.13.3.2 Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.13.3.3 Light process identifier

This parameter contains a unique identifier for the light process.

6.6.13.3.4 Result (-)

This selection type parameter indicates that the service request failed.

6.6.13.3.5   Error

This parameter shall indicate an error among the following choices:

— Light process could not be identified

— Any other error

6.6.13.4   Service procedure

This service investigates the identifier of the actual light process of the service requesting user. If this is not possible an error is returned.

6.6.14   Create Counted Semaphore

6.6.14.1   Service overview

This service creates a control handle for resources which may be used only a limited number at the same time. How often a resource may be used at the same time is specified with the service request.

6.6.14.2   Service parameter structure

The service parameters for this service are shown in Table 34.

Table 34 — Create Counted Semaphore parameter structure

Parameter name	Req	Cnf
Argument	M	
Number of concurrent usages	M	
Result (+)		S
Semaphore handle		M
Result (-)		S
Error		M

6.6.14.3   Service parameters

6.6.14.3.1   Argument

The argument contains the parameters of the service request.

6.6.14.3.2   Number of concurrent usages

This parameter specifies how many users are allowed to access the resource concurrently.

6.6.14.3.3   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.14.3.4 Semaphore handle

This is the handle for controlling the access to the protected resource.

6.6.14.3.5 Result (-)

This selection type parameter indicates that the service request failed.

6.6.14.3.6 Error

This parameter shall indicate an error among the following choices:

— Number of concurrent users not valid

— No more semaphore control available

— Any other error

6.6.14.4 Service procedure

It is checked if the number of concurrent users is valid and if resources for access control are available. Then a control object is created, an associated counter is loaded with the number of maximum concurrent users and a handle is returned. Otherwise an error is returned.

6.6.15 Wait for Counted Semaphore

6.6.15.1 Service overview

With this service the access to a resource is requested which is protected by the specified semaphore. If the access is not allowed, access is deferred until a specified time interval.

6.6.15.2 Service parameter structure

The service parameters for this service are shown in Table 35.

Table 35 — Wait for Counted Semaphore parameter structure

Parameter name	Req	Cnf
Argument	M	
Semaphore handle	M	
Maximum waiting time	M	
Result (+)		S
Result (-)		S
Error		M

6.6.15.3   Service parameters

6.6.15.3.1   Argument

The argument contains the parameters of the service request.

6.6.15.3.2   Semaphore handle

This parameter is the handle of the counted semaphore protecting the resource. The handle was returned by Create Counted Semaphore service.

6.6.15.3.3   Maximum waiting time

This is the maximum time for deferring access to the protected resource.

6.6.15.3.4   Result (+)

This selection type parameter indicates that the service request succeeded and the protected resource may be accessed.

6.6.15.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.6.15.3.6   Error

This parameter shall indicate an error among the following choices:

— Semaphore handle not valid

— Maximum waiting time exceeded

— Any other error

6.6.15.4   Service procedure

If the semaphore handle is valid and the associated counter is greater than zero, the counter is decremented by one and access to the protected resource is granted. Otherwise access is deferred until the counter is greater than zero or the specified waiting time is exceeded. If the waiting time is exceeded, an error is returned.

6.6.16   Release Counted Semaphore

6.6.16.1   Service overview

With this service the access to a resource protected by the specified semaphore is released.

6.6.16.2   Service parameter structure

The service parameters for this service are shown in Table 36.

Table 36 — Release Counted Semaphore parameter structure

Parameter name	Req	Cnf
Argument	M	
Semaphore handle	M	
Result (+)		S
Result (-)		S
Error		M

6.6.16.3   Service parameters

6.6.16.3.1   Argument

The argument contains the parameters of the service request.

6.6.16.3.2   Semaphore handle

This parameter is the handle of the counted semaphore protecting the resource. The handle was returned by a Create Counted Semaphore service.

6.6.16.3.3   Result (+)

This selection type parameter indicates that the service request succeeded and the release of the protected resource is notified.

6.6.16.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.6.16.3.5   Error

This parameter shall indicate an error among the following choices:

— Semaphore handle not valid

— No more release possible

— Any other error

6.6.16.4   Service procedure

If the semaphore handle is valid and the associated counter is less than the initial value, the counter is incremented by one. Otherwise an error is returned.

6.6.17   Delete Counted Semaphore

6.6.17.1   Service overview

This service deletes a counted semaphore and frees all resources which had been allocated for it.

6.6.17.2   Service parameter structure

The service parameters for this service are shown in Table 37.

Table 37 — Delete Counted Semaphore parameter structure

Parameter name	Req	Cnf
Argument	M	
Semaphore handle	M	
Result (+)		S
Result (-)		S
Error		M

6.6.17.3   Service parameters

6.6.17.3.1   Argument

The argument contains the parameters of the service request.

6.6.17.3.2   Semaphore handle

This parameter is the handle of the counted semaphore protecting the resource. The handle was returned by a Create Counted Semaphore service.

6.6.17.3.3   Result (+)

This selection type parameter indicates that the service request succeeded and the semaphore is deleted.

6.6.17.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.6.17.3.5   Error

This parameter shall indicate an error among the following choices:

—   Semaphore handle not valid

—   Protected resource still accessed

—   Any other error

6.6.17.4   Service procedure

If the semaphore handle is valid and the protected resource is no more accessed, the semaphore is deleted. Otherwise an error is returned.

6.6.18 Create Private Semaphore

6.6.18.1   Service overview

This service creates a control handle for resources which may be used in an unlimited manner by the owning light process. A light process owns the resource when it gets access to it. No other light process may access the resource until it is completely released by the owning light process.

6.6.18.2   Service parameter structure

The service parameters for this service are shown in Table 38.

Table 38 — Create Private Semaphore parameter structure

Parameter name	Req	Cnf
Result (+)		S
Semaphore handle		M
Result (-)		S
Error		M

6.6.18.3   Service parameters

6.6.18.3.1   Argument

This service request has no parameters.

6.6.18.3.2   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.18.3.3   Semaphore handle

This is the handle for controlling the access to the protected resource.

6.6.18.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.6.18.3.5   Error

This parameter shall indicate an error among the following choices:

— No more semaphore control available

— Any other error

6.6.18.4   Service procedure

If resources for access control are available, a control object is created and a handle is returned. Otherwise an error is returned.

6.6.19 Wait for Private Semaphore

6.6.19.1   Service overview

With this service the access to a resource is requested which is protected by the specified semaphore. The access is always allowed for the owning light process. If another light process requests access, access to the resource is deferred until a specified time is exceeded or the owning process releases the resource.

6.6.19.2   Service parameter structure

The service parameters for this service are shown in Table 39.

Table 39 — Wait for Private Semaphore parameter structure

Parameter name	Req	Cnf
Argument	M	
Semaphore handle	M	
Maximum waiting time	M	
Result (+)		S
Result (-)		S
Error		M

6.6.19.3   Service parameters

6.6.19.3.1   Argument

The argument contains the parameters of the service request.

6.6.19.3.2   Semaphore handle

This parameter is the handle of the private semaphore protecting the resource. The handle was returned by a Create Private Semaphore service.

6.6.19.3.3   Maximum waiting time

This is the maximum time for deferring access to the protected resource.

6.6.19.3.4   Result (+)

This selection type parameter indicates that the service request succeeded and the protected resource may be accessed.

6.6.19.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.6.19.3.6   Error

This parameter shall indicate an error among the following choices:

—   Semaphore handle not valid

—   Maximum waiting time exceeded

—   Any other error

6.6.19.4   Service procedure

If the semaphore handle is valid and the resource is owned by this light process or the resource is not owned by any light process, access to the protected resource is granted. If the resource is owned by another light process, access to the resource is deferred until the resource is completely released or the specified waiting time is exceeded. If the waiting time is exceeded, an error is returned.

If this is the first access of this light process, it becomes the owner of the resource.

6.6.20   Release Private Semaphore

6.6.20.1   Service overview

With this service the access to a resource protected by the specified semaphore is released. The ownership of the resource by a light process expires if the number of releases by this service is equal to the number of unreleased accesses by a Wait for Private Semaphore service.

6.6.20.2   Service parameter structure

The service parameters for this service are shown in Table 40.

Table 40 — Release Private Semaphore parameter structure

Parameter name	Req	Cnf
Argument	M	
Semaphore handle	M	
Result (+)		S
Result (-)		S
Error		M

6.6.20.3   Service parameters

6.6.20.3.1   Argument

The argument contains the parameters of the service request.

6.6.20.3.2   Semaphore handle

This parameter is the handle of the counted semaphore protecting the resource. The handle was returned by a Create Private Semaphore service.

6.6.20.3.3 Result (+)

This selection type parameter indicates that the service request succeeded and the release of the protected resource is notified.

6.6.20.3.4 Result (-)

This selection type parameter indicates that the service request failed.

6.6.20.3.5 Error

This parameter shall indicate an error among the following choices:

— Semaphore handle not valid

— No more release possible

— Any other error

6.6.20.4 Service procedure

If the semaphore handle is valid, the release of the semaphore is notified. Otherwise an error is returned.

The matching of services Waiting for Private Semaphore and Release Private Semaphore is controlled. The protected resource may be accessed by another light process if the numbers of both requests are equal.

6.6.21 Delete Private Semaphore

6.6.21.1 Service overview

This service deletes a private semaphore and frees all resources which had been allocated for it.

6.6.21.2 Service parameter structure

The service parameters for this service are shown in Table 41.

Table 41 — Semaphore parameter structure

Parameter name	Req	Cnf
Argument	M	
Semaphore handle	M	
Result (+)		S
Result (-)		S
Error		M

6.6.21.3 Service parameters

6.6.21.3.1 Argument

The argument contains the parameters of the service request.

6.6.21.3.2   Semaphore handle

This parameter is the handle of the private semaphore protecting the resource. The handle was returned by a Create Private Semaphore service.

6.6.21.3.3   Result (+)

This selection type parameter indicates that the service request succeeded and the semaphore is deleted.

6.6.21.3.4   Result (-)

This selection type parameter indicates that the service request failed.

6.6.21.3.5   Error

This parameter shall indicate an error among the following choices:

—   Semaphore handle not valid

—   Protected resource still owned

—   Any other error

6.6.21.4   Service procedure

If the semaphore handle is valid and the protected resource is not owned by a light process, the semaphore is deleted. Otherwise an error is returned.

6.6.22   Open Debug Log

6.6.22.1   Service overview

This service opens a logging channel for debug messages.

6.6.22.2   Service parameter structure

The service parameters for this service are shown in Table 42.

Table 42 — Open Debug Log parameter structure

Parameter name	Req	Cnf
Argument	M	
Debug channel identifier	M	
Result (+)		S
Log handle		M
Result (-)		S
Error		M

6.6.22.3   Service parameters

6.6.22.3.1   Argument

The argument contains the parameters of the service request.

6.6.22.3.2   Debug channel identifier

This parameter identifies a debug channel. It is defined by an RMS user and debug channel resources are assigned by the RMS provider.

6.6.22.3.3   Result (+)

This selection type parameter indicates that the service request succeeded.

6.6.22.3.4   Log handle

This is the handle for the further access to the debug channel.

6.6.22.3.5   Result (-)

This selection type parameter indicates that the service request failed.

6.6.22.3.6   Error

This parameter shall indicate an error among the following choices:

⎯ No more debug channel available

⎯ Identifier not valid

⎯ Any other error

6.6.22.4   Service procedure

If a debug channel is available and the identifier is valid or no identifier is given, a new debug channel is created for streaming text messages. Otherwise an error is returned.

6.6.23   Write Debug Message

6.6.23.1   Service overview

This service writes a message to a debug channel.

6.6.23.2   Service parameter structure

The service parameters for this service are shown in Table 43.

Table 43 — Write Debug Message parameter structure

Parameter name	Req	Cnf
Argument	M	
Log handle	M	
Debug message	M	
Result (+)		S
Result (-)		S
Error		M

6.6.23.3 Service parameters

6.6.23.3.1 Argument

The argument contains the parameters of the service request.

6.6.23.3.2 Log handle

This parameter is the handle for the debug channel. It was returned by an Open Debug Log service.

6.6.23.3.3 Result (+)

This selection type parameter indicates that the service request succeeded and the message is accepted.

6.6.23.3.4 Result (-)

This selection type parameter indicates that the service request failed.

6.6.23.3.5 Error

This parameter shall indicate an error among the following choices:

— Log handle not valid

— Debug channel temporarily not available

— Debug channel exhausted

— Any other error

6.6.23.4 Service procedure

If the log handle is valid and the debug channel is ready to accept a message, the message is written to the debug log. Otherwise an error is returned.

6.6.24 Close Debug Log

6.6.24.1 Service overview

This service closes a debug channel.

6.6.24.2 Service parameter structure

The service parameters for this service are shown in Table 44.

Table 44 — Close Debug Log parameter structure

Parameter name	Req	Cnf
Argument	M	
Log handle	M	
Result (+)		S
Result (-)		S
Error		M

6.6.24.3 Service parameters

6.6.24.3.1 Argument

The argument contains the parameters of the service request.

6.6.24.3.2 Log handle

This parameter is the handle for the debug channel. It was returned by an Open Debug Log service.

6.6.24.3.3 Result (+)

This selection type parameter indicates that the service request succeeded and the debug channel is closed.

6.6.24.3.4 Result (-)

This selection type parameter indicates that the service request failed.

6.6.24.3.5 Error

This parameter shall indicate an error among the following choices:

⎯ Log handle not valid

⎯ Debug channel busy

⎯ Any other error

6.6.24.4 Service procedure

If the log handle is valid and the debug channel is not busy, the channel is closed. Otherwise an error is returned.

6.7   States of RMS state machine

6.7.1   State transitions

Transitions from one status to the next are carried out by services, timeouts and finishing of tasks. Service errors always inhibit a state transition and are excluded in the state diagrams.

6.7.2   State diagram overview

The RMSI may hold similar services of different versions. Before a service is used, it has to be selected with the required version number. Users may only select those services that they need for their application, which may only be a part of all RMS (see Figure 4).
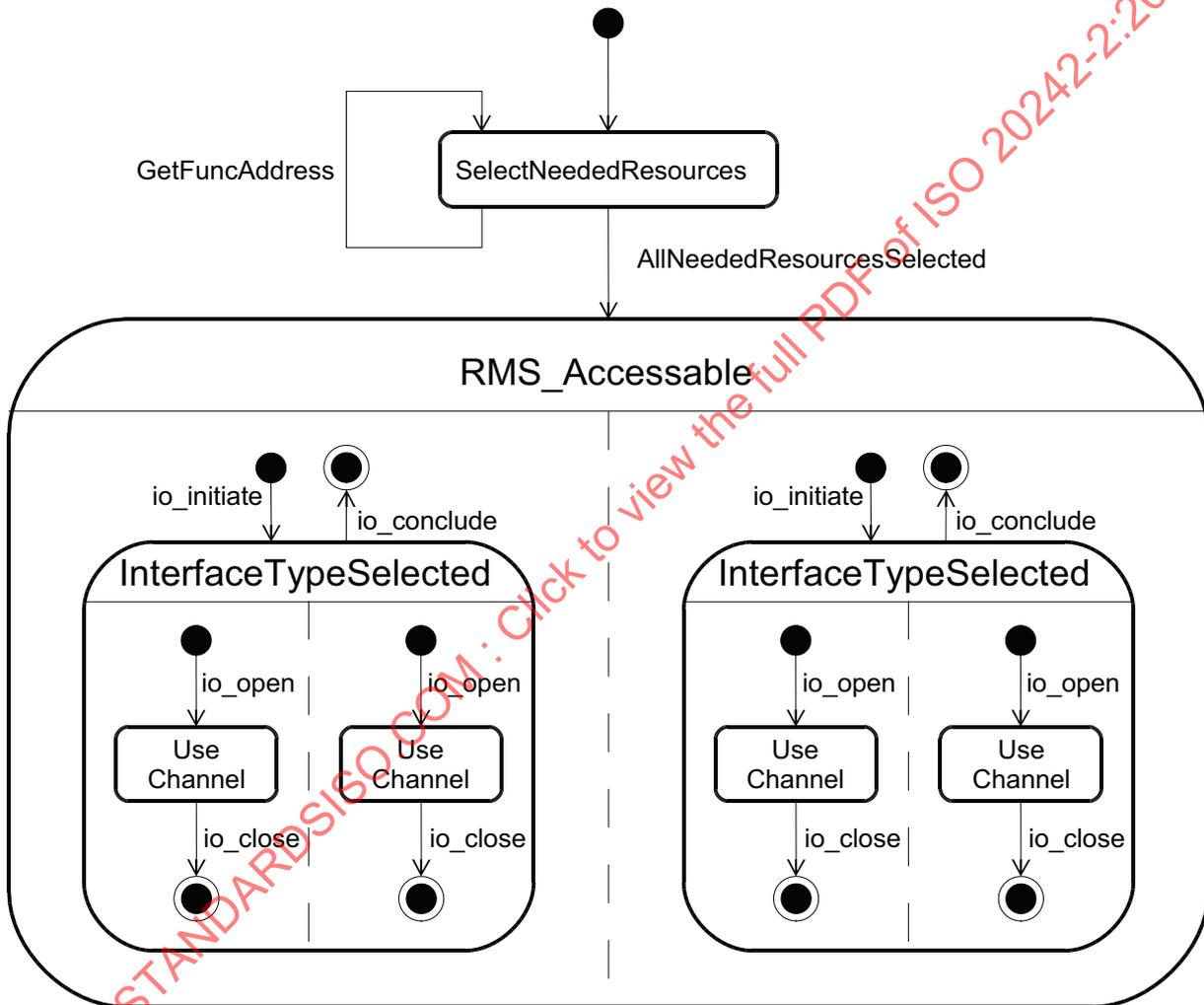

Figure 4 — RMS overview state diagram with multiple peripheral interfaces

The RMSI may support any number of different types of peripheral interfaces which may have any number of communication channels. An io_initiate service creates an object for an interface type and an io_open service creates an object for a communication channel. Therefore different interface type objects may be in different states as different channel objects of one interface type may be in different states. Figure 4 shows this with an example of two different regions out of any number for interface type objects and channel objects.

6.7.3 Channel state diagram

For the data exchange with peripheral interfaces or the execution of procedures dealing with peripheral components, processing time has to be considered and timeout may be specified or the process may be cancelled. The UML state diagram in Figure 5 shows details for these procedures.
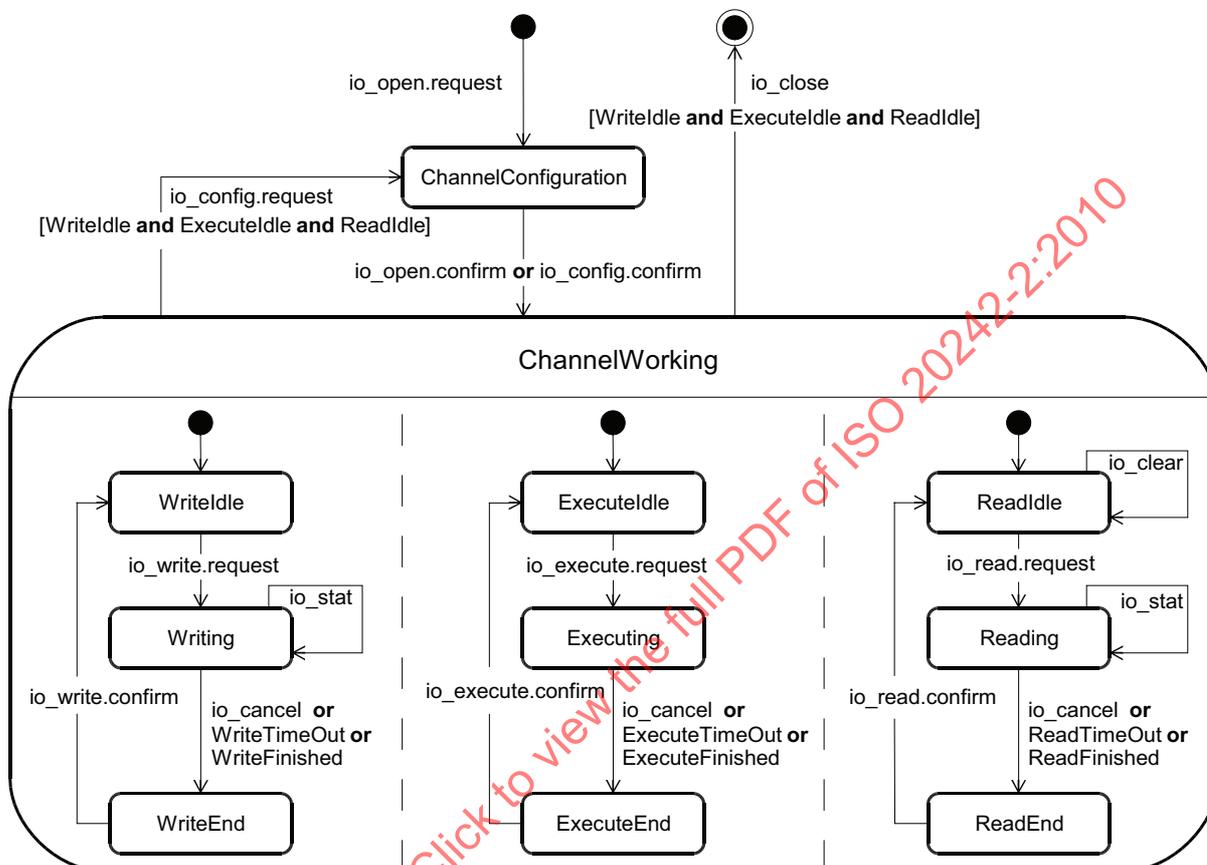

Figure 5 — RMS channel state diagram with input/output services

Input/output services and their matching extended services lead to special states of corresponding procedures in RMS. As input/output services are directly linked to extended services, state transitions are described with input/output services only.

6.7.4 RMSI states

6.7.4.1 SelectNeededResources state

This is the initial RMSI state, where only the GetFuncAddress service is available. This service is used for selecting the other needed services by service name and version number. After selection, the service is accessible by an RMS user.

6.7.4.2 RMS_Accessible state

This state is entered by the RMSI user if he or she has selected all needed resources. The state transition happens with the first access of a service other than GetFuncAddress.

6.7.4.3   InterfaceTypeSelected state

This state is entered with an io_initiate service. A specified type of peripheral interface or component is selected and may be used. The number of different types is not limited. Any number of TypeSelected states may exist concurrently for different types.

6.7.4.4   ChannelConfiguration state

This state belongs to a single channel of a selected interface type and is entered via an io_open.request service from the TypeSelected state or via an io_config.request service from the ChannelWorking state. If the configuration of the peripheral interface or component with an io_open service fails, the InterfaceTypeSelected state is assumed again. If a reconfiguration with an io_config service fails, the ChannelWorking state is assumed again with the old configuration.

6.7.4.5   ChannelWorking state

This state is entered if a configuration of a peripheral interface channel or component is successfully completed. Otherwise it is entered again if a reconfiguration fails and the old configuration stays valid.

6.7.4.6   WriteIdle state

A channel is ready to transmit data and waits for an io_write.request order. If the read and execute regions of this channel are in the ReadIdle or ExecuteIdle state, respectively, then the ChannelWorking state may be left by an io_close service.

6.7.4.7   Writing state

A channel is transmitting data. The transmission may be aborted with an io_cancel service. A transition from this state occurs if the transmission is finished or it takes longer than a specified time. With an io_stat service the actual number of transmitted bytes may be ascertained.

6.7.4.8   WriteEnd state

Writing is finished or aborted and a confirmation is created to enter the WriteIdle state again.

6.7.4.9   ReadIdle state

A channel is ready to receive data and waits for an io_read.request order. If the write and execute regions of this channel are in the WriteIdle state and ExecuteIdle state, respectively, then the ChannelWorking state may be left by an io_close service.

6.7.4.10   Reading state

A channel is receiving data and waits for peripheral input. The receiving activity may be aborted with an io_cancel service. The state is abandoned if the receiving activity is finished or it takes longer than a specified time. With an io_stat service the actual number of received bytes may be ascertained.

6.7.4.11   ReadEnd state

Reading is finished or aborted and a confirmation is created to enter the ReadIdle state again.

6.7.4.12   ExecuteIdle state

A channel is ready to execute a peripheral process and waits for an io_execute.request. If the read and write regions of this channel are in the ReadIdle state and WriteIdle state, the ChannelWorking state may be abandoned by an io_close service.