Infrared Data Association

Specifications for Ir Mobile Communications (IrMC)

Version 1.0.1

January 10, 1998

Ericsson

Hewlett-Packard

Matsushita/Panasonic

Motorola

NEC

Nokia Mobile Phones Ltd.

NTT DoCoMo

Puma

TU-KA Phone Kansai Inc.

Authors:

Kazuya Anzawa, anzawa@mdev.nttdocomo.co.jp (NTT DoCoMo),

Katsunori Hamada, hamada@mdev.nttdocomo.co.jp (NTT DoCoMo),

Heli Helanummi, heli.helanummi@nmp.nokia.com (Nokia Mobile Phones, Ltd.),

Petri Nykänen, petri.nykanen@nmp.nokia.com (IrDA Telecom SIG Chair, Nokia Mobile Phones, Ltd.),

James Scales, james.scales@nmp.nokia.com (Nokia Mobile Phones, Ltd.),

Robert K. Lockhart, rob.lockhart@mot.com (Motorola)

Contributors:

H. Chad Johnson (Ericsson),

Estella Martinez (Ericsson),

Kohei Akiyama (Hewlett-Packard Japan, Ltd.),

Ray Chock (Hewlett-Packard),

John Petrilla (Hewlett-Packard),

Tetsuya Kaku (IBM Japan,Ltd.),

Katsuya Matsunaga (IBM Japan,Ltd.),

Yuko Saeki (Matsushita/Panasonic),

John Byrns (Motorola),

Sheila Rader (Motorola),

Hiroshi Ono (NEC),

Petri Heinonen (Nokia Mobile Phones, Ltd.),

Kiyohito Nagata (NTT DoCoMo),

Masanari Arai (Puma),

Glen Walant (Puma),

Tadami Tanabe (SHARP),

Hidetaka Suzuki (TU-KA Phone Kansai Inc.),

Toshihiro Nishihata (TU-KA Phone Kansai Inc.),

Takashi Fujimoto (TU-KA phone Kansai Inc.)

Editor:

Stéphane Bouet, bouet@japan.nmp.nokia.com (Nokia Mobile Phones, Ltd.)

Document Status: **Version 1.0 - First Release, October 15, 1997**

Version 1.0.1 - Title Correction, January 10, 1998

INFRARED DATA ASSOCIATION (IrDA) - NOTICE TO THE TRADE -

SUMMARY:

Following is the notice of conditions and understandings upon which this document is made available to members and non-members of the Infrared Data Association.

- Availability of Publications, Updates and Notices
- Full Copyright Claims Must be Honored
- Controlled Distribution Privileges for IrDA Members Only
- Trademarks of IrDA - Prohibitions and Authorized Use
- No Representation of Third Party Rights
- Limitation of Liability
- Disclaimer of Warranty
- Certification of Products Requires Specific Authorization from IrDA after Product Testing for IrDA Specification Conformance

IrDA PUBLICATIONS and UPDATES:

IrDA publications, including notifications, updates, and revisions, are accessed electronically by IrDA members in good standing during the course of each year as a benefit of annual IrDA membership. Electronic copies are available to the public on the IrDA web site located at irda.org. IrDA publications are available to non-IrDA members for a pre-paid fee. Requests for publications, membership applications or more information should be addressed to: Infrared Data Association, P.O. Box 3883, Walnut Creek, California, U.S.A. 94598; or e-mail address: info@irda.org; or by calling John LaRoche at (510) 943-6546 or faxing requests to (510) 934-5600.

COPYRIGHT:

1. Prohibitions: IrDA claims copyright in all IrDA publications. Any unauthorized reproduction, distribution, display or modification, in whole or in part, is strictly prohibited.

2. Authorized Use: Any authorized use of IrDA publications (in whole or in part) is under NONEXCLUSIVE USE LICENSE ONLY. No rights to sublicense, assign or transfer the license are granted and any attempt to do so is void.

DISTRIBUTION PRIVILEGES for IrDA MEMBERS ONLY:

IrDA Members Limited Reproduction and Distribution Privilege: A limited privilege of reproduction and distribution of IrDA copyrighted publications is granted to IrDA members in good standing and for sole purpose of reasonable reproduction and distribution to non-IrDA members who are engaged by contract with an IrDA member for the development of IrDA certified products. Reproduction and distribution by the non-IrDA member is strictly prohibited.

TRANSACTION NOTICE to IrDA MEMBERS ONLY:

Each and every copy made for distribution under the limited reproduction and distribution privilege shall be conspicuously marked with the name of the IrDA member and the name of the receiving party. Upon reproduction for distribution, the distributing IrDA member shall promptly notify IrDA (in writing or by e-mail) of the identity of the receiving party.

A failure to comply with the notification requirement to IrDA shall render the reproduction and distribution unauthorized and IrDA may take appropriate action to enforce its copyright, including but not limited to, the termination of the limited reproduction and distribution privilege and IrDA membership of the non-complying member.

TRADEMARKS:

1. Prohibitions: IrDA claims exclusive rights in its trade names, trademarks, service marks, collective membership marks and certification marks (hereinafter collectively "trademarks"), including but not limited to the following trademarks: INFRARED DATA ASSOCIATION (wordmark alone and with IR logo), IrDA (acronym mark alone and with IR logo), IR logo, IR DATA CERTIFIED (composite mark), and MEMBER IrDA (wordmark alone and with IR logo). Any unauthorized use of IrDA trademarks is strictly prohibited.

2. 2. Authorized Use: Any authorized use of a IrDA collective membership mark or certification mark is by NONEXCLUSIVE USE LICENSE ONLY. No rights to sublicense, assign or transfer the license are granted and any attempt to do so is void.

3. Third party brands, trademarks, registered trademarks, service marks, and names are the property of their respective owners.

NO REPRESENTATION of THIRD PARTY RIGHTS:

IrDA makes no representation or warranty whatsoever with regard to IrDA member or third party ownership, licensing or infringement/non-infringement of intellectual property rights. Each recipient of IrDA publications, whether or not an IrDA member, should seek the independent advice of legal counsel with regard to any possible violation of third party rights arising out of the use, attempted use, reproduction, distribution or public display of IrDA publications.

IrDA assumes no obligation or responsibility whatsoever to advise its members or non-members who receive or are about to receive IrDA publications of the chance of infringement or violation of any right of an IrDA member or third party arising out of the use, attempted use, reproduction, distribution or display of IrDA publications.

LIMITATION of LIABILITY:

BY ANY ACTUAL OR ATTEMPTED USE, REPRODUCTION, DISTRIBUTION OR PUBLIC DISPLAY OF ANY IrDA PUBLICATION, ANY PARTICIPANT IN SUCH REAL OR ATTEMPTED ACTS, WHETHER OR NOT A MEMBER OF IrDA, AGREES TO ASSUME ANY AND ALL RISK ASSOCIATED WITH SUCH ACTS, INCLUDING BUT NOT LIMITED TO LOST PROFITS, LOST SAVINGS, OR OTHER CONSEQUENTIAL, SPECIAL, INCIDENTAL OR PUNITIVE DAMAGES. IrDA SHALL HAVE NO LIABILITY WHATSOEVER FOR SUCH ACTS NOR FOR THE CONTENT, ACCURACY OR LEVEL OF ISSUE OF AN IrDA PUBLICATION.

DISCLAIMER of WARRANTY:

All IrDA publications are provided "AS IS" and without warranty of any kind. IrDA (and each of its members, wholly and collectively, hereinafter "IrDA") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND WARRANTY OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS.

IrDA DOES NOT WARRANT THAT ITS PUBLICATIONS WILL MEET YOUR REQUIREMENTS OR THAT ANY USE OF A PUBLICATION WILL BE UN-INTERRUPTED OR ERROR FREE, OR THAT DEFECTS WILL BE CORRECTED. FURTHERMORE, IrDA DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING USE OR THE RESULTS OR THE USE OF IrDA PUBLICATIONS IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN PUBLICATION OR ADVICE OF A REPRESENTATIVE (OR MEMBER) OF IrDA SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY.

LIMITED MEDIA WARRANTY:

IrDA warrants ONLY the media upon which any publication is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of distribution as evidenced by the distribution records of IrDA. IrDA's entire liability and recipient's exclusive remedy will be replacement of the media not meeting this limited warranty and which is returned to IrDA. IrDA shall have no responsibility to replace media damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE MEDIA, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM PLACE TO PLACE.

CERTIFICATION and GENERAL:

Membership in IrDA or use of IrDA publications does NOT constitute IrDA compliance. It is the sole responsibility of each manufacturer, whether or not an IrDA member, to obtain product compliance in accordance with IrDA rules for compliance.

All rights, prohibitions of right, agreements and terms and conditions regarding use of IrDA publications and IrDA rules for compliance of products are governed by the laws and regulations of the United States. However, each manufacturer is solely responsible for compliance with the import/export laws of the countries in which they conduct business. The information contained in this document is provided as is and is subject to change without notice.

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This document presents the rules and restrictions that apply to any device implementing all or part of the IrDA Telecom specification.

1.2 Scope

This document describes the infrared interface between devices adhering to the phase 1 IrDA Telecom specification. The phase 1 specification is limited in scope to a set of applications. This set consists of exchanging phone book or contact directory information, calendar information, alphanumeric messages, and device information. In addition to this kind of object exchange call control and full-duplex, two-way audio interfaces are specified.

Data stream services are referenced but no new specifications are made since these services are specified by [IrCOMM].

This document is intended to be a companion document to the IrDA IrPHY, IrLAP, IrLMP, Lite, Ultra and IrOBEX specifications (see section 1.3 References). The referenced specifications provide the complete description of the respective protocols.

1.3 References

GSM 04.08	GSM 04.08 European digital cellular telecommunications system (phase 2): Mobile radio interface Layer 3 Specification
GSM 07.07	GSM 07.07 Digital cellular telecommunications system (Phase 2+); AT command set for GSM Mobile Equipment (ME)
IrCOMM	'IrCOMM': Serial and Parallel Port Emulation over IR (Wireless Replacement), Version 1.0, Infrared Data Association
IrLAP	Serial Infrared Link Access Protocol, IrLAP, Version 1.1, Infrared Data Association
IrLMP	Link Management Protocol, IrLMP, Version 1.1, Infrared Data Association
IrPHY	Serial Infrared Physical Layer Link Specification, IrPHY, Version 1.2, Infrared Data Association
ITU-T V.25ter	ITU-T Recommendation V.25ter: "Serial asynchronous automatic dialing and control", 08/1995, International Telecommunication Union
LITE	Minimal IrDA Protocol Implementation, IrDA Lite, Version 1.0, Infrared Data Association
OBEX	IrDA Object Exchange Protocol, IrOBEX, Version 1.0, Infrared Data Association
ULTRA	Guidelines for Ultra Protocols, Version 0.5
RFC822	RFC #822 - Standard for the Format of the Arpa Internet Text Messages
VCARD	"vCard - The Electronic Business Card Exchange Format - Version 2.1", The Internet Mail Consortium (IMC), September 18, 1996, http://www.imc.org/pdi/vcard-21.doc
VCALENDAR	"vCalendar - the Electronic Calendaring and Scheduling Format - Version 1.0", The Internet Mail Consortium (IMC), September 18, 1996, http://www.imc.org/pdi/vcal-10.doc

2. Telecom Framework

The IrDA Telecom specification defines the rules for utilization of IR in wireless communications equipment, e.g., in mobile handsets and pagers. The phase 1 specification handles exchange of objects such as business cards, calendar items, device information, and alphanumeric messages. Also voice communication and call control utilization through an optical interface are defined.

The Telecom phone book application enables easy exchange of business cards and phone book entries between various wireless communication devices or between wireless communications devices and PCs. Similarly, the messaging specification defines the rules for message exchange among pagers and handsets as well as between those mobile devices and personal computers. Finally, the calendar application allows calendar information access between handsets, pagers and PCs. The call control and voice applications enable mobile handset control and real time audio transmission respectively. These two applications are for communication between a handset and PC or a handset and car cradle.

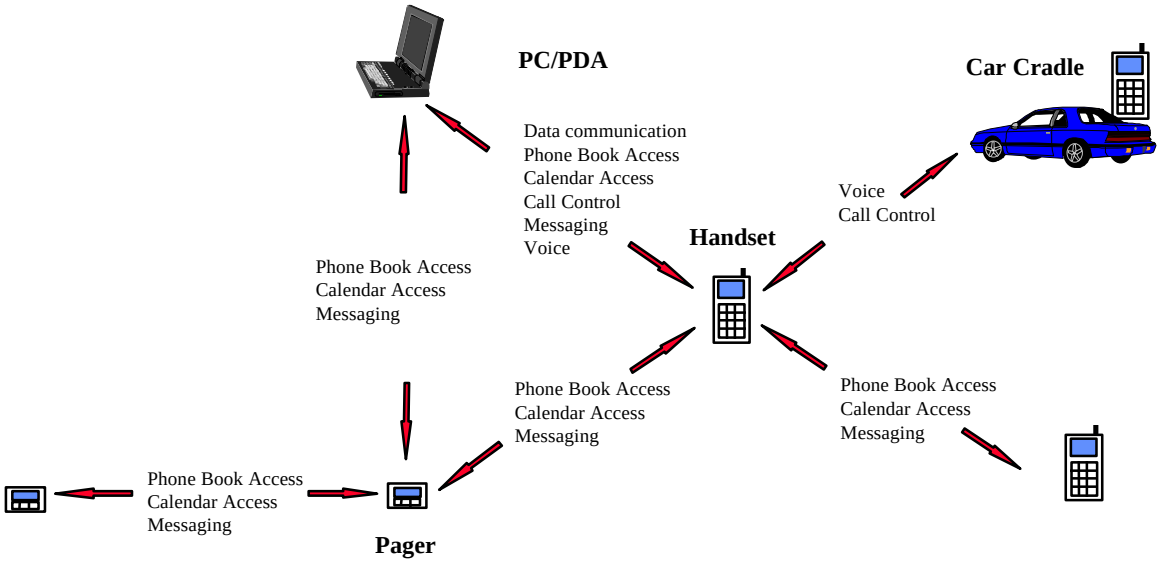


Figure 2-1 Telecom phase 1 applications

Secure phone book/calendar/message access may be implemented, for example, by a PIN code request. It should be noted, however, that no passwords, PIN codes or other means of security should be exchanged over IR due to the insecure nature of the transmission media.

The Telecom framework uses the existing IrDA specifications as much as possible. IrDA connection-oriented services are used as is, and the connectionless service (Ultra) is further defined to be applicable to Telecom devices.

Because the power consumption requirements of the Telecom devices are very critical a low power option for these devices has been specified in the IrPHY version 1.2. This specification defines a lighter physical layer and allows up to ten times savings in the LED drive current. It should be noted that this specification, which only expects the Telecom devices to be capable of 20 cm link distance from Telecom device to Telecom device and 30 cm from Telecom device to a standard IrDA device, is optional and the original IrDA-SIR values may be used in all Telecom device implementations. Despite the shorter communication range of some Telecom devices, all devices must clearly be compatible with the IrDA-SIR in the short range.

Devices supporting IrCOMM communications with personal computers should realize that the RF interference shielding of PCs may be imperfect. As the interference may cause malfunction in the communicating devices or the IR link, short communication range is not always desirable.

The IR functionality of a Telecom device can be divided into three categories based on the nature of the communications:

- Atomic information exchange
- Stream-oriented information exchange
- Time-bounded information exchange

Atomic information exchange

Exchange of information objects, such as phone book entries, calendar items and alphanumeric messages, falls into this category. Atomic information exchange does not necessarily require a connection. Thus simple functions such as transferring a business card can be implemented with connectionless data only. More complicated functions such as synchronization of a phone book or a calendar require, however, connection establishment. Where the capability to establish a connection exists, the connection-oriented transfer method should be used.

All atomic information exchange is based on the use of OBEX-operations. Information related to the Telecom specification and accessed through OBEX uses the Telecom-naming convention, i.e. <phone-book-stream-object-name>, <calendar-indexed-object-name>, etc.

Stream-oriented information exchange

Data services of cellular networks typically fall into this category. The specifications for the stream-oriented information exchange already exist; services for centronics, 3wire, 9wire, IrTA (infrared terminal adapter) and IrGW (infrared gateway) access are defined by the [IrCOMM]. The Telecom specifications do not redefine these services although they are important for any device implementing stream-oriented information exchange.

Time-bounded information exchange

Of the time-bounded services this specification defines the real-time audio service.

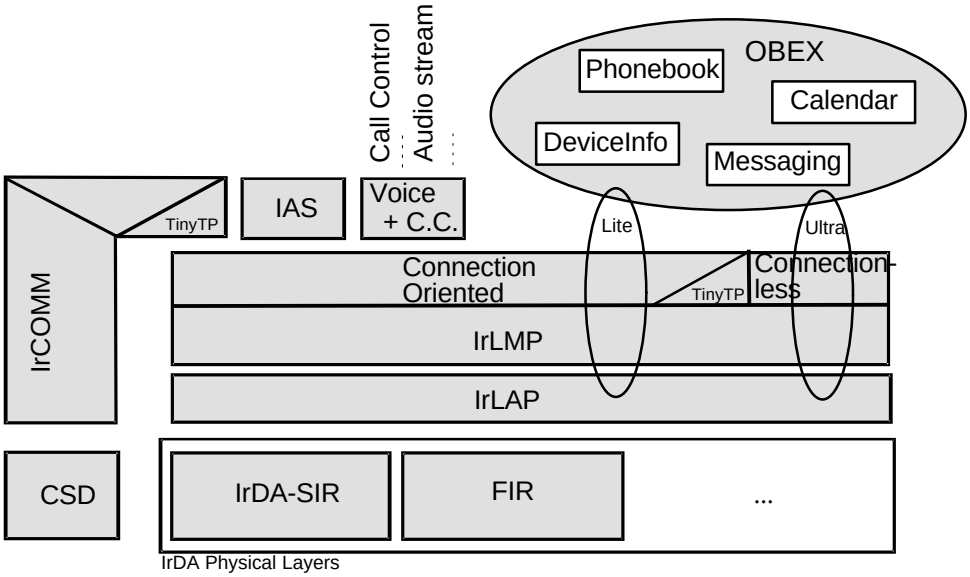


Figure 2-2 The IrDA Telecom Framework

3. Data Transmission Services

Device, phone book, calendar and message information is accessible both through a connection-oriented service and a connectionless service. Establishment of a connection is recommended for all devices capable of supporting this. For two-way communication functions, such as synchronization of phone books and requesting specific calendar information, connection is mandatory.

3.1 Connection Oriented Service

Connection-oriented services rely on the error-free link provided by the IrLAP and IrLMP protocols. Any connection-oriented phone book operation starts with an OBEX connect procedure and ends with an OBEX disconnect procedure. For more information about the OBEX server-client operation, please refer to [OBEX].

3.2 Connectionless Service

Connectionless services are used for offering information between devices with limited communications capabilities. Relying on the Ultra protocol allows data transmission without any connect or disconnect procedures. Devices implementing the connectionless service thus only need to support the OBEX PUT operation.

Ultra does not provide any error correction facility. However, every frame received with correct Frame Checksum Sequence (FCS) contains error-free data. The only way to inform the end user about correctly received data is to indicate this on the user interface of the receiving device.

IrDA Telecom devices must follow the recommendation in the Ultra specification that limits the maximum service data field size to 62 octets in length. Connectionless service is thus limited to use frames with maximum payload size of 60[1] octets. The Segmentation and Re-assembly (SAR) functionality in Ultra can be used to send data that would not fit into the 60 octets available in one connectionless frame. There is no IrDA Telecom specific limitation on the number of frames in the sequence, i.e., up to 15 fragments (900 octets) can be sent.

Normal IrLAP 1.1 MAC rules apply to IrDA Telecom devices, thus the link speed is limited to 9600 BPS, and a media sense period is required between each of the sent frames.

Each frame should never include more than one device/phone book/calendar/message object. Objects should not be combined into one same frame.

It should be noted that each frame must carry the XBOFS specified by the IrLAP specification. This is essential in order to allow the receiver some time for processing the incoming frames and thus avoid flow control problems.

4. OBEX Information Access

OBEX GET and PUT operations are used for accessing phone book, calendar, messaging, or device information. The access schemes for each of these applications are similar. With the exception of vCard/ vCalendar/ vMessage objects that are pushed into a device inbox, the object names passed to GET and PUT should always include the path information.

4.1 Read Access

For reading phone book, calendar, message, or device entries, OBEX GET-request/response is used. Each GET-request carries the NAME of the object to be read. In order to help the receiver identify the object to be read, the name should indicate the format of the object with an appropriate extension. The extension for phone book is VCF and for calendar VCS. Messages are identified by a VMG extension and device information by a TXT extension.

Header Identifier	Value	Description
NAME	object name with an extension	name.extension The name of the requested object with an appropriate extension, e.g., VCF for a phone book object, VCS for a calendar object, VMG for a message object and TXT for a device information object[1] .

A remote device responds to an OBEX GET request with an OBEX GET response. This response carries a response code that indicates the success of the GET operation.

SUCCESS
If the requested object was found and read successfully, the GET response carries a "success" code 0x0A. The contents of the vCard/vCalendar/vMessage/device information are included in the BODY part of the response.

EMPTY ENTRY
If the requested phone book/calendar/message/device information entry is empty, the GET response carries a "success" response code 0x0A with no response data, i.e. no BODY part is present.

It should be noted that devices with dynamic memory structure can use the successful empty entry return to indicate free memory space in which an entry can be written to.

RESTRICTED ACCESS
Some phone book/calendar/message/device information entries may have a limited read access. If an unauthorized GET request is made to one of these entries, the GET response is to indicate failure with failure code "unauthorized" 0xC1. No response data is included in this case.

INDEX NOT FOUND
If a read request is made to a non-existent phone book/calendar/message/device information entry, the response is to carry a failure code of "not found" 0xC4. No response data is included.

For more information about the OBEX GET-client/server, please refer to the [OBEX].

[1] As .TXT is a very common extension, it is recommended to always use the specific file name “telecom/devinfo.txt” to avoid any confusion with other .TXT files.

Phone Book Read Example: Consecutive read requests to indexed vCard entries with static range of indices [1...8] and five entries would return the following results.

	/telecom/pb/0.vcf	- >	Success	[0xA0] + contents of the entry
	/telecom/pb/1.vcf	- >	Success	[0xA0] + contents of the entry
[empty]	/telecom/pb/2.vcf	- >	Success	[0xA0]
	/telecom/pb/3.vcf	- >	Success	[0xA0] + contents of the entry
[limited access]	/telecom/pb/4.vcf	- >	Unauthorized	[0xC1]
	/telecom/pb/5.vcf	- >	Success	[0xA0] + contents of the entry
	/telecom/pb/6.vcf	- >	Not Found	[0xC4]
	/telecom/pb/7.vcf	- >	Not Found	[0xC4]

Corresponding read requests to indexed vCard entries with dynamic range of indices would return the following results. It should be noted that here an additional empty entry has been reserved after the last actual vCard. This approach allows indication of free memory space into which an entry can be made.

	/telecom/pb/0.vcf	- >	Success	[0xA0] + contents of the entry
	/telecom/pb/1.vcf	- >	Success	[0xA0] + contents of the entry
[empty]	/telecom/pb/2.vcf	- >	Success	[0xA0]
	/telecom/pb/3.vcf	- >	Success	[0xA0] + contents of the entry
[limited access]	/telecom/pb/4.vcf	- >	Unauthorized	[0xC1]
	/telecom/pb/5.vcf	- >	Success	[0xA0] + contents of the entry
	/telecom/pb/6.vcf	- >	Success	[0xA0]

4.2 Write Access

For writing phone book, calendar, message or device information entries OBEX PUT-request/response is used. Each PUT- request carries the NAME, the LENGTH and the BODY of the object to be written. The name indicates the name of the phone book/calendar/message/device information entry to which the write operation is to be applied as well as the format of the object to be written. The right format for a phone book object is VCF, for a calendar object VCS, for a message object VMG and for a device information object TXT. It is recommended that devices with dynamic range of indices always try to write to the first free index. The length field in the PUT-request indicates the length of the object to be written as specified in the [OBEX]. The contents of the object are included in the body field of the request. The body field may also be non-existent if the write is used for a delete - operation.

Header Identifier	Value	Description
NAME	object name with an extension	name.extension The name of the requested object with an appropriate extension, e.g., VCF for a phone book object, VCS for a calendar object and VMG for a message object..
LENGTH	object length	The length of the object to be written as defined in the OBEX specification.
BODY	PHONE BOOK: vCard stream	The contents of a *.vcf object.
	CALENDAR: vCalendar stream	The contents of a *.vcs object.
	MESSAGING: vMessage stream	The contents of a *.vmg object.

A remote device responds to an OBEX PUT request with an OBEX PUT response. This response carries a response code that indicates the success of the PUT operation.

SUCCESS
If the phone book/calendar/message/device information entry write operation was successful, the response carries a "success" code 0x0A. No response data is included.

EMPTY ENTRY
If the phone book/calendar/message/device information write request has no body header, the write action is regarded as an "entry delete". Any previous information is cleared. Moreover, for devices where the distinction between erasing an entry and creating an empty entry makes sense, a phone book/ calendar/ message/ device information write request with an empty body header (0 length) implies the creation of an empty entry. If this distinction is not supported by the device, then a phone book/calendar /message/ device information write request with an empty body header is treated in the same way as a phone book/calendar /message/ device information write request without body header.

It is recommended that an indication about the deletion as well as a possibility to refuse it is given to the end user. If the user decides to reject the deletion, a response indicating failure with failure code "unauthorized" 0xC1 shall be returned. No response data is included.

If the entry delete is accepted by the end user and is successful, the response carries a "success" response code 0xA0 with no response data.

RESTRICTED ACCESS
Some phone book/calendar/message/device information entries may have a limited write access. If an unauthorized PUT request is made to one of these entries, the PUT response is to indicate failure with failure code "unauthorized" 0xC1. No response data is included.

INDEX NOT FOUND
If a write request is made to a non-existent phone book/calendar/message/device information entry, the response is to carry a failure code of "not found" 0xC4. No response data is included in this case.

For more information about the OBEX PUT -client/server, please refer to the [OBEX].

Calendar Write Example: Calendar entry write operations to consecutive calendar indices. As explained previously an "unauthorized" response may be caused either by an access limitation or by an user originated write-rejection.

[limited access]	/telecom/pb/0.vcs	&	no body	- >	Unauthorized	[0xC1]
	/telecom/pb/1.vcs	&	vCalendar body	- >	Success	[0xA0]
	/telecom/pb/2.vcs	&	no body	- >	Success	[0xA0]
[delete rejected]	/telecom/pb/3.vcs	&	no body	- >	Unauthorized	[0xC1]
	/telecom/pb/4.vcs	&	vCalendar body	- >	Not Found	[0xC4]

4.3 Restricted Access

Read/write access to OBEX database objects may be restricted depending on the device implementation and the information to be accessed. OBEX enables password exchange procedure during the setup of the OBEX service connection. However, authentication based on this kind of remote device password query is insecure due to the nature of infrared transmission medium. Generally, password, PIN code, etc. exchange over IR is not recommended.

A more recommendable practice is to prompt for a password/PIN code in the user interface of the device whose data is to be accessed. It is dependent on the device implementation whether one or more security levels exist. Access to some of the objects may be denied depending on the security level chosen.

4.4 Avoiding "Race Condition"

Simultaneous phone book/calendar/message access from multiple communication channels (one being infrared) may cause a "race condition" in which some of the applications accessing a piece of information have an

outdated copy of it. In order to avoid such a condition, it is recommended that some way of prioritizing the communications channels is implemented. The details of prioritization or access limitation are implementation specific issues and thus out of the scope of this paper.

It should be noted that despite possible access limitations no information should be lost.

4.5 Information Access Examples

4.5.1 Ultra Write

Two devices with limited connection establishment capabilities. Device1 pushes a phone book entry to the Device2 by using the Ultra protocol. As shown in the Figure 4.5-1 no connection establishment is required, OBEX PUT command is used to originate the data transfer and all user data is carried in unnumbered frames. No acknowledgment of the received information is sent or expected.

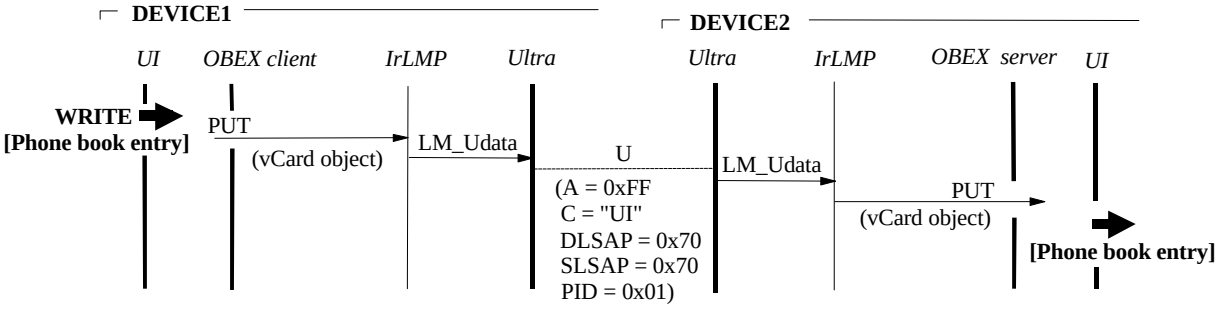


Figure 4.5-1 Object push with Ultra

4.5.2 Connection Oriented Read

Two devices with calendar access level support. Device1 requests a calendar stream read operation as shown in the Figure 4.5-2. OBEX CONNECT-command is used to set up the connection layer by layer starting from IrLAP. OBEX GET-command is then used for reading the calendar stream. GET-request is confirmed by Device2 with a GET-response which carries the requested stream object. All user data is transmitted in IrLAP information frames. Once the object read is complete the connection is disconnected with OBEX DISCONNECT-command.

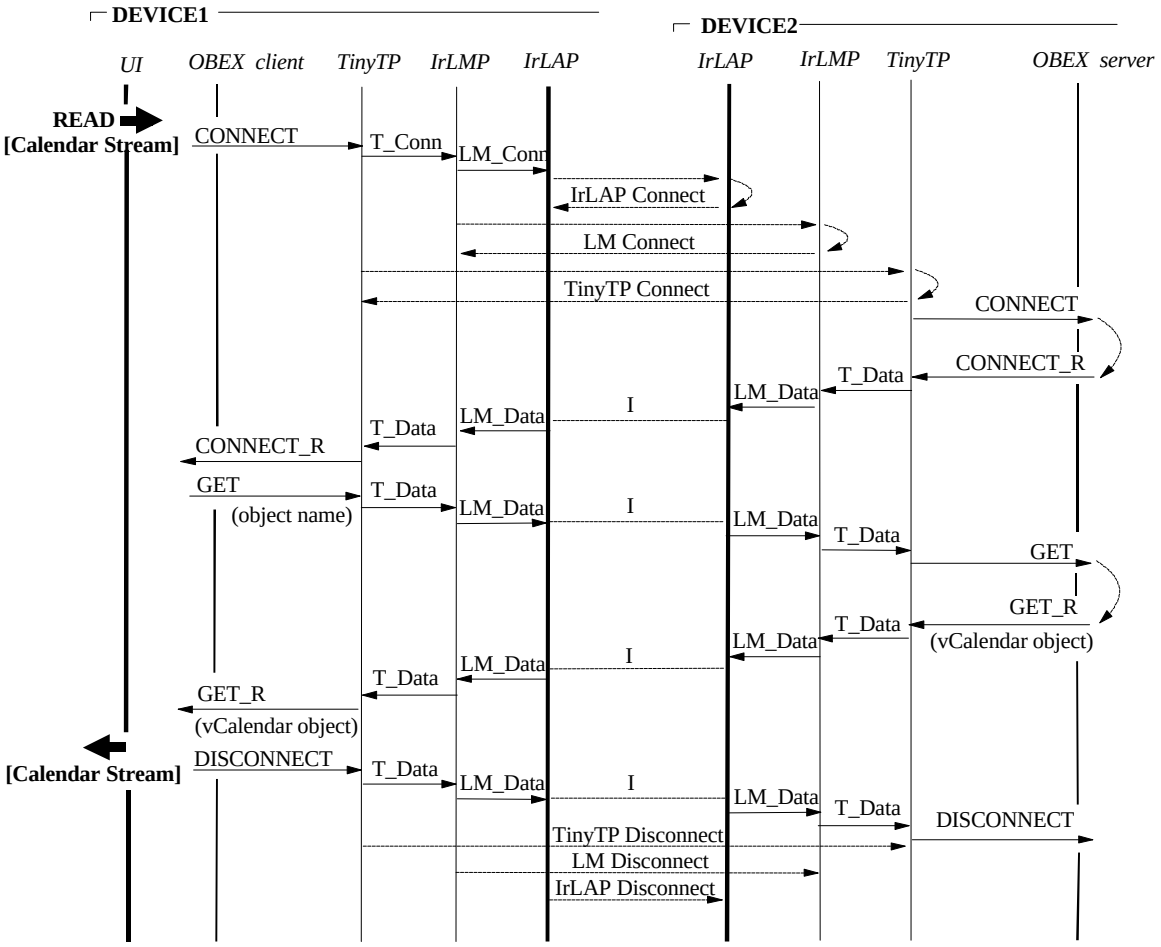


Figure 4.5-2 Object read

5. Notation

The notation used in this document uses the common elements presented here. Backus-Naur Form (BNF) notation is used to describe the formats. The BNF format and extensions used are as follows:

- In BNF notation "::=" means "definition", where a non-terminal symbol is on the left side of the operator "::=", and the definition is on the right side.
- The symbol order in BNF notation is the same as the syntax symbol order.
- Terminal symbols are enclosed between quotes ("), and the symbols are written in **bold**.
- Non-terminal symbols are enclosed between < and > characters, and the symbols are written in *italics*.
- Textual definitions for non-terminal characters are enclosed between apostrophes (‘).
- Operator "|" is used as a delimiter between multiple choices.
- Grouping of items is expressed by enclosing them in meta symbols { and }.
- Optional parts are enclosed in meta symbols [and].
- Kleene’s "+" operator is supported, i.e., <A>$^{+}$ means repetition of non-terminal <A> from **1** to ∞ times.
- Kleene’s "*" operator is supported, i.e., <A>* means repetition of non-terminal <A> from **0** to ∞ times.
- Semicolon symbol ";" means that the rest of the line contains comments.

5.1 Common Elements

Character Definitions

<*line-feed*> ::= ‘Character with ASCII value 10 decimal.’

<*space*> ::= ‘Character with ASCII value 32 decimal’

<*default-char*> ::= ‘Any character in the default character-set (or in character-set explicitly indicated).’

<*default-char-not-lf*> ::= ‘Any character in the default character-set (or in character-set explicitly indicated) except <*line-feed*>.’

<*common-digit*> ::= “**1**” | “**2**” | “**3**” | “**4**” | “**5**” | “**6**” | “**7**” | “**8**” | “**9**” | “**0**”

<*common-white-space*> ::= <*space*>*

Date/Time Definitions

<*common-date*> ::= <*year*><*month*><*day*> [<*time*> [<*type-designator*>]]

;’The basic format of ISO 8601 for date and time. for example, 19971031T231210’.

<*year*> ::= <*common-digit*> <*common-digit*> <*common-digit*> <*common-digit*> ; ‘i.e., “**1997**”’

<*month*> ::= <*common-digit*> <*common-digit*> ; ‘i.e., “**03**”’ for March’

<*day*> ::= <*common-digit*> <*common-digit*> ; ‘i.e., “**08**”

<*time*> ::= “**T**” <*hours*> <*minutes*> <*seconds*>

<*hours*> ::= <*common-digit*> <*common-digit*> ; ’24-hour format’

<*minutes*> ::= <*common-digit*> <*common-digit*>

<*seconds*> ::= <*common-digit*> <*common-digit*>

<*type-designator*> ::= “**Z**” ; ’This means the time is Universal Time Coordinated (UTC)’

External Definitions

<*common-vcard*> ::= ‘vCard object as defined in [VCARD].’

<*common-vcalendar*> ::= ‘vCalendar object as defined in [VCALENDAR].’

6. Device Information

The manufacturer-specific information of a Telecom device is stored as a <device-info-object> with the name <device-info-object-name>. This object is mandatory for all Telecom devices and may have restricted write-access.

The <device-info-object> includes several property identifiers, which define individual attribute values associated with the device. The property names are defined in section 6.1. The property values are expressed as property strings. Encoding, Character set and Language property parameters as defined in the vCard specification [VCARD] are used for the properties of the object.

This object can be used, for example, in device programming for providing information about the model and firmware/software versions of the device.

6.1 Property Identifiers

6.1.1 Manufacturer

The Manufacturer property identifier specifies the manufacturer of the device. This property identifier is mandatory.

MANU:Big Factory, Ltd.

6.1.2 Model

The Model property identifier specifies the model of the device. This property identifier is mandatory.

MOD:4119

6.1.3 OEM

The OEM property identifier specifies the OEM-manufacturer of the device. This property identifier is optional.

OEM:Jane’s phones

6.1.4 Firmware-Version

The Firmware-version property identifier specifies the version of the firmware created. This property identifier is optional. The format of the field is manufacturer dependent.

FW-VERSION:2.0e

6.1.5 Firmware-Date

The Firmware-date property identifier specifies the date of the firmware version created. The date format used is the basic format of ISO8601. This property identifier is optional.

FW-DATE:19971031T231210

6.1.6 Software-Version

The software-version property identifier specifies the version of the software created. The format of the field is manufacturer dependent. This property identifier is optional.

SW-VERSION:2.0

6.1.7 Software-Date

The software-date property identifier specifies the date of the software version created. The date format used is the basic format of ISO8601. This property identifier is optional.

SW-DATE:19971031T231210

6.1.8 Hardware-Version

The hardware-version property identifier specifies the version of the hardware created. The format of the field is manufacturer dependent. This property identifier is optional.

HW-VERSION:1.22i

6.1.9 Hardware-Date

The hardware-date property identifier specifies the date of the hardware version created. The date format used is the basic format of ISO8601. This property identifier is optional.

HW-DATE:19971031T231210

6.1.10 Serial Number

The serial-number property identifier specifies the serial number of the device. The format of the field is manufacturer dependent. This property identifier is optional.

SN:1218182THD000001-2

6.1.11 Extensions

Manufacturer specific extension property and/or property parameter names are identified by the initial sub-string of X- (capital X followed by a dash character). It is recommended that vendors concatenate onto this sentinel an added short sub-string to identify the vendor. All Telecom devices are expected to be able to interpret the extensions properties and property parameters but may ignore them.

X-BIGFACTORY-START-UP-MESSAGE:Hello World

6.2 Formal Definition

<*device-info-object-name*> ::= “**telecom/devinfo.txt**”

<*device-info-object*> ::= { <*device-info-object-field*> <*line-feed*> }*

<*device-info-object-field*> ::=
<*device-info-manufacturer-field*> |
<*device-info-model-field*> |
<*device-info-oem-field*> |
<*device-info-firmware-version-field*> |
<*device-info-firmware-date-field*> |
<*device-info-software-version-field*> |
<*device-info-software-date-field*> |
<*device-info-hardware-version-field*> |
<*device-info-hardware-date-field*> |
<*device-info-serial-number-field*> |
<*device-info-extension-field*>

<*device-info-manufacturer-field*> ::=
"**MANU**" <*vcard-properties*>* "**:**"<*default-char-not-lf*>*

<*device-info-model-field*> ::=
"**MOD**" <*vcard-properties*>* "**:**"<*default-char-not-lf*>*

<*device-info-oem-field*> ::=
"**OEM**" <*vcard-properties*>* "**:**"<*default-char-not-lf*>*

<*device-info-firmware-version-field*> ::=
"**FW-VERSION**" <*vcard-properties*>* "**:**"<*default-char-not-lf*>*

<*device-info-firmware-date-field*> ::=
"**FW-DATE:**" <*common-date*>

<*device-info-software-version-field*> ::=
"**SW-VERSION**" <*vcard-properties*>* "**:**"<*default-char-not-lf*>*

<*device-info-software-date-field*> ::=
"**SW-DATE:**" <*common-date*>

<*device-info-hardware-version-field*> ::=
"**HW-VERSION**" <*vcard-properties*>* "**:**"<*default-char-not-lf*>*

<*device-info-hardware-date-field*> ::=
"**HW-DATE:**" <*common-date*>

<*device-info-serial-number-field*> ::=
"**SN"**<*vcard-properties*>* "**:**"<*default-char-not-lf*>*

<*device-info-extension-field*> ::=
"**X-**" <*device-info-extension-manufacturer*> "**-**"
<*device-info-extension-identifier*> <*vcard-properties*>*
"**:**"<*default-char-not-lf*>*

<*device-info-extension-manufacturer*> ::= <*default-char-not-lf*>*
<*device-info-extension-identifier*> ::= <*default-char-not-lf*>*

<*vcard-properties*> ::= "**;**" <*vcard-property-identifier*>
<*vcard-property-identifier*> ::= 'Encoding, Character set and Language property parameters as defined in [VCARD].'

7. Phone Book

7.1 Phone Book Object Format

The object format used for phone book and business card objects is the vCard format as defined in the [VCARD]. The phone book objects are thus identified by the VCF-extension in the name. All the mandatory property fields of vCard, i.e. *Name* and *Version*, must be present. It should be noted that even though the Name identifier is always required in all vCard objects, the name field value may be left empty.

Received vCards may include fields which are not supported by the device. It is recommended that no supplied fields are removed in order to maintain the original content. The Unique Identifier property, if present, should always be stored. Generally, Telecom devices should include the *Telephone Number* field in addition to the mandatory fields in all vCard objects transmitted.

vCard field	Property Name	Description
Name	N	Name of the person, place or thing associated with the vCard.
Version	VERSION	Version number of the vCard specification used in the implementation.
Telephone Number	TEL	Telephone number for the person, place or thing associated with the vCard.

For more information about the vCard structure and the subtypes of the vCard fields, please refer to the [VCARD].

7.2 Phone Book Hierarchy

Figure 7.2-1 shows the four possible levels of phone book support. The way a phone book is organized and accessed depends on the level of support implemented in the device. A device implementing any of the higher support levels also has to provide support of the lower levels. The level of the support should be unambiguously identified by the IAS PhoneBookSupport parameter as described in section 12.1.2.1.

In addition to the four service levels, there are three optional phone book objects which may be implemented regardless of the level of the phone book service supported.

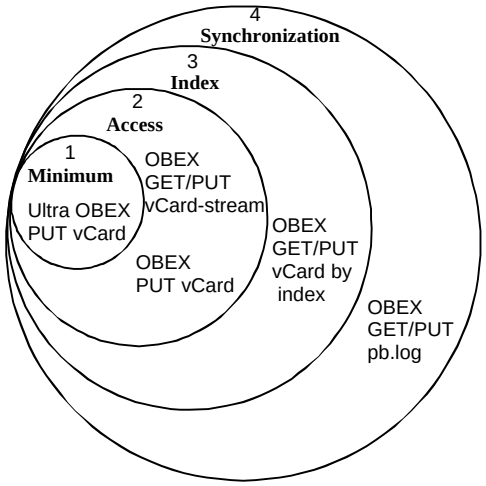


Figure 7.2-1 Phone book support levels

(1) MINIMUM PHONE BOOK SUPPORT

The minimum implementation of a phone book application provides a simple push vCard -functionality. The name of the pushed object is specified as <phone-book-minimum-support-object-name>, and the object as <phone-book-minimum-support-object>.

At the remote end, received vCards are recommended to be stored into an inbox with the original name of the object. Yet, it should be noted that the names of the entries may vary between the originator and the receiver due to various system specific naming rules. No automatic actions should be taken based on the name of the received object, e.g. receiving an object with the <phone-book-stream-object-name> should not cause automatic update of the phone book stream.

Minimum Phone Book Hierarchy Example: Four phone book entries received by a remote device push. All the entries are stored in the device inbox with their original names for further processing.

device inbox: [inbox for storing the vCards received by a remote push]
Yoshiko.vcf
Alex.vcf
Home.vcf
John Smith.vcf

(2) PHONE BOOK ACCESS SUPPORT

Phone book access support provides a read-all/write-all functionality. Devices with the access support implement an object listing all the phone book entries as a vCard stream as shown in the Figure 7.2-2 (note that inbox phone book entries are not taken into account in this vCard stream). This stream is named <phone-book-stream-object-name> and it's format is defined as <phone-book-stream-object>.

The first entry in the <phone-book-stream-object> is always the vCard of the owner/local device. If no local information is available, the first entry will hold an empty vCard. Entries with restricted access are also listed as empty vCards.

If phone book index support is implemented the organization of the vCard stream in the <phone-book-stream-object> must reflect the organization of the single <phone-book-indexed-object> entries.

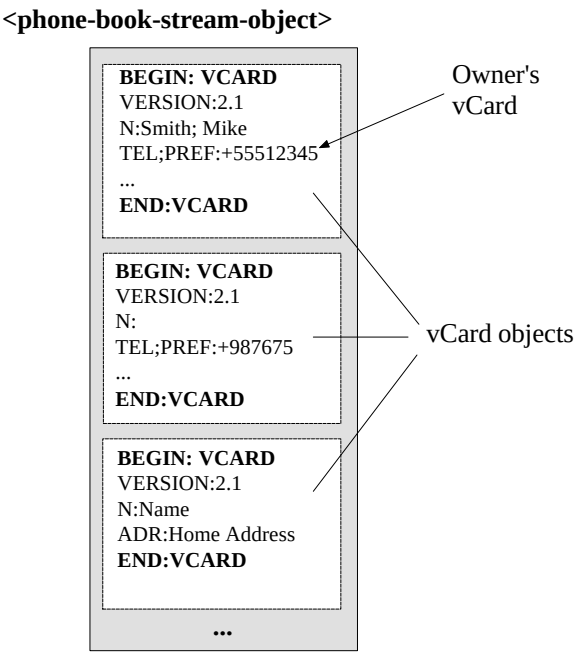


Figure 7.2-2 A stream of all vCards of a phone book

Phone Book Access Hierarchy Example: A phone book stream object "telecom/pb.vcf" allows an easy ready/write access to the entire phone book. Support for connectionless push operation is also implemented. In the device inbox there are four vCards, which have been received by a remote push.

device inbox: [inbox for storing the vCards received by a remote push]
Yoshiko.vcf
Alex.vcf
Home.vcf
John Smith.vcf

telecom/pb.vcf [vCard stream object listing all the phone book entries]

(3) PHONE BOOK INDEX SUPPORT

If read/write access to single phone book entries is supported, each vCard entry is assigned an individual index and named as <phone-book-indexed-object-name>. The format of these objects is defined as <phone-book-indexed-object>. The entry indices are expressed as integers starting from 0. The indices do not correspond to any internally used sorting scheme such as the location of the entry in an alphanumerically organized entry list, but are merely used to indicate the virtual location of a entry in the vCard list. Index "0" always stores the vCard information of the owner/local device. If no local information is available, the location "0" will hold an empty vCard.

The organization of the vCard stream in the level two <phone-book-stream-object> must reflect the organization of the single <phone-book-indexed-object> entries. Accordingly, any changes in any of the phone book index entries must be updated in the phone book stream object. The phone book entry indices may be static or dynamic. If a phone book is based on static indexing, it is up to the implementation whether all empty locations are indicated in the <phone-book-stream-object> as shown in the Figure 7.2-3. This is recommended in order to allow straightforward access to single vCards based on the information given by a read-all operation. Typically, systems with static indexing will return an empty vCard response for read requests into indices with no vCard whereas systems with dynamic indexing return response code "not found". As internal organization of the phone book entries may have significance to the application user, no unauthorized changes in the entry order should be made.

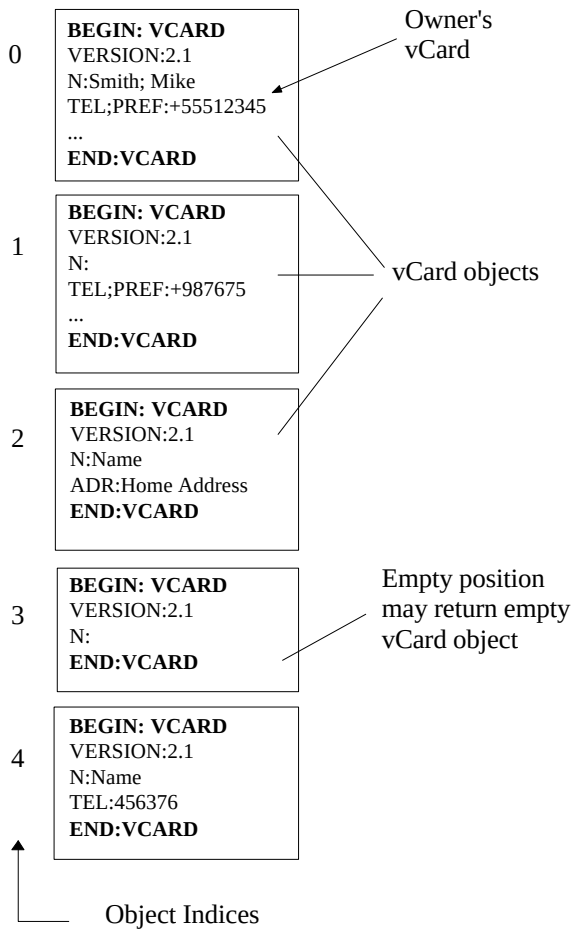


Figure 7.2-3 vCard entries with indices

Phone Book Index Hierarchy Example: Each vCard in the phone book is assigned an individual index. Index "0" holds the vCard of the local device. Read all/write all -operations are provided by including the "telecom/pb.vcf" object. Device inbox holds one entry which has been received by a remote push.

device inbox: Yoshiko.vcf	[inbox for storing the vCards received by a remote push]
telecom/pb.vcf	[vCard stream object listing all the phone book entries]
telecom/pb/**0**.vcf	[vCard of the local device]
telecom/pb/**1**.vcf	[phone book objects with individual indices]
telecom/pb/**2**.vcf	
telecom/pb/**3**.vcf	
telecom/pb/**4**.vcf	

(4) PHONE BOOK SYNCHRONIZATION SUPPORT

Finally, at the phone book synchronization level, an additional log object named <phone-book-modification-log-object-name> is to be implemented. This object, defined as <phone-book-modification-log-object>, is to hold information about all changes in the any of the <phone-book-indexed-object>.

Phone Book Synchronization Hierarchy Example: A phone book log object with information about all changes in the phone book entries in included. Each vCard in the phone book is assigned an individual index. Read all/write all -operations are provided by including the "telecom/pb.vcf" object. Device inbox holds one entry which has been received by a remote push.

device inbox: [inbox for storing the vCards received by a remote push]
Yoshiko.vcf

telecom/pb.vcf [vCard stream object listing all the phone book entries]

telecom/pb/0.vcf [phone book objects with individual indices]
telecom/pb/1.vcf
telecom/pb/2.vcf

telecom/pb/pb.log [phone book modification log object]

7.3 Minimum Support

Devices with limited connection-oriented transmission capabilities may push phone book entries to remote devices using OBEX PUT server on top of Ultra. No connection establishment or data acknowledgment is included in this one-way process.

Received phone book entries are recommended to be stored into an inbox of the remote device. No response is expected to the offer. It is, however, recommended that a user interface indication is given at the remote device about the received phone book entry. New entries should not be saved to the phone book without user intervention.

Also devices with connection support are required to provide phone book entry push on top of Ultra. This service is mandatory for all devices with a Telecom Phone Book application.

Phone Book Entry Push Example:

Local Device **PUT** [name - Celia.vcf]

Remote Device

device inbox:
Celia.vcf

UI indication "Do you want to save this number into your phone book? "

7.4 Phone Book Access Support

The phone book access support service relies on the connection-oriented service provided by the IrLAP and IrLMP layers. Devices with phone book access support provide functions for reading, writing, and deleting an entire phone book. These devices have all the entries of a phone book listed in a <phone-book-stream-object>.

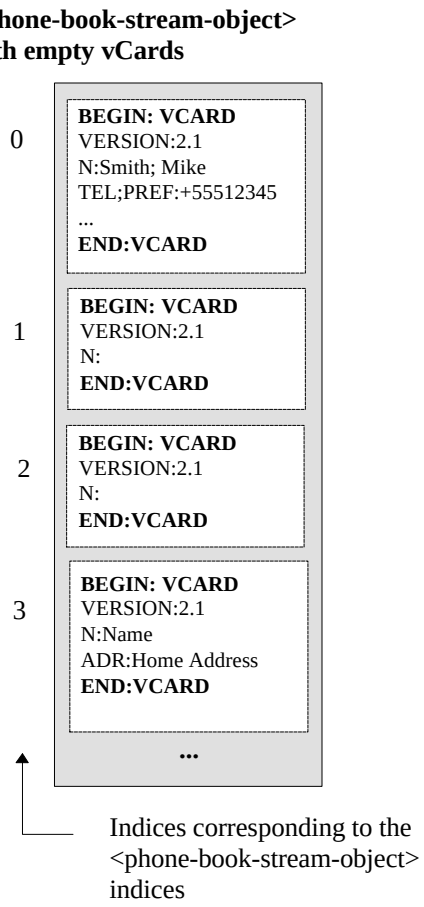


Figure 7.4-1 A phone book stream with an empty vCard

When reading or writing all the entries of a phone book, all empty locations may be indicated by an empty vCard as shown in the Figure 7.4-1. This way it is possible to maintain the original location of the entries in the list. If empty vCards (Figure 7.4-2) are indicated these have to be taken into account by the receiver. If no new phone book entries are made, the relative order of the entries remains the same in a read-write -operation. It should be noted that this is not necessarily true for a write-read -operation due to possible internal sorting schemes applied to the list.

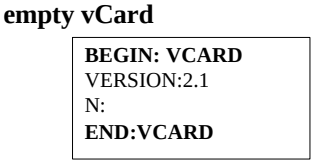


Figure 7.4-2 An empty vCard

The <phone-book-stream-object> may be deleted by writing an empty entry on top of the original entry, i.e., by applying a PUT operation without a BODY to the original object. If a phone book synchronization support is provided, <phone-book-stream-object> should always be read during a synchronization operation.

OBEX GET and PUT operations used for reading and writing phone book entries are explained in the section 4 OBEX Information Access.

Phone Book Access Support Example:

Local Device	**GET**	[name - telecom/pb.vcf]
Remote Device	**GET response**	[response code - success body - telecom/pb.vcf]

7.5 Phone Book Index Support

The phone book index access support relies on the connection-oriented service provided by the IrLAP and IrLMP layers. Devices with this level of service allow reading, writing, modifying and deleting individual phone book entries. These devices have the phone book entries organized as <phone-book-indexed-object> as explained in the section 7.2 Phone Book Hierarchy.

When modifying a single phone book entry, the entry is first read from the phone book, then edited. The new entry is written back to the same index in the phone book. If the phone book synchronization support is provided, each new phone book entry, as well as modification of a phone book entry, should be listed in the <phone-book-modification-log-object>.

A single entry in a phone book may be deleted by a PUT request into that index with no body included in the request. Again, if phone book synchronization support is provided, each phone book entry delete should be listed in the <phone-book-modification-log-object>.

OBEX GET and PUT operations are used for reading and writing phone book entries as explained in the section 4 OBEX Information Access. It should be noted that during an OBEX- connection, the indexing of the objects must not be affected by the writing of a new entry.

Phone Book Index Support Example:

Local Device	**GET**	[name - telecom/pb/5.vcf]
Remote Device	**GET response**	[response code - unauthorized]
Local Device	**GET**	[name - telecom/pb/4.vcf]
Remote Device	**GET response**	[response code - success body - telecom/pb/4.vcf]
Local Device	**EDIT 4.vcf** **PUT**	**[**name - telecom/pb/4.vcf]

7.6 Phone Book Synchronization Support

For synchronizing phone books of different devices, information about any changes or additions in the phone book entries is to be stored in a modification log <phone-book-modification-log-object>. Starting from the oldest modification, this log lists in chronological order all the changed objects in a phone book. Some implementations may list a changed entry several times while others only list each entry once even if there have been several separate modifications to some of the items. It should be noted that the entries marked as ‘changed’ in the synchronization log may be totally new objects which have nothing in common with the original entry.

Phone book synchronization is most effective when there are means to uniquely identify each entry in the phone book. Because of this, devices with true synchronization requirements are recommended to implement the Unique Identifier property field of the vCard. This optional vCard property, identified by the name UID, enables globally unique representation of the vCard entries and allows thus reliable synchronization functionality. For more information about the UID values, please refer to the vCard specification.

When synchronization is performed, each of the changed phone book entries listed in the modification log is read. Also the phone book stream object is read. Once synchronization is finished, it is the responsibility of the

remote device performing the synchronization to clear the modification log. This is done by deleting the object using an OBEX PUT operation to the <phone-book-modification-log-object> with a non-existent body.

If there are too many modifications to be included in a log, all changes will be ignored and the log will only list one item, "*", which indicates that the system is unable to list all the changes. This situation may be due to storage space limitations and means that the whole phone book must be synchronized.

Phone Book Synchronization Example: Figure 7.6-1 lists several possible synchronization log objects.
(i) A log object indicating changes in three vCard objects; (ii) A log object indicating changes in three vCard objects, modification time included; (iii) A log object indicating several changes in some vCard objects, modification time included; (iv) A log object with too many modifications to be listed; (v) No log object.

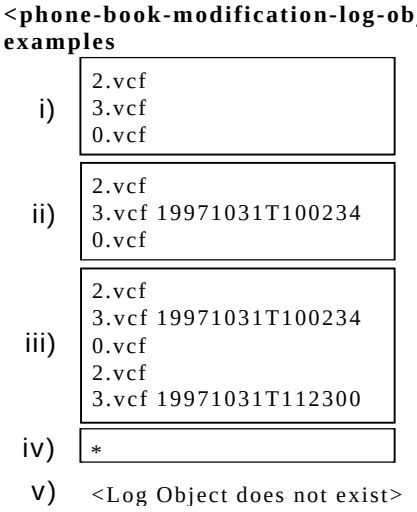


Figure 7.6-1 Phone book log objects

7.7 Call History Objects

In order to allow access to the latest received, made and missed calls devices may implement optional call history objects. These objects may be read, wrote and deleted in the same way as the <phone-book-stream-object> by using the OBEX GET and PUT operations.

7.7.1 Incoming Calls

To allow access to the information of the latest received calls, a <phone-book-incoming-call-history-object> object may be included.

This object is a stream of vCard entries, each with an individual index. This indexing is dynamic so that the information of the latest call always has the index 1. Similarly, the call received before the last one has index 2 and so on (Figure 7.7-1).

If the number of an incoming call is for some reason unavailable, an empty entry is included into the stream to maintain the order of the calls.

The support of this object is optional. If the object is supported, it should be unambiguously stated by the IAS PhoneBookOptional parameter as described in section 12.1.2.1.

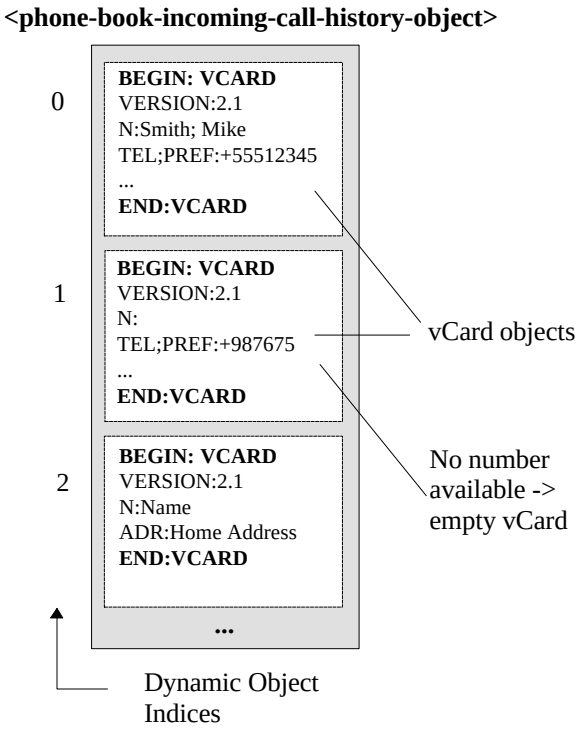


Figure 7.7-1 Incoming call history stream

7.7.2 Outgoing Calls

To allow access to the information of the outgoing calls, a <phone-book-outgoing-call-history-object> object may be included. This object is a stream of vCard entries, each with an individual index. This indexing is dynamic so that the information of the latest outgoing call always has the index 1. Similarly, the call made before the last one has index 2 and so on.

The support of this object is optional. If the object is supported, it should be unambiguously stated by the IAS PhoneBookOptional parameter as described in section 12.1.2.1.

7.7.3 Missed Calls

To allow access to the information of the latest missed calls, a <phone-book-missed-call-history-object> object may be included. This object is a stream of vCard entries, each with an individual index. This indexing is dynamic so that the information of the latest missed call always has the index 1. Similarly, the call missed before the last one has index 2 and so on.

If the number of a missed call is for some reason unavailable, an empty entry is included into the stream to maintain the order of the calls.

The support of this object is optional. If the object is supported, it should be unambiguously stated by the IAS PhoneBookOptional parameter as described in section 12.1.2.1.

7.8 Formal Definition

7.8.1 Minimum Support

<*phone-book-minimum-support-object-name*> ::= <*default-char-not-lf*> "**.vcf**"

<*phone-book-minimum-support-object*> ::= <*common-vcard*>

7.8.2 Phone Book Access Support

<*phone-book-stream-object-name*> ::= "t**elecom/pb.vcf"**

<*phone-book-stream-object*> ::= <*common-vcard*>*

7.8.3 Phone Book Index Support

<*phone-book-indexed-object-name*> ::= "t**elecom**/**pb/**" <*digit*>+ "**.vcf**"

<*phone-book-indexed-object*> ::= <*common-vcard*>

7.8.4 Phone Book Synchronization Support

<*phone-book-modification-log-object-name*> ::= "**telecom/pb/pb.log**"

<*phone-book-modification-log-object*> ::= <*several-modifications-indication*> | <*modified-entry*>*

<*several-modifications-indication*> ::= "*****" <*line-feed*>

<*modified-entry*> ::= <*entry-name*> [<*space*> <*entry-modify-date*>] <*line-feed*>

<*entry-name*> ::= <*phone-book-indexed-object-name*>,name of the object, i.e., "**2.vcf**".'

<*entry-modify-date*> ::= <*common-date*>

7.8.5 Call History Objects

7.8.5.1 Incoming Calls

<*phone-book-incoming-call-history-object-name*> ::= "t**elecom/ich.vcf**"

<*phone-book-incoming-call-history-object*> ::= <*common-vcard*>*

7.8.5.2 Outgoing Calls

<*phone-book-outgoing-call-history-object-name*> ::= "**telecom/och.vcf**"

<*phone-book- outgoing -call-history-object*> ::= <*common-vcard*>*

7.8.5.3 Missed Calls

<*phone-book-missed-call-history-object-name*> ::= "**telecom/mch.vcf**"

<*phone-book-missed-call-history-object*> ::= <*common-vcard*>*

8. Calendar

8.1 Calendar Object Format

The object format used for calendar objects is the vCalendar format as defined in the [VCALENDAR]. These objects are identified by a VCS-extension in the name. Each calendar object has to include the mandatory vCalendar VERSION- property field. In addition to this, Telecom Calendar implementations are expected to be able to handle the fields specified as mandatory in the [VCALENDAR].

Received calendar objects may include fields which are not supported by the device. It is recommended that no supplied fields are removed in order to maintain the original content. The Unique Identifier property, if present, should always be stored.

For more information about the vCalendar structure and the Simplegram encoding, please refer to the [VCALENDAR].

8.2 Calendar Hierarchy

Figure 8.2-1 shows the four possible levels of the calendar support. The way a calendar is organized and accessed depends on the level of support implemented in the device. A device implementing any of the higher support levels also has to provide support of the lower levels. The level of the support should be unambiguously identified by the IAS CalendarSupport parameter as described in section 12.1.2.2.

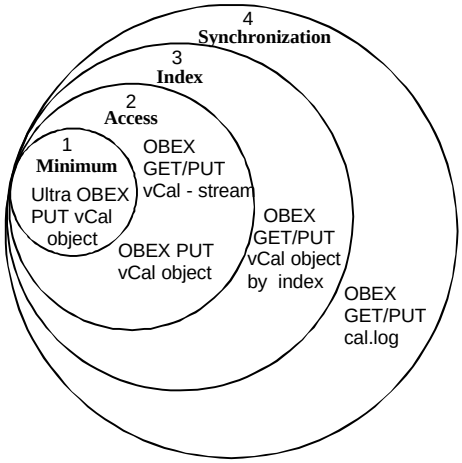


Figure 8.2-1 Calendar support levels

(1) MINIMUM CALENDAR SUPPORT

The minimum implementation of a calendar application provides a simple push vCalendar object -functionality. The name of the pushed object is specified as <calendar-minimum-support-object-name>, and the object as <calendar-minimum-support-object>.

At the remote end, received vCalendar entries are recommended to be stored into an inbox with the original name of the object. Yet, it should be noted that the names of the entries may vary between the originator and the receiver due to various system specific naming rules. No automatic actions should be taken based on the name of the received object, e.g. receiving an object with the <calendar-stream-object-name> does not cause automatic update of the calendar stream.

Minimum Calendar Hierarchy Example: Four calendar entries received by a remote device push. All the entries are stored in the device inbox with their original names for further processing.

device inbox: [inbox for storing the vCalendar objects received by a remote push]
- meeting.vcs
- reminder.vcs
- call.vcs
- party.vcs

(2) CALENDAR ACCESS SUPPORT

Calendar access support provides a read-all/write-all functionality. Devices with the access support implement an object listing all the calendar entries as a vCalendar stream as shown in the Figure 8.2-2 (note that inbox calendar entries are not taken into account in this vCalendar stream). This stream is named <calendar-stream-object-name> and it's format is defined as <calendar-stream-object>.

Since the <calendar-stream-object> is handled as a single vCalendar object, BEGIN:VCALENDAR and END:VCALENDAR delimiters are included only in the beginning and ending of the stream, not in the beginning and ending of each individual event or todo-entry in the stream. Entries with restricted access are listed as empty objects in the stream.

If calendar index support is implemented the organization of the calendar entries in the <calendar-stream-object> must reflect the organization of the single <calendar-indexed-object> entries.

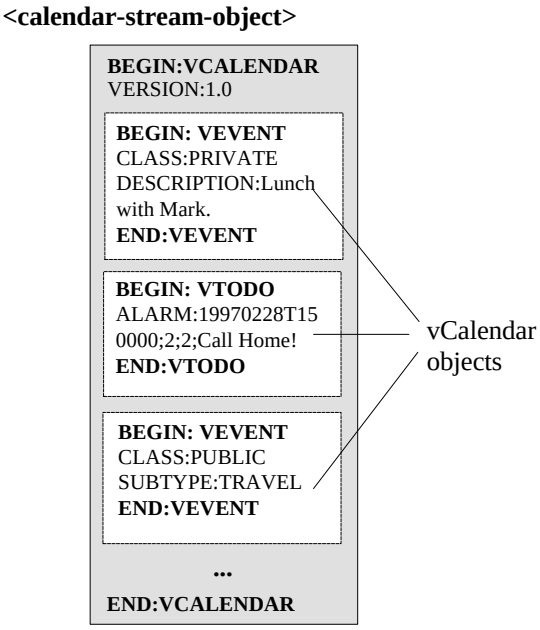


Figure 8.2-2 A stream of all calendar objects

Calendar Access Hierarchy Example: A calendar stream object "telecom/cal.vcf" allows an easy ready/write access to the entire calendar. Support for connectionless push operation is also implemented. In the device inbox there are four objects, which have been received by a remote push.

device inbox: meeting.vcs reminder.vcs call.vcs party.vcs	[inbox for storing the vCalendar objects received by a remote push]
telecom/cal.vcs	[vCalendar stream object listing all the calendar entries]

(3) CALENDAR INDEX SUPPORT

If read/write access to single calendar entries is supported, each vCalendar entry is assigned and individual index and named <calendar-indexed-object-name>. The format of these objects is defined as <calendar-indexed-object>. Entry indexes are expressed as integers starting from 0. The indices do not correspond to any internally used sorting scheme such as the location of the entry in an alphanumerically organized entry list, but are merely used to indicate the virtual location of an entry in the vCalendar list. All vEvent and vTodo objects are listed as individual objects with their own index numbers as shown in the Figure 8.2-3.

The organization of entries in the level two <calendar-stream-object> must reflect the organization of the single <calendar-indexed-object> entries. Accordingly, any changes in any of the calendar index entries must be updated in the stream object. The calendar entry indices may be static or dynamic. If a calendar is based on static indexing, it is up to the implementation whether all empty locations are indicated in the <calendar-stream-object>. This is recommended in order to allow straightforward access to single vCalendar objects based on the information given by a read-all operation. Typically, systems with static indexing will return an empty response for read requests into indices with no content whereas systems with dynamic indexing return response code "not found". As internal organization of the calendar entries may have significance to the application user, no unauthorized changes in the entry order should be made.

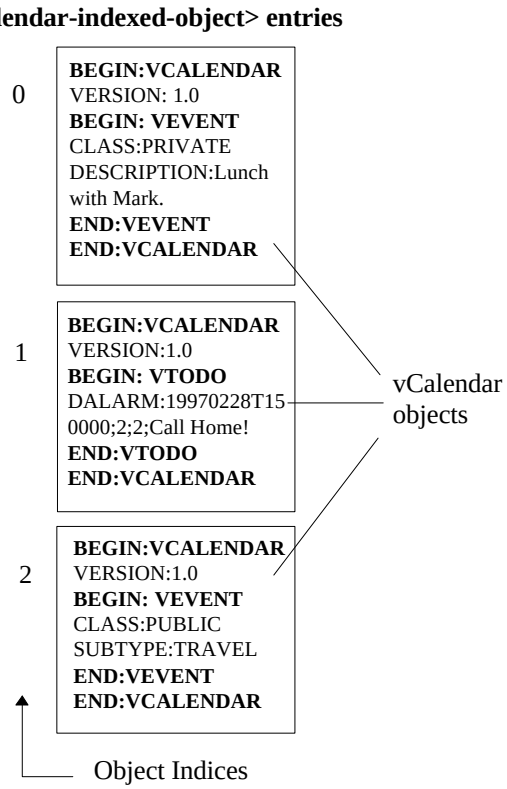


Figure 8.2-3 vCalendar entries with indices

Calendar Index Hierarchy Example: Each object in the calendar is assigned an individual index. Read all/write all -operations are provided by including the "telecom/cal.vcs" object. Device inbox holds one entry which has been received by a remote push.

device inbox: meeting.vcs	[inbox for storing the vCalendar objects received by a remote push]
telecom/cal.vcs	[vCalendar stream object listing all the calendar entries]
telecom/cal/**0**.vcs telecom/cal/**1**.vcs telecom/cal/**2**.vcs telecom/cal/**3**.vcs	[calendar objects with individual indices]

(4) CALENDAR SYNCHRONIZATION SUPPORT

Finally, at the calendar synchronization level, an additional log object named <calendar-modification-log-object-name> is to be implemented. This object, defined as <calendar-modification-log-object>, is to hold information about all changes in the any of the <calendar-indexed-object>.

Calendar Synchronization Hierarchy Example: A calendar log object with information about all changes in the calendar entries in included. Each object in the calendar is assigned an individual index. Read all/write all - operations are provided by including the "telecom/cal.vcs" object. Device inbox holds one entry which has been received by a remote push.

device inbox: meeting.vcs	[inbox for storing the vCalendar objects received by a remote push]
telecom/cal.vcs	[vCalendar stream object listing all the calendar entries]
telecom/cal/0.vcs telecom/cal/1.vcs telecom/cal/2.vcs telecom/cal/3.vcs	[calendar objects with individual indices]
telecom/cal/cal.log	[calendar modification log object]

8.3 Minimum Support

Devices with limited connection-oriented transmission capabilities may push calendar entries to a remote device using OBEX PUT server on top of Ultra. No connection establishment or data acknowledgment is included in this one-way process.

Received calendar entries are recommended to be stored into an inbox of the remote device. No response is expected to the offer. It is, however, recommended that a UI indication is given at the remote device about the received calendar object. New entries should not be saved to the calendar without user intervention.

Also devices with connection support are required to provide calendar object push on top of Ultra. This service is mandatory for all devices with a Telecom Calendar application.

Calendar Entry Push Example:

Local Device	**PUT**	[name - meeting.vcs]
Remote Device		
	device inbox meeting.vcs	
	UI indication "Do you want to save this entry in your calendar? "	

8.4 Calendar Access Support

The calendar access support service relies on the connection-oriented service provided by the IrLAP and IrLMP layers. Devices with calendar access support provide functions for reading, writing and deleting all calendar objects by a single operation. These devices have all the calendar entries listed in a <calendar-stream-object>.

The contents of a <calendar-stream-object> may be deleted by writing an empty object on top of the original, i.e. by applying a PUT operation without a BODY part to the object. If the calendar synchronization support is provided, the <calendar-stream-object> should always be read during a synchronization operation.

OBEX GET and PUT servers are used as explained in the section 4 OBEX Information Access for reading and writing respectively.

Calendar Access Support Example:

Local Device	**GET**	[name - telecom/cal.vcs]
Remote Device	**GET response**	[response code - success body - telecom/cal.vcs]

8.5 Calendar Index Support

The calendar index support relies on the connection-oriented service provided by the IrLAP and IrLMP layers. Devices with this level of the calendar service allow reading, writing, modifying and deleting individual calendar objects. These devices have the <calendar-indexed-object> organized as explained in the 8.2 Calendar Hierarchy.

When modifying a calendar object, the object is first read from the calendar and then edited. The new object is written back to the same index in the calendar. If calendar synchronization support is provided, each modification of a calendar object should be listed in the <calendar-modification-log-object>.

A single object in a calendar may be deleted by making a PUT operation to the index with no body included in the request. Again, if calendar synchronization support level is provided, each calendar object delete should be listed in the <calendar-modification-log-object>.

OBEX GET and PUT servers are used as explained in the 4 OBEX Information Access for reading and writing calendar entries respectively. It should be noted that during an OBEX- connection the indexing of the objects must not be affected by the writing of a new entry.

Calendar Index Support Example: The local device uses the calendar stream object telecom/cal.vcs to check which calendar entry to write to the calendar database of the remote device. As shown in the Figure 8.5-1, both GET and PUT commands are used in this process.

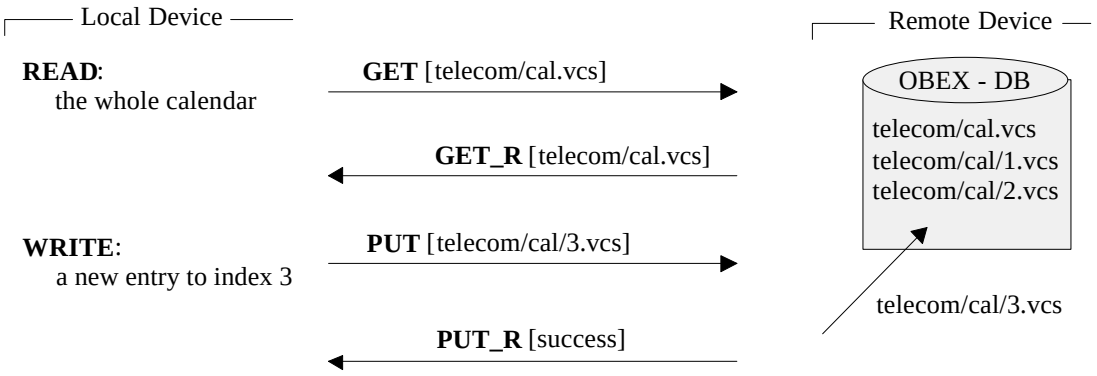


Figure 8.5-1 Calendar read-write -sequence

8.6 Calendar Synchronization Support

When synchronizing calendars of different devices, the information about any changes or additions in the calendar objects is to be stored in a modification log <calendar-modification-log-object>. Starting from the oldest modification, this log lists in chronological order all the changes in the calendar. Some implementations may list a changed entry several times while others only list each entry once even if there have been several modifications to some of the items. It should be noted that the entries marked as ‘changed’ in the synchronization log may be totally new objects which have nothing in common with the original entry.

Calendar synchronization is most effective when there are means to uniquely identify each entry in the calendar. Because of this, devices with true synchronization requirements are recommended to implement the Unique Identifier property field of the vCalendar. This optional vCalendar property, identified by the name UID, enables globally unique representation of the vCalendar entries and allows thus reliable synchronization functionality. For more information about the UID values, please refer to the vCalendar specification.

When synchronization is performed, each of the changed calendar objects listed in the modification log is read. Also the calendar stream object is read. Once synchronization is finished, it is the responsibility of the remote device performing the synchronization to clear the modification log by an OBEX PUT operation of the <calendar-modification-log-object> with a non-existent body.

If there are too many modifications to be included in the log, all changes will be ignored and the log will only list one item, "*", which indicates that the system is unable to list all the changes. This situation may be due to storage space limitations and means that the whole calendar must be synchronized.

8.7 Formal Definition

8.7.1 Minimum Support

<*calendar-minimum-support-object-name*> ::= <*default-char-not-lf*>* "**.vcs**"

<*calendar--minimum-support-object*> ::= <*common-vcalendar*>

8.7.2 Calendar Access Support

<*calendar-stream-object-name*> ::= "**telecom/cal.vcs**"

<*calendar-stream-object*> ::= <*common-vcalendar*>

8.7.3 Calendar Index Support

<*calendar-indexed-object-name*> ::= "**telecom/cal/**" <*digit*>+ ".**vcs**"

<*calendar-indexed-object*> ::= <*common-vcalendar*>

8.7.4 Calendar Synchronization Support

<*calendar-modification-log-object-name*> ::= "**telecom/cal/cal.log**"

<*calendar-modification-log-object*> ::= <*several-modifications-indication*> | <*modified-entry*>*

<*several-modifications-indication*> ::= "*****" <*line-feed*>

<*modified-entry*> ::= <*entry-name*> [<*space*> <*entry-modify-date*>] <*line-feed*>

<*entry-name*> ::= <*calendar-indexed-object-name*> ;'name of the object, i.e., "**2.vcs**".'

<*entry-modify-date*> ::= <*common-date*>

9. Messaging

9.1 Message Object Format

The Telecom messaging application data is organized according to the vMessage format and identified with a VMG -extension. Each vMessage object includes property identifiers which define individual attribute values associated with the vMessage. The property names are defined as explained in section 9.1.1. The property values are expressed as property strings. The unique identifier property, if present, should always be stored.

The formal definition of the vMessage is introduced in section 9.1.2.

9.1.1 Property Identifiers

9.1.1.1 Version

The Version property identifier specifies the highest version number of the vMessage format specification supported by the implementation that created the vMessage object. The value of this property must be 1.0 to correspond to this specification.

VERSION:1.0

9.1.1.2 Encoding, Character Set and Language

Encoding, character set and language parameters, as defined in the vCard specification [VCARD] are indicated in <vmessage-body-property>. Their value take precedence over the encoding, character set and language parameters in the originator and recipient vCards. In case one of these parameters is not present in the vBody, the corresponding parameter listed first in the vMessage object is to take precedence (i.e. the corresponding parameter specified in the originator vCard or the first recipient vCard, should there be no originator vCard).

9.1.2 Formal Definition

An individual vMessage object format is: (In extended BNF)

<*vmessage-object*> ::=
"**BEGIN**:**VMSG**" <*line-feed*>
<*vmessage-property*>*
<*vmessage-originator*>*<*vmessage-envelope*>
"**END**:**VMSG**" <*line-feed*>

<*vmessage-originator*> ::= <*vcard*>

<*vmessage-envelope*> ::=
"**BEGIN**:**VENV**" <*line-feed*>
{ <*vmessage-recipient*>* <*vmessage-envelope*> | <*vmessage-content*> }
"**END**:**VENV**" <*line-feed*>

<*vmessage-recipient*> ::= <*vcard*>

<*vmessage-content*> ::=
"**BEGIN**:**VBODY**" <*line-feed*>

<vmessage-body-property>*

<*vmessage-body-content*>

"**END**:**VBODY**" <*line-feed*>

<*vmessage-body-content*> ::= 'message as specified in RFC822'

<vmessage-body-property> ::= <vmessage-body-encoding-property>|<vmessage-body-charset-property>|<vmessage-body-language-property>

<vmessage-body-encoding-property> ::= ‘encoding, as defined in the vCard specification [VCARD]’

<vmessage-body-charset-property> ::= ‘character set, as defined in the vCard specification [VCARD]’

<vmessage-body-language-property> ::= ‘language, as defined in the vCard specification [VCARD]’

<*vmessage-property*> ::= <*vmessage-version-property*>

<*vmessage-version-property*> ::= <*common-digit*>+ “**.**”<*common-digit*>+ ;‘i.e., version property as specified in [vCard] and in [vCalendar].’

Transparency: Transparency of the vMessage body contents should be provided in the following way:

(i) While sending message, when <*line-feed*> "END:VBODY" sequence is part of the information to be placed in <vmessage-body-content>, then that sequence is replaced with <*line-feed*> "/END:VBODY" sequence.

(ii) While sending message, when <*line-feed*> "/END:VBODY" sequence is part of the information to be placed in <vmessage-body-content>, then that sequence is replaced with <*line-feed*> "//END:VBODY" sequence, and so on.

(iii) While receiving message, when <*line-feed*> "/END:VBODY" sequence is found in the <vmessage-body-content>, then that sequence is replaced with<*line-feed*> "END:VBODY" sequence.

(iv) While receiving message, when <*line-feed*> "//END:VBODY" sequence is found in the <vmessage-body-content>, then that sequence is replaced with <*line-feed*> "/END:VBODY" sequence, and so on.

vMessage Example: Figure 9.1-1 illustrates a typical vMessage object with only one recipient. The message originator information is included in the vCard in the beginning of the message. The originator information is followed by a message envelope that contains all the recipient specific data, i.e. the vCard of the recipient and the contents of the message in yet another envelope. The version numbers of the vMessage and vCard specifications are mandatory in all messages and are added straight after the BEGIN:VMSG and BEGIN:VCARD delimiters.

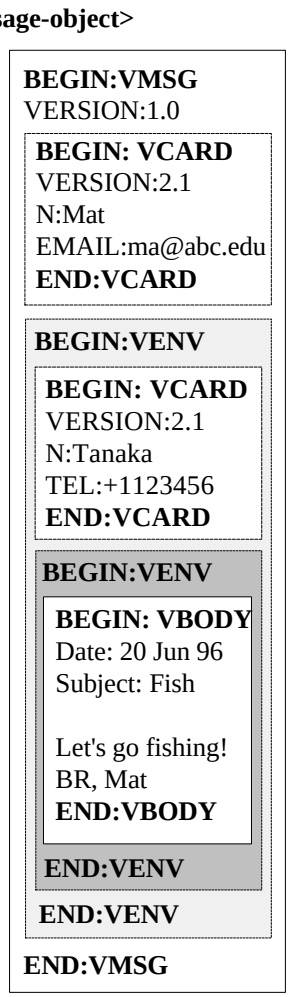


Figure 9.1-1 A vMessage with one recipient

Nested vMessage Example: A vMessage with nested recipients. Note, the tabulation is only added for readability.

```
BEGIN:VMSG                                  // open message
    VERSION:1.0                             // mandatory vMessage
                                               property
    BEGIN:VCARD                                 // vCard of the originator
        VERSION:2.1                         // mandatory vCard property
        N:                                  // empty Name field
    END:VCARD
    BEGIN:VENV                              // open Envelope 1
        BEGIN:VCARD                             // vCard of the middleman
            VERSION:2.1
            N:
            TEL:+44-321-5678
        END:VCARD
        BEGIN:VENV                          // open Envelope 2
            BEGIN:VCARD                         // vCard of the final recipient
                VERSION:2.1
                N:Ann Taylor
                TEL:+44-123-4321
            END:VCARD                       // open Envelope 3
            BEGIN:VENV
                BEGIN:VBODY                     // open Message content
                    Date:  26 Aug 96 1430 EDT
                    From:  friend@city

                    Call me!
                END:VBODY                   // close Message content
            END:VENV                        // close Envelope 3
        END:VENV                            // close Envelope 2
    END:VENV                                // close Envelope 1
END:VMSG                                    // close Message
```

9.2 Message Hierarchy

Figure 9.2-1 shows the four possible levels of messaging support. The way messages are organized and accessed depends on the support level implemented in the device. A device implementing any of the higher support levels also has to provide support of the lower levels. The level of support should be unambiguously identified by the IAS MessageSupport parameter as described in section 12.1.2.3.

In addition to the four service levels, there is one optional messaging object which may be implemented regardless of the level of the message service supported.

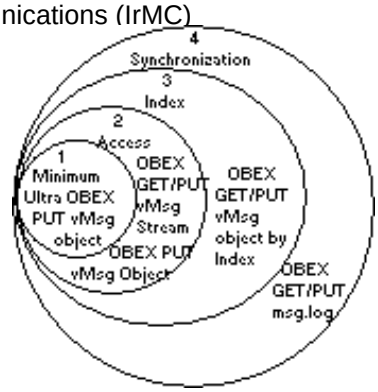


Figure 9.2-1 Message support levels

(1) MINIMUM MESSAGE SUPPORT

The minimum implementation of the message application provides a simple push vMessage -functionality. The name of the pushed object is specified as <message-minimum-support-object-name>, and the object as <message-minimum-support-object>.

At the remote end, received vMessages are recommended to be stored into an inbox with the original name of the object. Yet, it should be noted that the names of the entries may vary between the originator and the receiver due to various system specific naming rules. No automatic actions should be taken based on the name of the received object, e.g. receiving a <message-incoming-stream-object> does not cause automatic update of the incoming message stream.

Minimum Message Hierarchy Example: Three messages received by a remote device push. All the entries are stored in the device inbox with their original names for further processing.

device inbox: [inbox for storing the Messages received by a remote push]
Reminder.vmg
Note.vmg
Congratulations.vmg

(2) MESSAGE ACCESS SUPPORT

Message access support provides a reliable read-all/write-all functionality. Devices with the access support implement two vMessage streams listing all the incoming and outgoing message entries(note that neither the incoming message entries stream nor the outgoing message entries stream includes inbox message entries). These objects are called <message-incoming-stream-object-name> and <message-outgoing-stream-object-name> and defined as <message-incoming-stream-object > and <message-outgoing-stream-object> (Figure 9.2-2).

Messages with restricted access are listed as empty objects in the <message-incoming-stream-object> and <message-outgoing-stream-object>.

If message index support is implemented the organization of the message entries in the <message-incoming-stream-object> and <message-outgoing-stream-object> must reflect the organization of single <message-incoming-indexed-object> and <message-outgoing-indexed-object> entries respectively.

Message Access Hierarchy Example: ”telecom/inmsg.vmg" allows an easy read/write access to all received messages, and "telecom/outmsg.vmg" allows an easy ready/write access to all messages to be sent (please note that once sent, outgoing messages are to be removed from “telecom/outmsg.vmg”) Support for connectionless push operation is also implemented. In the device inbox there are four objects, which have been received by a remote push.

device inbox: [inbox for storing the Messages received by a remote push]
Reminder.vmg
Note.vmg
Congratulations.vmg

telecom/inmsg.vmg [vMessage object listing all the received messages]
telecom/outmsg.vmg [vMessage object listing all the messages waiting to be sent]

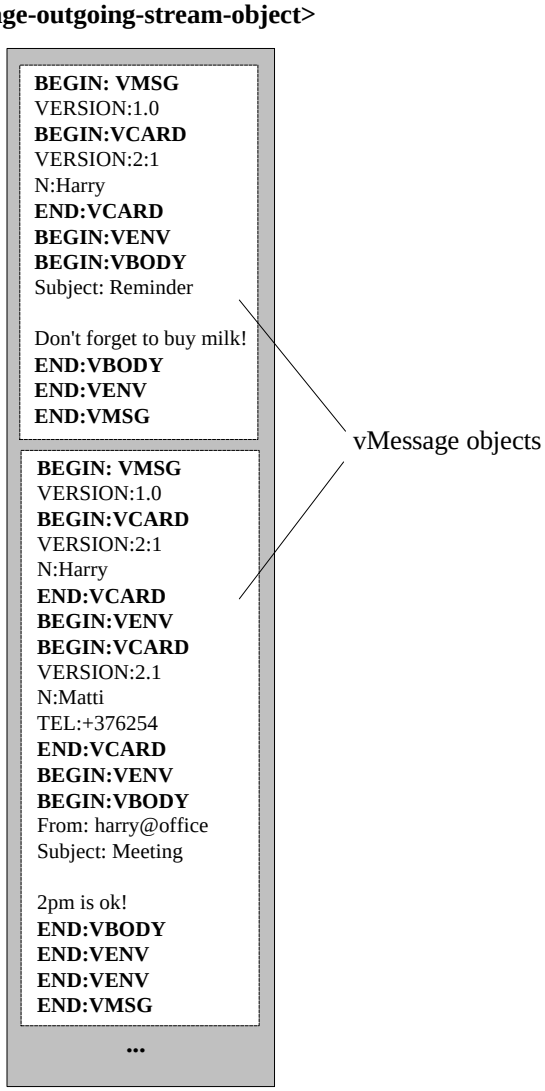


Figure 9.2-2 A stream of outgoing messages

(3) MESSAGE INDEX SUPPORT

Devices with index support provide read/write-access to individual message entries. Messages received from an external source are stored as <message-incoming-indexed-object> with the name <message-incoming-indexed-object-name>. Correspondingly, messages to be sent to an external end point are written and stored as <message-outgoing-indexed-object> with the name <message-outgoing-indexed-object-name>.

All <message-incoming-indexed-object> are stored in dynamically indexed stacks so that the latest received message is always in the location with the index 0 (Figure 9.2-3). The message before the last one is then stored in the location with the index 1 and so on. <message-outgoing-indexed-object> are stored with dynamic indices according to the First-In-First-Out principle. The organization of the message entries in the level three <message-incoming-stream-object> and <message-outgoing-stream-object> must reflect the organization of single <message-incoming-indexed-object> and <message-outgoing-indexed-object> entries respectively.

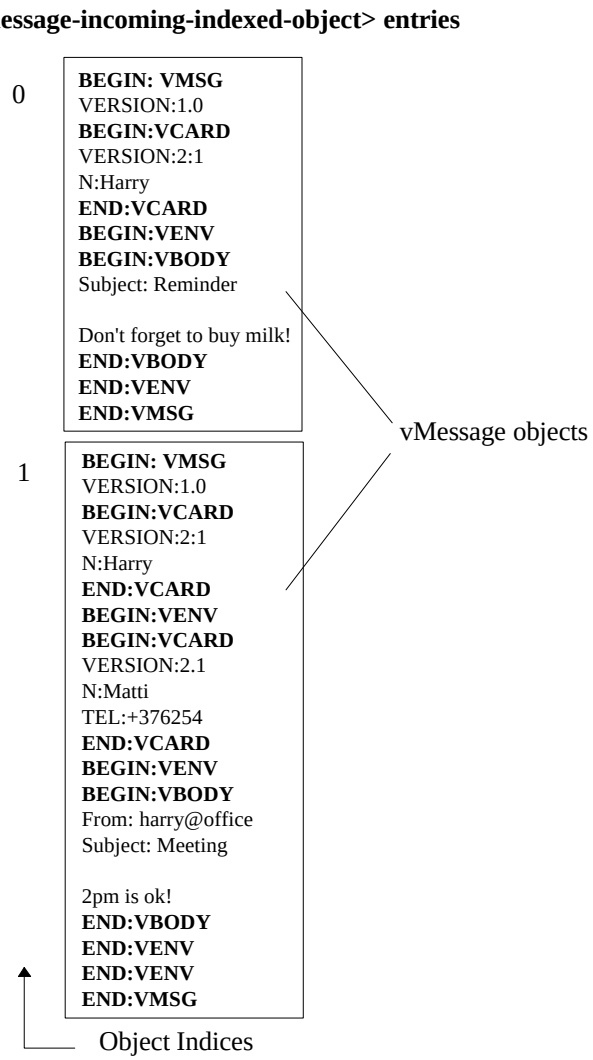


Figure 9.2-3 Received messages with indices

Message Index Example - Incoming Messages: Three messages in the incoming list. A new message is received and stored in the index 1.

telecom/msg/in/**1**.vmg [shiny]
telecom/msg/in/**2**.vmg [happy]
telecom/msg/in/**3**.vmg [people]

-> *new message [hello] received*

telecom/msg/in/**1**.vmg [hello]
telecom/msg/in/**2**.vmg [shiny]
telecom/msg/in/**3**.vmg [happy]
telecom/msg/in/**4**.vmg [people]

Message Index Example - Outgoing Messages: Three messages in the outgoing list. A new message to be sent is created and stored in the list after the previous messages.

telecom/msg/out/**1**.vmg [shiny]
telecom/msg/out/**2**.vmg [happy]
telecom/msg/out/**3**.vmg [people]

-> *new to be sent message [hello] received*

telecom/msg/out/**1**.vmg [shiny]
telecom/msg/out/**2**.vmg [happy]
telecom/msg/out/**3**.vmg [people]
telecom/msg/out/**4**.vmg [hello]

-> *one message is sent (according to the FIFO definition of* <message-outgoing-indexed-object> stacks, *this single message as to be [shiny])*

telecom/msg/out/**1**.vmg [happy]
telecom/msg/out/**2**.vmg [people]
telecom/msg/out/**3**.vmg [hello]

-> *two messages are sent*

telecom/msg/out/**1**.vmg [hello]

(4) MESSAGE SYNCHRONIZATION SUPPORT

Finally, at the message synchronization level, an additional log object named <message-modification-log-object-name> is to be implemented. This object, defined as <message-modification-log-object>, is to hold information about all changes of any <message-indexed-object>.

Message Synchronization Hierarchy Example: A Message log object with information about all changes in the message entries is included. Each vMessage object is assigned an individual index. Read all/write all -operations are provided by including the "telecom/inmsg.vmg" object or the "telecom/outmsg.vmg" object. Device inbox holds one entry which has been received by a remote push.

device inbox:	[inbox for storing the vMessage objects received by a remote push]
telecom/inmsg.vmg	[vMessage stream object listing all the incoming message entries]
telecom/msg/in/0.vmg telecom/msg/in/1.vmg telecom/msg/in/2.vmg telecom/msg/in/3.vmg	[message objects with individual indices]
telecom/msg/vmg.log	[message modification log object for incoming and outgoing messages]

9.3 Minimum Support

Devices with limited connection-oriented transmission capabilities may push messages to a remote device using OBEX PUT server on top of Ultra. No connection establishment or data acknowledgment is included in this one-way process.

It is recommended that the offered entries are stored into an inbox of the remote device. No response is expected to the offer. It is recommended that a UI indication is given at the remote device about the received message. New entries should not be saved to the message directories without user intervention.

Also devices with connection support are required to provide message push on top of Ultra. This service is mandatory for all devices with Telecom Messaging application.

Minimum Support Example: Figure 9.3-1 illustrates the messaging objects and functionality of a minimum support device. Message objects at this service level have random names and are identified as vMessages by the VMG-extension. Device with the minimum support have functionality to push single messages to remote devices as well as receive messages pushed by them. Received messages are recommended to be saved into an inbox for further processing.

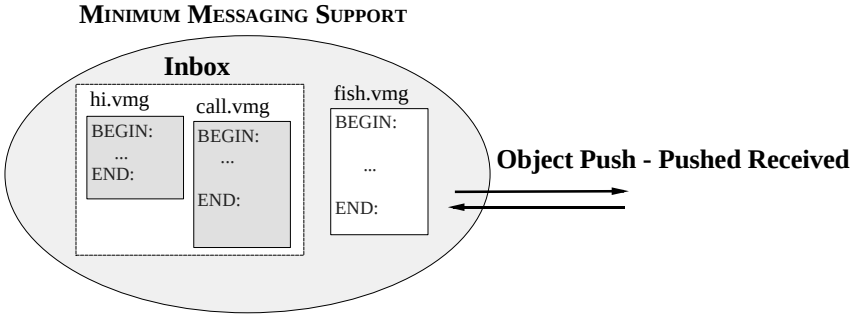


Figure 9.3-1 Objects and functions of the minimum message support

9.4 Message Access Support

This message access support service relies on the connection-oriented service provided by the IrLAP and IrLMP layers. Devices with message access support provide functions for reading, writing and deleting all message

objects by a single operation. These devices have received and sent messages listed in two vMessage streams <message-incoming-stream-object> and <message-outgoing-stream-object> as explained in the section 9.2 Message Hierarchy.

The message stream objects may be deleted by writing an empty entry on top of the original entry, i.e. by applying a PUT operation without a BODY field to the original object.

OBEX GET and PUT servers are used as explained in the section 4 OBEX Information Access for reading and writing message objects.

Access Support Example: Figure 9.4-1 illustrates the messaging objects and functionality of a device with access level support. Additional objects and functionality to the previous level support, i.e. minimum support, are presented in white and bold. Enhanced functionality of the access level devices is provided by including two new objects <message-incoming-stream-object> named "telecom/inmsg.vmg" in the figure and <message-outgoing-stream-object> named "telecom/outmsg.vmg". Unlike the minimum push function access level read and write operations are two-way activities with reliable OBEX transmissions.

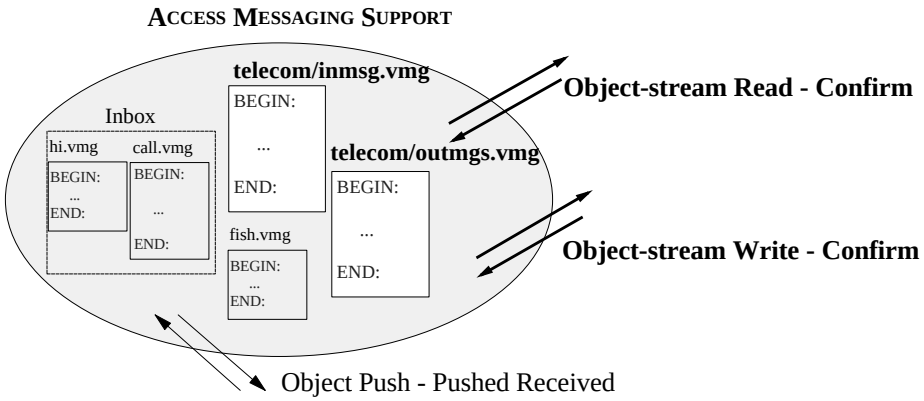


Figure 9.4-1 Objects and functions of the message access support

9.5 Message Index Support

The message index access support service relies on the connection-oriented service provided by the IrLAP and IrLMP layers. Devices with this level of the messaging service allow reading, writing, modifying and deleting of individual message objects. These devices have the messages organized as <message-incoming-indexed-object> and <message-outgoing-indexed-object> as shown in the Figure 9.5-1and explained in the section 9.2 Message Hierarchy.

When modifying a message object, the object is first read from the remote device and edited. The new message is then written back to the same index in the remote device. If message synchronization support is provided, each modification of a message object should be listed in the <message-modification-log-object>.

Single message objects may be deleted by applying a PUT operation to the index without a BODY in the request. Again, if message synchronization support level is provided, each message object delete should be listed in the <message-modification-log-object>.

OBEX GET and PUT servers are used as explained in the section 4. OBEX Information Access for reading and writing messages respectively. It should be noted that during an OBEX- connection the indexing of the objects must not be affected by the writing of a new entry.

Index Support Example: Figure 9.5-1 illustrates the messaging objects and functionality of a device with index level support. In addition to the objects of the lower support levels, i.e. minimum and access support levels, devices with index support hold several received and to be sent vMessages each with an individual index

number. It is thus possible to not only read and write the object streams but also read and write individual messages using the reliable OBEX PUT and GET operations.

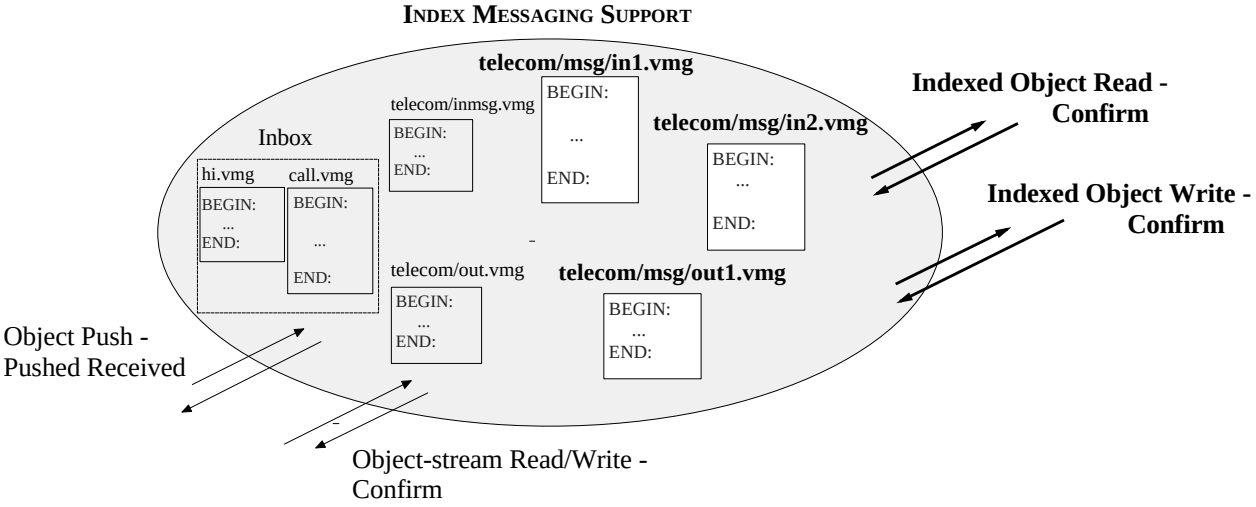


Figure 9.5-1 Objects and functions of the message index support

9.6 Message History Objects

The <message-incoming-stream-object> and <message-outgoing-stream-object> at the access support level and the <message-incoming-indexed-object> and <message-outgoing-indexed-object> at the index support level provide information about the latest messages received and sent. Because of this there is no need for separate incoming and outgoing message history objects. Information about the latest unsuccessful message transmissions can be provided by implementing an optional missed message history object. This object can be read, wrote and deleted in the same way as the level two message stream objects.

9.6.1 Missed Messages

To allow access to the information on the latest missed messages, a <message-missed-history-object> may be included. This object is a stream of vMessage entries, each with an individual index. This indexing is dynamic so that the information of the latest missed message always has the index 1. Similarly, the message missed before the last one has index 2 and so on.

The support of this object is optional. If the object is supported, it should be unambiguously stated by the IAS MessageOptional parameter as described in section 12.1.2.3.

9.7 Message Synchronization Support

When synchronizing messages from different devices, the information about any changes or additions in the message objects is to be stored in a modification log <message-modification-log-object>. Starting from the oldest modification, this log lists in chronological order all the changes for the messages. Some implementations may list a changed entry several times while others only list each entry once even if there have been several modifications to some of the items. It should be noted that the entries marked as ‘changed’ in the synchronization log may be totally new objects which have nothing in common with the original entry.

Message synchronization is most effective when there are means to uniquely identify each message entry. Because of this, devices with true synchronization requirements are recommended to implement the Unique Identifier property field of the vMessage. This optional vMessage property, identified by the name UID, enables globally unique representation of the vMessage entries and thus allows a reliable synchronization functionality.

The Unique Identifier is identified by the property name, UID. This property defines a globally unique identifier associated with the vMessage entity and is used as a mechanism to relate different vMessage entities. The implemenation of the UID will be as described in the vCalendar specification. For more information about the UID values, please refer to the vCalendar specification.

When synchronization is performed, each of the changed message objects listed in the modification log is read. Also the message stream object is read. Once synchronization is finished, it is the responsibility of the remote device performing the synchronization to clear the modification log by an OBEX PUT operation of the <message-modification-log-object> with a non-existent body.

If there are too many modifications to be included in the log, all changes will be ignored and the log will only list one item, "*", which indicates that the system is unable to list all the changes. This situation may be due to storage space limitations and means that the whole message data store must be synchronized.

9.8 Formal Definition

9.8.1 Minimum Support

<*message-minimum-support-object-name*> ::= <*default-char-not-lf*>* "**.vmg**"

<*message-minimum-support-object*> ::= <*common-vmessage*>

9.8.2 Message Access Support

<*message-incoming-stream-object-name*> ::= "**telecom/inmgs.vmg**"

<*message-incoming-stream-object*> ::= <*common-vmessage*>*

<*message-outgoing-stream-object-name*> ::= "**telecom/outmgs.vmg**"

<*message-outgoing-stream-object*> ::= <*common-vmessage*>*

9.8.3 Message Index Support

<*message-incoming-indexed-object-name*> ::= "**telecom/msg/in**" <*digit*>+ ".**vmg**"

<*message-incoming-indexed-object*> ::= <*common-vmessage*>

<*message-outgoing-indexed-object-name*> ::= "**telecom/msg/out**" <*digit*>+ ".**vmg**"

<*message-outgoing-indexed-object*> ::= <*common-vmessage*>

9.8.4 Missed Messages History Object

< *message-missed-history-object-name*> ::= "**telecom/mmh.vmg**"

<*message-missed-history-object*> ::= <*common-vmessage*>*

9.8.5 Message Synchronization Support

<*message-modification-log-object-name*> ::= "**telecom/msg/vmg.log**"

<*message-modification-log-object*> ::= <*several-modifications-indication*> | <*modified-entry*>*

<*message-modifications-indication*> ::= "*****" <*line-feed*>

<*modified-entry*> ::= <*entry-name*> [<*space*> <*entry-modify-date*>] <*line-feed*>

<*entry-name*> ::= <*message-incoming-indexed-object-name*>|<*message-outgoing-indexed-object-name*>
<*entry-modify-date*> ::= <*common-date*>

10. Call Control

In this section the call control commands used in the voice and call control (C.C.) application are described. Control commands are transmitted between handset (so called Mobile Equipment: ME) and car cradle or PC/PDA (so called Terminal Equipment: TE). There are two categories of command sets, common command set , and system oriented command set. The common command set ensures the global usage between ME and TE in which IrDA is installed. The system oriented command set is used between the equipment in the same cellular system.

These commands are based on ITU-T V.25ter and GSM 07.07. Normally AT commands defined in ITU-T V.25ter cannot be accepted after data transmission begins (in on-line data state), but commands specified in this section can be accepted while the voice data is transmitted.

Command sets specified in this section are applicable only to the voice and C.C. application. Other AT command sets can be applied to other applications, e.g. CSD.

10.1 Command Syntax and Character Set

10.1.1.1 Definitions

The following syntactical definitions apply.

<CR> Carriage return character, which value is specified with command S3.

<LF> Linefeed character, which value is specified with command S4.

<...> Name enclosed in angle brackets is a syntactical element. Brackets themselves do not appear in the command line.

[...] Optional subparameter of a command or an optional part of ME information response is enclosed in square brackets. Brackets themselves do not appear in the command line. When subparameter is not given in parameter type commands, new value equals to its previous value. In action type commands, action should be done on the basis of the recommended default setting of the subparameter.

10.1.2 Command Syntax

(1) COMMAND LINE

Figure 10.1-1 shows the basic structure of a command line. Standardized basic commands are found only in V.25ter. Some control commands use syntax rules of extended commands. Every extended command has a test command (trailing '=?') to test the existence of the command and to give information about the type of its subparameters. Parameter type commands also have a read command (trailing '?') to check the current values of subparameters. Action type commands do not store the values of any of their possible subparameters, and therefore do not have a read command.

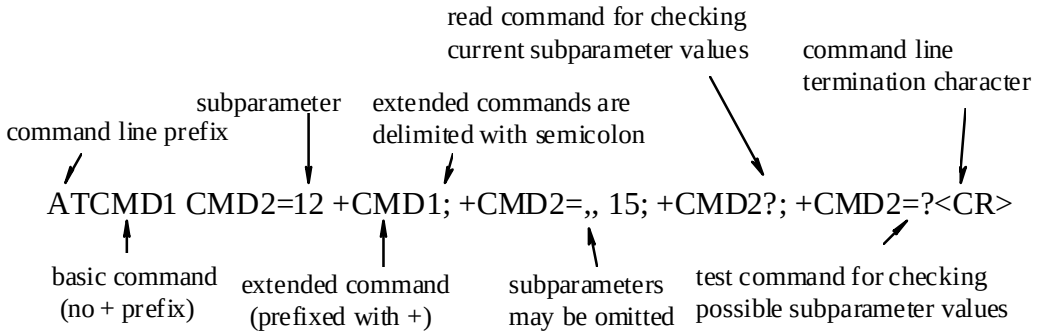


Figure 10.1-1 Basic structure for a command line

If verbose responses are enabled with command V1 and all commands in a command line has been performed successfully, result code <CR><LF>OK<CR><LF> is sent from the ME to the TE. If numeric responses are enabled with command V0, result code 0<CR> is sent instead.

If verbose responses are enabled with command V1 and subparameter values of a command are not accepted by the ME (or command itself is invalid, or command cannot be performed for some reason), result code <CR><LF>ERROR<CR><LF> is sent to the TE and no subsequent commands in the command line are processed. If numeric responses are enabled with command V0, result code 4<CR> is sent instead. ERROR (or 4) response may be replaced by +CME ERROR: <err> (refer clause 10.2.8) when command was not processed due to an error related to ME operation.

(2) INFORMATION RESPONSES AND RESULT CODES

The ME response for the example command line of Figure 10.1-1 could be as shown in Figure 10.1-2. Here, verbose response format is enabled with command V1. If numeric format V0 would have been used, <CR><LF>headers of information responses would have been left out and final result code changed to 0<CR>.

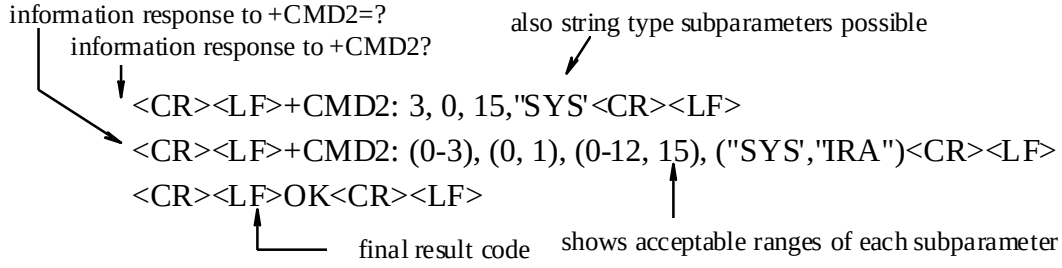


Figure 10.1-2 Response to a command line

So called intermediate result code inform about progress of ME operation (e.g. connection establishment CONNECT), and so called unsolicited result codes indicate occurrence of an event not directly associated with issuance of a command from TE(e.g. ring indication RING)

10.1.3 Character Sets

International reference alphabet (ITU-T T.50 IRA: 7bit code) is used for control commands or responses. When 7 bit code is used, the most significant bit of each word in IrLAP frame shall be set to zero. Unicode (16 bit

code) that is used in alpha entries in phone book commands is hexadecimal coded into IRA characters in AT command interface (e.g. Unicode character string ‘Abc’ is presented as “004100620063” in AT command parameters (A’s hexadecimal value is 0x0041 and so forth).)

10.2 Common Command Set

10.2.1 System / Command Version Identification

(1) SELECT WIRELESS NETWORK

(i) Command: +WS46

+WS46 parameter command syntax

Command	Possible responses
+WS46=<n>	
+WS46?	<n>
+WS46=?	(list of supported <n>s)

(ii) Description

Set command selects the cellular network supported by ME. Read command shows current setting, and test command displays the cellular networks supported by ME.

At the beginning of control commands transmission, TE asks ME about the cellular networks that ME supports. If ME supports more that one cellular system, TE can select the cellular system.

If the network supported by TE coincides with that supported by ME, both common command set and system oriented command set can be used. In the case of inconsistency, only common command set can be used, and commands and responses that are not understood in TE and ME, are ignored.

(iii) Defined Values

<n>: 0 Reserved
1 GSTN
2 Mobitex
3 Data TAC
4 CDPD
5 One-Way Numeric Paging
6 ARDIS
7 AMPS Analog Cellular -Data Mode
8 One-Way Alpha Paging
9 Pinpoint ARRAY
10 Metricom
11 Inmarsat
12 GSM Digital Cellular
13 CDMA Digital Cellular
14 TDMA Digital Cellular
15 Multiple Concurrent WDSs
16 Reserved
17 AMPS Analog Cellular -Voice Mode

18 Wireline Voice Mode

19 PCSI Host Packet Interface

20 Personal Digital Cellular (PDC)

21 N star (Japanese Mobile Satellite Service)

22 W-CDMA

240-255 Experimental/Unknown/Unregistered

Other reserved by PCCA

(iv) Implementation
Mandatory.

(2) COMMAND SET VERSION IDENTIFICATION

(i) Command: +CCVI

+CCVI action command syntax

Command	Possible responses
+CCVI	+CCVI: <com ver>[,<n>,<sys ver>[...]]
+CCVI=?	

(ii) Description
Command shows the command set version supported by ME. For each system supported by ME a system id - system oriented command set version pair is returned.

(iii) Defined values
<com ver>: string type value; format is "Common ver X.X", where "X.X" indicate the version of common command set defined by IrDA Telecom Specification.

<n>: cellular system values as defined under +WS46 command.

<sys ver>: string type value; format is "verX.X", where "X.X" indicate the version of system oriented command set used in cellular system. Value is defined by IrDA Telecom Specification.

(iv) Implementation
Mandatory.

10.2.2 Generic Commands

Table 10.2.2-1 summarizes V.25ter generic ME control commands that can be utilized as common commands. For more information, please refer to ITU-T V.25ter.

Table 10.2.2-1 V.25ter generic ME control commands

Command	Section	Impl.	Contents
Z[<value>]	6.1.1	mand.	ME sets all parameters to their defaults as specified by a user memory profile or by the manufacturer, and resets ME
&F[<value>]	6.1.2	mand.	ME sets all parameters to their defaults as specified by the manufacturer
I[<value>]	6.1.3	opt.	Request manufacturer specific information about the ME (software cannot use this command to determine the capabilities of a ME)
+GMI	6.1.4	mand.	Request ME manufacturer identification
+GMM	6.1.5	mand.	Request ME model identification
+GMR	6.1.6	mand.	Request ME revision identification
+GSN	6.1.7	opt.	Request ME serial number identification

10.2.3 TE-ME Interface Commands

Table 10.2.3-1 summarizes V.25ter TE-ME interface commands relating to command line and response formatting, and TE-ME interface operation. All commands can be utilized as common control commands. For more information, please refer to ITU-T V.25ter.

Table 10.2.3-1 V.25ter TE-ME interface commands

Command	Section	Impl.	Contents
S3=[<value>]	6.2.1	mand.	Command line termination character (mandatory default setting IRA 13)
S4=[<value>]	6.2.2	mand.	Response formatting character (recommended default IRA 10)
S5=[<value>]	6.2.3	mand.	Command line editing character (recommended default IRA 8)
E[<value>]	6.2.4	mand.	Command echo (recommended default 1 i.e. ME echoes commands back)
Q[<value>]	6.2.5	mand.	Result code suppression (recommended default 0 i.e. ME transmits result codes)
V[<value>]	6.2.6	mand.	ME response format (recommended default 1 i.e. verbose format)
X[<value>]	6.2.7	mand.	Defines CONNECT result code format; values manufacturer specific

10.2.4 Call Control Commands and Methods

(1) DIAL

(i) Command: ITU-T V.25ter dial command D

(ii) V.25ter dialing digits
1 2 3 4 5 6 7 8 9 0 * # + A B C D (implementation of these characters is mandatory if corresponding cellular system supports them).

(iii) V.25ter semicolon character ";"
Add semicolon after the last character. Semicolon indicates ’voice call’.

If a comma "," is inserted between dialing digits, all digits before it are treated as phone number information of the originated call, and digits after comma as DTMF tones. Each comma will cause a pause for number of seconds defined by S8 command. If D is directly followed by "," and a voice call is active DTMF tones can be sent similarly, except that the first comma is not treated as a pause.

(iv) Implementation
Mandatory.

(v) GSM additions (refer also GSM 07.07)

; initiate voice call; ME returns to command state immediately or after possible +COLP result code.

I or i override the CLIR supplementary subscription default value for this call; I=invocation (restrict CLI presentation) and i=suppression (allow CLI presentation).

G or g control the CUG supplementary service information for this call; uses index and info values set with command +CCUG.

(2) ANSWER A CALL

(i) Command: ITU-T V.25ter A

(ii) Implementation
Mandatory.

(3) REJECT A INCOMING CALL

(i) Command: ITU-T V.25ter Hook control command H.

(ii) In the case of a second call, a second call should be terminated.

(iii) Implementation
Mandatory.

(4) TERMINATE A CALL

(i) Command: ITU-T Ver.25ter Hook control command H.

(ii) Implementation
Mandatory.

(5) SELECT PHONE BOOK MEMORY STORAGE

(i) Command: +CPBS

+CPBS parameter command syntax

Command	Possible response(s)
+CPBS=<storage>	
+CPBS?	+CPBS: <storage>[,<used>,<total>]
+CPBS=?	+CPBS: (list of supported <storage>s)

(ii) Description
Set command selects phone book memory storage <storage>, which is used by other phone book commands. If setting fails in an ME, +CME ERROR:<err> is returned. Refer section 10.2.8 for <err> values.

Read command returns currently selected memory, and when supported by manufacturer, number of used locations and total number of locations in the memory.

Test command returns supported storages as compound value.

(iii) Defined values
<storage>:

"ME" ME phone book (refer to section 7 Phone Book)

"RC" ME received calls (incoming calls, refer to section 7.8.5.1 Incoming Calls)

"DC" ME dialed calls (outgoing call, refer to section 7.8.5.2 Outgoing Calls)

"MC" ME missed calls (refer to section 7.8.5.3 Missed Calls)

"EN" ME or SIM emergency number

"ON" ME or SIM own number list

"FD" SIM fix dialing-phone book

"LD" SIM last-dialing-phone book

"SM" SIM phone book

"MT" combined ME and SIM phone book

<used>: integer type value indicating the number of used locations in selected memory.

<total>: integer type value indicating the total number of locations in selected memory.

(iv) Implementation
Optional.

(6) DIRECT DIALING FROM PHONE BOOKS

(i) Commands

1. D><str>[;] originate call to phone number which corresponding character field is <str> (if possible, all available memories should be searched for the correct entry). If the format of the memory is specified by VCARD, property values under Property Name 'N' are compared with <str>.

2. D>mem<n>[;] originate call to phone number in memory ‘mem’ entry location <n> (available memories may be those returned by +CPBS=?). If the format of the memory is specified by VCARD, <n> indicates the index of object file.

3. D><n>[;] originate call to phone number in entry location <n> (it is manufacturer specific which memory storage of SIM and ME is used.) If the format of the memory is specified by VCARD, <n> indicates the index of object file.

[NOTE] Semicolon character indicates "voice call".

(ii) Description
ME and SIM can contain phone books which have a phone number and a character field for each phone book entry location. The use of V.25ter dialing command ensures that direct dialing from ME and SIM phone book is possible through ordinary communications software which just gives the phone number field to be filled and then use the D command to originate the call. (If vCard is the format used for phone book entries, then the phone number to be dialed is the one listed first in the selected entry)

(iii) Defined values
mem: same as <storage> in +CPBS (e.g. "D>RC1")

<str>: string type value, which should equal to an alphanumeric field in at least one phone book entry in the searched memories. Used character set should be hexadecimal coded Unicode.

<n>: integer type memory location should be in the range of locations available in the memory used

(iv) Responses
Possible error responses include +CME ERROR: <err> when error is related to ME functionality. Refer section 10.2.8 for possible error values. Otherwise ME responses can have values defined by V.25ter.

(v) Implementation
Optional.

(vi) GSM additions
Also G, i and I dial modifier characters may be present.

(7) ITU-T V.25TER CALL CONTROL COMMANDS

Table 10.2.4-1 summarizes V.25ter call control commands relating to call control function. All commands can be utilized as common control commands. For more information, please refer to ITU-T V.25ter.

Table 10.2.4-1 V.25ter call control commands

Command	Section	Impl.	Contents
S0=[<value>]	6.3.8	mand.	Sets the number of call indications (rings) before automatically answering the call; value equaling zero disables automatic answering and is the default
S8=[<value>]	6.3.11	mand.	Sets number of seconds to wait when comma dial modifier encountered in dial string of D command (default is 2 seconds)

10.2.5 Phone Book Access

(1) READ PHONE BOOK ENTRIES

(i) Command: +CPBR

+CPBR action command syntax

Command	Possible response(s)
+CPBR=<indexl>[,<index2>]	+CPBR: <index1>,<number>,[<type>],<text>[[...] <CR><LF>+CPBR: <index2>,<number>,[<type>],<text>]
+CPBR=?	+CPBR: (list of supported <index>s),<nlength>,<tlength>

(ii) Description
Execution command returns phone book entries in location number range <index1>...<index2> from the current phone book memory storage selected with +CPBS. If <index2> is left out, only location <index1> is returned. Entry fields returned are allocation number <indexn>, phone number stored there <number> (of format <type>) and text <text> associated with the number. If listing fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for <err> values.

Test command returns location range supported by the current storage as a compound value and the maximum lengths of <number> and <test> fields. If ME is not currently reachable, +CME ERROR: <err> is returned. Refer section 10.2.8 for <err> values.

(iii) Defined values
<index1>, <index2>, <index>: integer type in the range of location numbers of phone book memory

<number>: string type phone number of format <type>

<type>: type of address octet in integer format, corresponding bitmap is "1XXXYYYY" where

XXX= 0: unknown

1: international number

2: national number

3: network specific number

4: dedicated access, short code

YYYY= 0: unknown

1: ISDN/telephony numbering plan (Rec. E.164/E.163)

3: data numbering plan (Rec.X.121)

4: telex numbering plan (Rec.F.69)

8: national numbering plan

9: private numbering plan

All other values are reserved.

For more information refer to GSM 04.08 section 10.5.4.7.

When the format of phone book is specified by VCARD, <type>value is not return.

<text> string type field of maximum length <tlength>; character set is hexadecimal coded Unicode

<nlength> integer type value indicating the maximum length of field <number>

<tlength> integer type value indicating the maximum length of field <text> (in Unicode characters)

(iv) Implementation
Optional.

(2) WRITE PHONE BOOK ENTRY

(i) Command: +CPBW

+CPBW action command syntax

Command	Possible response(s)
+CPBW=[<index>][,<number > [,<type>[,<text >]]]	
+CPBW=?	+CPBW: (list of supported <index>s),<nlength>,(list of supported <type>s),<tlength>

(ii) Description
Execution command writes phone book entry in location number <index> in the current phone book memory storage selected with +CPBS. Entry fields written are phone number <number> (in the format <type>) and text <text> associated with the number. If those fields are omitted, phone book entry is deleted. If <index> is left out, but <number> is given, entry is written to the first free location in the phone book (the implementation of this feature is manufacturer specific). If writing fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for <err> values.

Test command returns location range supported by the current storage as a compound value, the maximum length of <number> field, supported number formats of the storage, and the maximum length of <text> field. If ME is not currently reachable, +CME ERROR: <err> is returned. Refer section 10.2.8 for <err> values. If storage does not offer format information, the format list should be empty parenthesis

(iii) Defined values
<index> integer type values in the range of location numbers of phone book memory

<number> string type phone number of format <type>

<type> type of address octet in integer format (refer +CPBR) ; default 145 when dialing string includes international access code character '+', otherwise 129, If the format of phone book is specified by VCARD, <type>value may be ignored.

<text> string type field of maximum length <tlength>; character set is hexadecimal coded Unicode.

<nlength> integer type value indicating the maximum length of field <number>

<tlength> integer type value indicating the maximum length of field <text> (in Unicode characters)

(iv) Implementation
Optional.

10.2.6 Prohibit Voice Data Transmission

ME can contain object files, i.e. phone books, calendars, or messages. While ME is in standby mode, these files can be access using IRDAOBEX. On the other hand, when ME is in talk made, IRDAOBEX cannot be used because IrLAP frames are filled with voice and control data in the case of IrSIR (115.2 kbps). To exchange object files in talk mode, temporary prohibition of voice data transmission is effective.

(1) PROHIBIT VOICE DATA TRANSMISSION

(i) Commands: +CPVT
+CPVT parameter command syntax

Command	Return
+CPVT=<n>	
+CPVT?	+CPVT: <n>
+CPVT=?	+CPVT: (list of supported <n>s)

(ii) Description
Set command prohibit ME transmitting voice data to TE. The command also indicates that voice data transmission from TE to ME is prohibited.

(iii) Defined value
<n> 0 disable
1 enable(Prohibited)

(iv) Implementation
Optional.

10.2.7 Mobile Station Control and Status Commands

(1) SET THE STATUS OF AUDIO APPLICATION

(i) Command: +CAUDIO

+CAUDIO parameter command syntax

Command	Possible response(s)
+CAUDIO=<tone>,<codec_type>, <length_attribute>[,<length>]	
+CAUDIO?	+CAUDIO: <tone>,<codec_type>,<length_attribute>[,<length>]
+CAUDIO =?	+CAUDIO: (list of supported <tone>s),(list of supported <codec_type>s),(list of supported <length_attribute>s)

(ii) Description

Set command sets the status of audio application (tone generation function, codec, and data length of IrLAP frame). Some ME may support the tone (ringing tone, busy tone, ring back tone, etc.) generation function for TE. And some ME may support the codec (RPE-LTP, VSELP, etc.). This command enables or disables the tone generation function in ME, and selects the codec supported in ME.

Test command returns values supported by the ME as compound values.

(iii) Defined value

<tone> integer type value enables or disables the tone generation function in ME. This function can be enabled when <codec_type>=0 (ADPCM is selected).

0 disable (default value)

1 enable

<codec_type> integer type value selects the codec.

0 ADPCM (default value)

1 PCM 64

2 PDC VSELP (Full Rate)

3 PDC PSI-CELP (Half Rate)

4 IS54 VCELP

5 QCELP

6 GSM FR (Full rate based on RPE-LPT 13Kbps)

7 GSM HR (Half rate based on VSELP 5.6 Kbps)

8 GSM EFR (Enhanced full rate based on ACELP 12.2 Kbps)

9 EVRC

10 MPEG Audio

11 Twin VQ

<length_attribute> integer type value select the length of IrLAP frame.

0 length of audio data in a IrLAP frame is fixed (default value)

1 length of audio data in a IrLAP frame is changed for each frame.

<length> integer type value in byte indicates the length of audio data in a IrLAP frame. This value is available when <length attribute>=0.

0-255 length of audio data in a IrLAP frame (default value is 80)

(iv) Implementation
Optional. If this command is not implemented, default value is available in ME.

(2) REQUEST SIGNAL QUALITY

(i) Command: +CSQ

+CSQ action command syntax

Command	Possible response(s)
+CSQ	+CSQ: <rssi>,<ber>
+CSQ=?	+CSQ: (list of supported <rssi>s),(list of supported <ber>s)

(ii) Description
Execution command returns received signal strength indication <rssi> and channel bit error rate <ber> from the ME. If command fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for possible <err> values.

Test command returns values supported by the ME as compound values.

(iii) Defined value

<rssi> 0-99: Receive level is defined for each system.

In GSM	0	-113dBm or less
	1	-111dBm
	2...30	-109...-53dBm
	31	-51dBm or greater
	99	not known or not detected
In PDC	0	less than -5dBu
	1	-5dBu.. -4dBu
	...	
	69	64dBu..-65dBu
	70	more than 65dBu
	99	not known or not detected

<ber> 0-99: Receive level is defined for each system.

In GSM	0	BER<0.2%
	1	0.2% < BER < 0.4%
	2	0.4% < BER < 0.8%
	3	0.8% < BER < 1.6%
	4	1.6% < BER < 3.2%
	5	3.2% < BER < 6.4%
	6	6.4% < BER < 12.8%
	7	12.8% < BER
	99	not known or not detected
In PDC	0	BER>3%
	1	1...3%
	2	0.3...1%
	3	BER<0.3%

99 not known or not detected

(iv) Implementation
Optional.

(3) REQUEST BATTERY CHARGE LEVEL

(i) Command: +CBC

+CBC action command syntax

Command	Possible response(s)
+CBC	+CBC: <bcs>,<bcl>
+CBC=?	+CBC: (list of supported <bcs> s),(list of supported <bcl> s)

(ii) Description
Execution command returns battery connection status <bcs> and battery charge level <bcl> of the ME. If command fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for possible <err> values.

Test command returns values supported by the ME as compound values.

(iii) Defined values
<bcs>:

0 ME is powered by the battery

1 ME has a battery connected, but is not powered by it

2 ME does not have a battery connected

3 Recognized power fault, calls inhibited

<bcl>:

0 battery is exhausted, or ME does not have a battery connected

1...100 battery has 1-100 percent of capacity remaining

(iv) Implementation
Optional.

(4) FACILITY LOCK (INCLUDE LOCK OR UNLOCK DIALING / KEYPAD, IR)

(i) Command: +CLCK

+CLCK parameter command syntax

Command	Possible response(s)
+CLCK=<fac>,<mode>[,<passwd>[,<class>]]	When mode.=2 and command successful: +CLCK: <status>[,<class1> [<CR><LF>+CLCK: <status>,<class2>[...]]
+CLCK=?	+CLCK: (list of supported <fac>s)

(ii) Description
Execute command is used to lock, unlock or interrogate a ME or a network facility <fac>. Password is normally needed to do such actions. If command fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for possible <err> value.

Test command returns facility values supported by the ME as a compound value.

(iii) Defined values
<fac>:

"CS" CNTRL (lock CoNTRoL surface (e.g. phone keyboard))

"AO" BAOC (Barr All Outgoing Calls)

"IRDA" IROBEX

"PS" lock Phone to SIM card (ME asks password when other than current SIM card inserted)

"SC" lock SIM card (SIM asks password in ME power-up and when this lock command issued)

"OI" BOIC (Barr Outgoing International Calls)

"OX" BOIC-exHC (Barr Outgoing International Calls except to Home Country)

"AI" BAIC (Barr All Incoming Calls)

"IR" BIC-Roam (Barr Incoming Calls when Roaming outside the home country)

"NM" barr incoming calls from numbers Not stored to ME memory

"NS" barr incoming calls from numbers Not stored to SIM memory

"NA" barr incoming calls from numbers Not stored to Any memory

"AB" All Barring services

"AG" All outGoing barring services

"AC" All inComing barring services

“PBA” lock PhoneBook access

“RTT” lock Reset of accumulated Talk Time

“RCH” lock Reset of accumulated call CHarge

<mode>

0: unlock

1: lock

2: query status

<status>

0: not active

1: active

<passwd>: string type; shall be the same as password specified for the facility from the ME user interface or with command Change Password +CPWD.

<classx> is a sum of integers each representing a class of information (default 7 equals to all classes):

1 voice

(2: data, 4: fax ;not used in this application)

also all other values below 128 are reserved.

(iv) Implementation
Optional.

(5) CHANGE PASSWORD

(i) Command: +CPWD

+CPWD action command syntax

Command	Possible response(s)
+CPWD=<fac>,<oldpwd>,<newpwd>	
+CPWD=?	+CPWD: list of supported (<fac>,<pwdlength>s)

(ii) Description

Action command sets a new password for the facility lock function defined by command Facility Lock +CLCK. If command fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for possible <err> value.

Test command returns a list of pairs which present the available facilities and the maximum length of their password.

(iii) Defined values

<fac>: refer Facility Lock +CLCK

"P2" SIM PIN2

Additionally "G1" "G2" "G3" "G4" can be used. These parameters indicate the groups of facilities set by manufacturer. ex) Facilities "CS" and "AO" may be set in "G2". In this case the passwords for "CS" and "AO" are changed by one operation whose <fac> is "G2".

<oldpwd>, <newpwd>: string type; <oldpwd> shall be the same as password specified for the ME user interface or with command Change Password +CPWD and <newpwd> is the new password; maximum length of password can be determined with <pwdlength>

<pwdlength>: integer type maximum length of the password for the facility

(iv) Implementation

Optional.

(6) REQUEST THE TALK TIME

(i) Command : +CRQTT

+CRQTT action command syntax

Command	Return
+CRQTT=<oper>	+CRQTT: <time>
+CRQTT=?	

(ii) Description

Execution command returns the last or the accumulated talk time, or reset the accumulated talk time. If command fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for possible <err> value.

(iii) Defined value

<oper>: string type value

"L" Last Call

"A" Accumulated

"R" Accumulated talk time is reset.

<time>: string type value. Format is "h...h: mm; ss"

If <oper> = "R", <time>="00:00:00"

(iv) Implementation
Optional.

(7) REQUEST THE CALL CHARGE

(i) Command: +CRQCC

+CRQCC action command syntax

Command	Possible Response
+CRQCC=<oper>	+CRQCC: <charge>
+CRQCC=?	

(ii) Description
Execution command returns the last or the accumulated call charge or reset the accumulated call charge. If command fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for possible <err> value.

(iii) Defined value
<oper>: string type value

"L" Last Call

"A" Accumulated

"R" Accumulated call charge is reset.

<charge>: string type value defined by manufacturer. (e.g. "$10.65")

If <oper> ="R", <charge> may become "0" (defined by manufacturer)

(iv) Implementation
Optional.

(8) SET DATE AND TIME, REQUEST CURRENT DATE AND TIME

(i) Command

+CCLK parameter command syntax

Command	Possible response(s)
+CCLK=<time>	
+CCLK?	+CCLK: <time>
+CCLK=?	

(ii) Description
Set command sets the real-time clock of the ME. If setting fails in an ME error, +CME ERROR: <err> is returned. If command fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for possible <err> value.

Read command returns the current setting of the clock.

(iii) Defined values
<time>:string type value; format is "yy/MM/dd,hh:mm:SS", where characters indicate year (two last digits), month, day, hour, minutes, seconds. The meaning of value is defined by operators or manufacturers.

Optionally format "yy/MM/dd,hh:mm:SS+-zz" where characters indicate year (two last digits),month, day, hour, minutes, seconds and time zone (indicates the difference, expressed in quarters of an hour, between the local time and GMT; range -47...+48), can be used. E.g. 6th of May 1994, 22:10:00 GMT+2 hours equals to "94/05/06, 22:10:00+08".

(iv) Implementation
Optional.

(9) SET AN ALARM TIME, REQUEST CURRENT ALARM TIME

(i) Command: +CALA

+CALA parameter command syntax

Command	Possible response(s)
+CALA=<time>[,<n>[,<type>[,<text>]]]	
+CALA?	+CALA: <time>,<n>,<type>,<text> [<CR><LF>+CALA: <time>,<n>,<type>,<text> [...]]
+CALA=?	+CALA: (list of supported <n>s),(list of supported <type>s),<tlength>

(ii) Description
Set command sets an alarm time in the ME. There can be an array of different types of alarms, and each alarm may cause different text to be displayed in the ME display. If setting fails in an ME error, +CME ERROR: <err> is returned. Refer section 10.2.8 for <err> values.

Read command returns the list of current alarm settings in the ME.

Test command returns supported array index values, alarm types, and maximum length of the text to be displayed.

(iii) Defined values
<time>: refer +CCLK

<n>: integer type value indicating the index of the alarm; default is manufacturer specific.

<type>: integer type value indicating the type of the alarm (e.g. sound, volume, LED); values and default are manufacturer specific.

<text>: string type value indicating the text to be displayed when alarm time is reached; maximum length <tlength>. Characters are hexadecimal coded Unicode.

<tlength>: integer type value indicating the maximum length of <text>

(iv) Implementation
Optional.

(10) REQUEST VOX

(i) Command: +CRVX

+CRVX parameter command syntax

Command	Return
+CRVX=<vx>	

+CRVX?	+CRVX: <vx>
+CRVX =?	+CRVX: (list of supported <vx>s)

(ii) Description
Set command disables/enables the VOX.

(iii) Defined Value
<vx>: integer type value

0 disable

1 enable

(iv) Implementation
Optional.

(11) CONTROL THE MUTING

(i) Command: +CMUT

+CMUT parameter command syntax

Command	Return
+CMUT=<mu>	
+CMUT?	+CMUT: <mu>
+CMUT=?	+CMUT: (list of supported<mu>s)

(ii) Description
Set command disables/enables voice-muting in uplink.

(iii) Defined value
<mu>:integer type value

0 disable

1 enable(muting)

(iv) Implementation
Optional.

(12) SET THE RINGER SOUND LEVEL OF ME

(i) Command: +CRSL

+CRSL parameter command syntax

Command	Return
+CRSL=<rgli>	
+CRSL?	+CRSL: <rgli>
+CRSL=?	+CRSL: (list of supported <rgli>s)

(ii) Description
Set command selects the ringer sound level of ME.

(iii) Defined Value
<rgli>: integer type value: ringer sound level with manufacturer specific range (smallest value represents the lowest sound level)(Tone level in voice channel of IrDA is not changed by this command)

(iv) Implementation
Optional.

(13) SET THE RINGER SOUND TYPE OF ME

(i) Command: +CSRT
+ CSRT parameter command syntax

Command	Return
+CSRT=<rgty>	
+CSRT?	+CSRT: <rgty>
+CSRT=?	+CSRT: (list of supported <rgty>s)

(ii) Description
Set command selects the ringer sound type of ME.

(iii) Defined Value
<rgty>: integer type value: ringer sound type with manufacturer specific range

(iv) Implementation
Optional.

(14) SET THE VIBRATOR OF ME

(i) Command: +CVIB
+CVIB parameter command syntax

Command	Return
+CVIB=<vb>	
+CVIB?	+CVIB: <vb>
+CVIB=?	+CVIB: (list of supported <vb>s)

(ii) Description
Set command disables/enables the vibrator.

(iii) Defined Value
<vb>: integer type value: state of vibrator

0 disable

1 enable

(iv) Implementation
Optional.

(15) SET THE VOLUME LEVEL OF ME LOUDSPEAKER

(i) Command: +CLVL

+CLVL parameter command syntax

Command	Return
+CLVL=<rvli>	
+CLVL?	+CLVL: <rvli>
+CLVL=?	+CLVL: (list of supported <rvli>s)

(ii) Description
Set command selects the receiver sound level of ME. (Level in voice channel of IrDA is not changed)

(iii) Defined Value
<rvli>: integer type value: receiver sound level with manufacturer specific range (smallest value represents the lowest sound level.

(iv) Implementation
Optional.

(16) SET ALARM MODE

(i) Command: +CALM

+ CALM parameter command syntax

Command	Return
+CALM =<al>	
+CALM?	+CALM: <al>
+CALM =?	+CALM: (list of supported <al>s)

(ii) Description
Set command selects the alarm sounds mode of ME.

(iii) Defined value
<al>: integer type value.

0 normal mode

1 silent mode (all sounds from ME are prevented)

2 normal mode, but ascending ringing volume

(Tone level in voice channel of IrDA is changed by this command, if the tone generation function is supported in ME.)

NOTE: Compare with +CRSL.

3 in case of incoming call beep once (it is manufacturer specific how other alarms are indicated)

4 in case of incoming call ring only once (it is manufacturer specific how other alarms are indicated)

(iv) Implementation
Optional.

(17) SET USER NAME (DEVICE NICKNAME), REQUEST CURRENT USER NAME (DEVICE NICKNAME)

(i) Command: +CDNN

+CDNN parameter command syntax

Command	Return
+CDNN=<name>	
+CDNN?	+CDNN: <name>
+CDNN=?	

(ii) Description
Set command sets the user name (device nickname).

Read command returns the current setting of the user name(device nickname).

(iii) Defined value
<name>:string type value, Device nickname of IrDA.

(iv) Implementation
Optional.

10.2.8 Mobile Equipment Error

10.2.8.1 Report Mobile Equipment Error

(i) Command: +CMEE

+CMEE Parameter command syntax

Command	Possible response(s)
+CMEE=[<n>]	
+CMEE?	+CMEE: <n>
+CMEE=?	+CMEE: (list of supported <n>s)

(ii) Description
Set command disables or enables the use of result code +CME ERROR: <err> as an indication of an error relating to the functionality of the ME. When enabled, ME related errors cause +CME ERROR: <err> final result code instead of the regular ERROR final result code. ERROR is returned normally when error is related to syntax, invalid parameters, or ME functionality.

Test command returns values supported by the ME as a compound value.

(iii) Defined values
<n>: 0 (default value) disable +CME ERROR: <err> result code and use ERROR instead

1 enable +CME ERROR: <err> result code and use numeric <err> values (refer section 10.2.8.2)

2 enable +CME ERROR: <err> result code and use verbose <err> values (refer section 10.2.8.2)

(iv) Implementation
Mandatory for <n> values 0 and 1.

10.2.8.2 Mobile Equipment Error Result Code +CME ERROR

The operation of +CME ERROR: <err> result code is similar to the regular ERROR result code: if +CME ERROR: <err> is the result code for any of the commands in a command line, none of the following commands

in the same command line is executed (neither ERROR nor OK result code shall be returned as a result of a completed command line execution). The format of <err> can be either numeric or verbose. This is set with command +CMEE (refer section 10.2.8.1).

NOTE: ITU-T V.25ter command V does not affect the format of this result code.

<err> values (numeric format followed by verbose format):

0 phone failure

1 no connection to phone

2 phone-adapter link reserved

3 operation not allowed

4 operation not supported

5 PH-SIM PIN required

10 SIM not inserted

11 SIM PIN required

12 SIM PUK required

13 SIM failure

14 SIM busy

15 SIM wrong

16 incorrect password

20 memory full

21 invalid index

22 not found

23 memory failure

24 text string too long

25 invalid characters in text string

26 dial string too long

27 invalid characters in dial string

30 no network service

31 network time-out

32 network not allowed -emergency calls only

100 unknown

also all other values below 256 are reserved.

(iv) Implementation
Mandatory for numeric format codes applicable to implemented command set.

10.2.9 Responses

Two unsolicited result code are defined to control the tone and audio application.

(1) INDICATE "TALK MODE"

(i) Result code: +CTALK: <n>

(ii) Description

This code indicates that ME is in talk mode. It also indicates the start/end timing of audio data transmission between ME and TE.

(iii) Defined value
<n>: 0:OFF (end timing of audio transmission)

1:ON (start timing of audio transmission)

(iv) Implementation
Mandatory.

(2) INDICATE "TONE"

(i) Result code: +CTONE: <n>

(ii) Description
This code requests TE to generate tones.

If TE enables the tone generation function in ME by the command +CAUDIO, the result code also indicates the start/end timing of audio data transmission. In this case, if +CTALK:1 and +CTONE:x (x isn't 0) are indicated, the audio transmission must be continued until both +CTALK:0 and +CTONE:0 occur.

(iii) Defined value
<n>: integer type value

0 off (end timing of audio transmission)

1 DT (start timing of audio transmission)
This tone is generated when hook is off, in general.

2 RGT(start timing of audio transmission)
This tone is generated when the network is calling ME, in general.

3 BT(start timing of audio transmission)
This tone is generated when the other party is already in talk mode, in general.

4 warning tone(start timing of audio transmission)
This tone is generated when ME is left off hook, in general..

5 RBT(start timing of audio transmission)
This tone is generated when the network is calling the other party, in general.

6 tone for originating(start timing of audio transmission)
This tone is generated when ME is transmitting dial numbers, in general.

7 battery alarm(start timing of audio transmission)
This tone is generated when the battery is empty, in general.

9 alarm (start timing of audio transmission)
This tone is generated when ME does not work normally, in general.

10 RGT (for transfer of the incoming call) (start timing of audio transmission)
This tone is generated when the network transfers the incoming call, in general.

11 tone (for holding a call) (start timing of audio transmission)
This tone is generated when a call is held, in general.

12 RGT in talk mode (start timing of audio transmission) (for PDC)

13 Call wiating tone (start timing of audio transmission) (for PDC)

14-20 Reserved

21-80 Optional (operator/manufacturer specified tone.)

(iv) Implementation
Mandatory.

10.3 System Oriented Command Set

10.3.1 Control Commands for GSM

The following GSM 07.07 commands (and result codes related to them) from the Telecom Specification system oriented command set for GSM

1. Select TE character set +CSCS (this must be set to "UCS2" before phone book commands can be accessed using Unicode as described in this specification). Mandatory when phone book commands implemented.
2. Cellular result codes +CRC (to distinguish between data and voice calls). Optional.
3. Network registration +CREG. Optional.
4. Operator selection +COPS, Optional.
5. Calling line id presentation +CLIP. Optional.
6. Calling line id restriction +CLIR. Optional.
7. Closed user group +CCUG. Optional.
8. Call forwarding number and conditions +CCFC. Optional.
9. Call forwarding +CCWA. Optional.
10. Call related supplementary services +CHLD. Optional.
11. Call deflection +CTFR. Optional.
12. Supplementary service notifications +CSSN. Optional.
13. List current calls +CLCC. Optional.
14. Enter PIN +CPIN. Optional.

10.3.2 Control Commands for PDC

10.3.2.1 Control Command

10.3.2.1.1 Call Control Commands

(1) HOLD AN INCOMING CALL

(i) Command: +CHOLD
+CHOLD parameter command syntax

Command	Possible response(s)
+CHOLD=<n>	
+CHOLD?	+CHOLD: <n>
+CHOLD=?	+CHOLD: (list of supported <n>s)

(ii) Description

Set command holds a call or recovers a held call.

(iii) Defined values
<n>: 0 recover held call
1 hold call

(2) TERMINATE A HELD INCOMING CALL

(i) Execute command: ITU-T Ver.25ter Hook control command H.

(3) FORWARD AN INCOMING CALL TO A SPECIFIED NUMBER OR VOICE-MAIL

(i) Command: +CFSV
+CFSV action command syntax

Command	Return
+CFSV=<n>	
+CFSV=?	+ CFSV: (list of supported <n>s)

(ii) Description
Execution command transfers the incoming call to the telephone whose number is registered in the exchanger, or voice-mail.

(iii) Defined value
<n>: 0 Transfer to the telephone whose number is registered in the exchanger.
1 Transfer to voice-mail.

(4) FORWARD A CALL IN TALK MODE

(i) Command: +CFCT
+CFCT action command syntax

Command	Return
+CFCT	
+CFCT=?	

(ii) Description
Execution command transfers the call in talk mode.

(5) REQUEST PARTY CALL

(i) Command: +CRPC
+ CRPC action command syntax

Command	Return
+CRPC=<n>	
+CRPC =?	+CRPC: (list of supported <n>s)

(ii) Description
Execution command requests ME to shift to party call mode. In party call mode two states, i.e. switching state and mixing state, exist. In switching state calls are switched alternately. In mixing state the voices of the two

calls are mixed. The initial state of party call mode must be switching state. After TE requests party call mode (switching state), ME shifts to switching state and can receive a second originating number.

Execution command also changes the state of party call mode.

(iii) Defined value
<n>: 0 request switching state
1 request mixing state

(6) HOOKING

(i) Command: +CHK
+ CHK action command syntax

Command	Possible Response
+CHK	
+CHK=?	

(ii) Description
Execution command answers the second incoming call in call waiting service.

Execution command also switches alternately between the two calls in call waiting mode or party call mode (switching state).

(7) TERMINATE A HELD CALL

(i) Command: +CTHC
+CTHC action command syntax

Command	Possible Response
+CTHC	
+CTHC=?	

(ii) Description
Execution command terminates a held call in switching state of party call mode.

(8) REQUEST AN ORIGINATING NUMBER

(i) Command: +CRON
+CRON action command syntax

Command	Return
+CRON	+CRON: <num>
+CRON=?	

(ii) Description
Execution command causes the ME to return the originating number of incoming call. In the case of call waiting mode, the phone number that is connected is sent to TE.

(iii) Defined value
<num>:string type value. Phone number of originating call.

(9) REQUEST NETWORK INFORMATION

(i) Command: +CRNI

+CRNI action command syntax

Command	Return
+CRNI	+CRNI: <res>,<cou>,<ope >,<net>
+CRNI=?	

(ii) Description

Execution command causes the ME to return the network information.

(iii) Defined value

Refer to Personal Digital Cellular Telecommunication System ARIB standard RCR STD-27 section 4.3.6.3.3.13.

<res>:integer type value:0

<cou>: integer type country number . ex) 1:Japan

<ope >: integer type operator group number.

<net>: integer type intra-group network number (0-15)

(10) REQUEST THE INCOMING SIDE BE NOTIFIED OF THE OUTGOING SUBSCRIBER NUMBER

(i) Command: +CRNS

+ CRNS action command syntax

Command	Return
+CRNS=<n>	
+CRNS=?	+CRNS: (list of supported <n>s)

(ii) Description

Set command causes ME to notify the incoming side of the outgoing subscriber's phone number.

(iii) Defined value

<n>: 0 disable (default)

1 enable (effective for only the next call)

(11) NOTIFY THE INFORMATION OF TE

(i) Command: +CNTE

+ CNTE action command syntax

Command	Return
+CNTE=<ope/man>[,<n>]	
+CNTE=?	

(ii) Description

Set command notifies ME of the information about TE.

(iii) Defined value

<ope/man>:integer type value. Value indicates the operator groups or manufacturer.

<n>:string type value. Information defined by the operator/manufacturer.

(12) REQUEST THE INFORMATION OF ME

(i) Command: +CRME

+ CRME action command syntax

Command	Return
+CRME	+CRME:<ope/man>[,<n>]
+CRME=?	

(ii) Description
Execution command returns the information about ME.

(iii) Defined value
Same as +CNTE

10.3.2.1.2 ME Control and Status Commands

(1) SET ALL STATES IN ME

(i) Command: +CRAM

+CRAM parameter command syntax

Command	Possible response
+CRAM=<vx>,<mu>,<rgli>,<rgty>,<vb>,<rvli>,<al>	
+CRAM?	+CRAM: <vx>,<mu>,<rgli>,<rgty>,<vb>,<rvli>,<al>
+CRAM=?	+CRAM: (list of supported <vx>s),(list of supported <mu>s),(list of supported <rgli>s),(list of supported <rgty>s),(list of supported <vb>s),(list of supported <rvli>s),(list of supported <al>s)

(ii) Description
Set command selects all staus of ME.

(iii) Defined value
<vx>:integer type value: state of VOX

<mu>:integer type value:state of muting

<rgli>:integer type value:ringer sound level

<rgty>:integer type value:ringer sound type

<vb>:integer type value:state of vibrator

<rvli>:integer type value:receiver sound level

<al>:integer type value:state of alarm

(2) REQUEST ME STATUS

(i) Command: +CRMS

+ CRMS action command syntax

Command	Possible response
+CRMS	+CRMS: <ms>
+CRMS=?	

(ii) Description

Execution command causes the ME to return the current status of ME.

(iii) Defined value

<ms>:integer type value.

0	out of service area	1	in service area
2	talk mode	3	originating
4	incoming	5	holding incoming call
6	call waiting	7	party call

(3) REQUEST AUTOMATIC RSSI LEVEL SENDING MODE

(i) Command: +CRIM

+CRIM parameter command syntax

Command	Return
+CRIM=<ro>	
+CRIM?	+CRIM: <ro>
+CRIM=?	+CRIM: (list of supported <ro>s)

(ii) Description

Set command sets ME to send the information about receive level without receiving commands from TE. Unsolicited result code "+CRSSI:<rssi>" is used .

(iii) Defined Value

<ro>:integer type value:state of RSSI level running out mode

0	disable (unsolicited result code can not be sent.)
1	enable (unsolicited result code can be sent.)

(4) REQUEST CAR MOUNT/HANDHELD STATUS

(i) Command: +CRCH

+CRCH action command syntax

Command	Return
+CRCH	+CRCH: <hc>
+CRCH=?	

(ii) Description

Execution command causes the ME to return its car mount/handheld status. If ME is in car mount status, TE can originate or or answer a call by Ir connection. If ME is in handheld status, TE cannot originate or answer a call but monitor the ME behavior.

(iii) Defined value
<hc>:integer type value: car mount/handheld status

0 Handheld

1 Car mount

(5) SET OPERATOR/MANUFACTURER SPECIFIED STATUS (A)

(i) Command: +CSOMA
+ CSOMA parameter command syntax

Command	Return
+ CSOMA =<n>	
+ CSOMA?	+ CSOMA: <n>
+ CSOMA =?	+ CSOMA: (list of supported <n>s)

(ii) Description
Set command selects the operator/manufacturer specified status in ME.

(iii) Defined value
<n>:integer type value: status specified by operator/manufacturer.

(6) SET OPERATOR/MANUFACTURER SPECIFIED STATUS (B)

(i) Command: +CSOMB
+ CSOMB parameter command syntax

Command	Return
+ CSOMB=<n>	
+ CSOMB?	+ CSOMB: <n>
+ CSOMB=?	+ CSOMB: (list of supported <n>s)

(ii) Description
Set command selects the operator/manufacturer specified status in ME.

(iii) Defined value
<n>:integer type value: status specified by operator/manufacturer.

10.3.2.1.3 Data / Facsimile Service

(1) REQUEST NON-VOICE COMMUNICATION SERVICE

(i) Command: +CRNV

+ CRNV action command syntax

Command	Return
+CRNV=<m>,<n>[,<o>]	
+CRNV=?	

(ii) Description

Execution command sets the data/fax transmission mode to ME

(iii) Defined value

<m>:integer type value

0 ON 1 OFF

<n>:integer type value

0 FAX

1 MNP

3 TEL

5 continuous data including(system)

6 continuous data including(display)

7 continuous data including(service)

All others are reserved.

<o>:continuous data.

<n>=5, <o>:integer type value.

1 V24 type

All others are reserved.

<n>=6, <o>: characters (length is 0-15).Used character set should be hexadecimal coded Unicode.

Example:

Before receive XID(CONNECTION)	"ASYNC"
MNP	"MNP 4", "MNP 5" or "MNP 10"
V.42	"V.42" or "v.42bis"
Normal mode	"NORMAL"
Other	"CONNECT"
Communication between high data ADPs	"ASYNC D-D "

<n>=7, <o>:string type value "X1,X2"

X1: 0 AT command, escape sequence is permitted

4 AT command, escape sequence is prohibited.

All others are reserved.

X2: 0 custom

1 MNP4

2 MNP5

3 MNP10
4 V.42
5 V.42bis
6 normal mode

10.3.2.1.4 Roaming Service

(1) SELECT THE ROAMING MODE

(i) Command: +CSRM
+CSRM parameter command syntax

Command	Return
+CSRM=<m>[,<net>]	
+CSRM?	+CSRM: <m>[,<net >]
+CSRM=?	+CSRM: (list of supported <m>s[,<net >s])

(ii) Description
Set command selects the roaming mode. Read command returns the current setting of the roaming mode.

(iii) Defined value
<m>: integer type value:same as value in +CRRI

0 roaming mode 0
1 roaming mode 1 (Network identity in group is designated)
2 roaming mode 2 (Network identity in group is designated)
3 roaming mode 3 (Network identity in group is designated)
4 roaming mode 4
5 roaming mode 5
6 roaming mode 6
7 roaming mode 7
8 roaming mode 1 (Network identity in group is not designated)
9 roaming mode 2 (Network identity in group is not designated)
10 roaming mode 3 (Network identity in group is not designated)

<net>: integer type network identity in group(0-15) :Refer to Personal Digital Cellular Telecommunication System ARIB standard RCR STD-27 section 4.3.6.3.3.13.

(2) REQUEST THE ROAMING STATUS

(i) Command: +CRRS
+CRRS action command syntax

Command	Return
+CRRS	+CRRS: <m>,<n>,<res>,<cou.>,<ope>,<net>
+CRRS=?	

(ii) Description
Execution command causes the ME to return the roaming status.

(iii) Defined value
<m>: integer type value: state of roaming

0 location registration failure (time out etc.)

1 location registration rejection (roaming prohibition)

2 location registration rejection (except roaming prohibition)

3 location registration completion

<n>: discrimination of network

0 home network

1 home group network

2 roaming network 1

3 roaming network 2

<res>: integer type value: 0

<cou.>: integer type country number . ex) 1:Japan

<ope>: integer type operator group number.

<net>: integer type network identity in group(0-15)

<cou.>,<ope>,<net>: Refer to Personal Digital Cellular Telecommunication System ARIB standard RCR STD-27 section 4.3.6.3.3.13.

(3) REQUEST THE ROAMING INFORMATION

(i) Command: +CRRI
+CRRI action command syntax

Command	Return
+CRRI	+CRRI:<cou0>,<ope0>,<cou1>,<ope1>,<cou2>, <ope2>,(list of supported <m>s)
+CRRI=?	

(ii) Description
Execution command causes the ME to return the roaming information.

(iii) Defined value
<cou0>: integer type home country number. ex) 1:Japan

<ope0>: integer type home operator group number.

<cou1>: integer type roaming country number 1.

<ope1>: integer type roaming operator group number 1.

<cou2>: integer type roaming country number 2.

<ope2>: integer type roaming operator group number 2.

<coux>,<opex>: Refer to Personal Digital Cellular Telecommunication System ARIB standard RCR STD-27 section 4.3.6.3.3.13.

<m>: integer type value

0 roaming mode 0:disable the roaming, or returns the home group network.

1 roaming mode 1:stay the network whose group is defined in the home operator group number <ope0>.
(Network identity in group is designated)

2 roaming mode 2: stay the network whose group is defined in the roaming operator group number <ope1>.
(Network identity in group is designated)

3 roaming mode 3: stay the network whose group is defined in the roaming operator group number <ope2>.
(Network identity in group is designated)

4 roaming mode 4: stay the network whose group is defined in the home operator group number <ope0> or the roaming operator group number <ope1>. The home operator group number <ope0> is taken precedence.

5 roaming mode 5: stay the network whose group is defined in the home operator group number <ope0> or the roaming operator group number <ope1>. The home operator group number <ope0> is taken precedence.

6 roaming mode 6: stay the network whose group is defined in the home operator group number <ope0> , the roaming operator group number <ope1> or the roaming operator group number <ope2>. The home operator group number <ope0> is taken precedence over <ope1> and <ope2>. The roaming operator group number <ope1> is taken precedence over <ope2>.

7 roaming mode 7: stay the network whose group is defined in the home operator group number <ope0> , the roaming operator group number <ope1> or the roaming operator group number <ope2>. The home operator group number <ope0> is taken precedence over <ope1> and <ope2>. The roaming operator group number <ope2> is taken precedence over <ope1>.

8 roaming mode 1 (Network identity in group is not designated)

9 roaming mode 2 (Network identity in group is not designated)

10 roaming mode 3 (Network identity in group is not designated)

10.3.2.2 Unsolicited Result Code

10.3.2.2.1 Result Code for Call Control

(1) INDICATE "NETWORK INFORMATION"

(i) Result code:+CNETINF: <res>,<cou.>,<ope>,<net>
This code indicates the network information.

(ii) Defined value
Refer to Personal Digital Cellular Telecommunication System ARIB standard RCR STD-27 section 4.3.6.3.3.13.

<res>:integer type value:0

<cou.>: integer type country number . ex) 1:Japan

<ope>: integer type operator group number.

<net>: integer type network identity in group(0-15)

(2) INDICATE "REASON CODE"

(i) Result code: +CRSN: <n>

(ii) Description
This code indicates the reason for call disconnection.

(iii) Defined value
<n>: integer type value

1	no answer
2	no response (out of area or power off)
3	no service (call is prohibited)
4	refused (reject incoming call)
5	no number found (a missing number)

(3) INDICATE "SECOND RINGING"

(i) Result code: +CRING2

(ii) Description
This code indicates the incoming of second call in call waiting service. It also notifies that a held call exist, when a call that was connected is disconnected in party call service(switching state).

(4) INDICATE "RELEASE THE HELD CALL"

(i) Result code: +CREL

(ii) Description
This code notifies that the held call in party call mode(switching mode) has been released.

(5) INDICATE "DISPLAY INFORMATION"

(i) Result code: +CDSINF: <str>

(ii) Description
This code should be sent when ME receives display information signal from network.

(iii) Defined value
<str>:string type value: Character set is hexadecimal coded Unicode.

(6) INDICATE "STANDBY MODE"

(i) Result code: +CMEST

(ii) Description
This code indicates that TE must be in standby mode.

10.3.2.2.2 Result Code for ME and Network Status

(1) INDICATE "RECEIVE LEVEL"

(i) Result code: +CRSSI:<rssi>

(ii) Description
Refer +CRIM.

(iii) Defined value
<rssi>: same as used in +CSQ

(2) INDICATE "OUT OF SERVICE AREA"

(i) Result code: +COUTAREA

(ii) Description
This code should be sent when ME is out of the service area.

(3) INDICATE "IN SERVICE AREA"

(i) Result code: +CINAREA

(ii) Description
This code should be sent when the ME is in the service area.

(4) INDICATE "RESTRICT ORIGINATING"

(i) Result code: +CRESORG: <n>

(ii) Description
This code indicates that originating is restricted.

(iii) Defined value
<n>: 0:OFF
1:ON

(5) INDICATE "ME RESET"

(i) Result code: +CRESET

(ii) Description
This code should be sent when ME is reset.

(6) INDICATE "STATUS OF BATTERY"

(i) Result code: +CBAT: <n>

(ii) Defined value

<n>: 0:empty

1:recovered

(iii) Description
The code "+CBAT: 0" should be sent when the battery level of ME is less than the minimum threshold level. The code "+CBAT: 1" should be sent to TE when the battery level of ME is more than the minimum threshold level.

(7) INDICATE "ME BREAKDOWN"

(i) Result code: +CBRKDOWN

(ii) Description
This code should be output If ME is broken.

(8) INDICATE "CALL IS HELD"

(i) Result code: +CCHLD:<n>

(ii) Defined value
<n>: 0:OFF

1:ON

(iii) Description
This code should be output If call is held.

(9) INDICATE "PARTY CALL"

(i) Result code: +CPRTY:<n>

(ii) Defined value
<n>: 0:OFF

1:ON

(iii) Description
This code should be output, If a party call is set.

10.3.2.2.3 Result Code for Data Transmission

(1) INDICATE "NON-VOICE COMMUNICATION SERVICE"

(i) Result code: +CSNV: <m>,<n>[,<o>]

(ii) Description
This code indicates the setting of the data/fax transmission mode in ME

(iii) Defined value
<m>: same as parameter in command:+CRNV

<n>: same as parameter in command:+CRNV

<o>:continuous data.

<n>=5, <o>:integer type value "X1,X2"

X1:	0	Low speed data
	1	High speed data
	All other are reserved.	
X2:	1	communication mode
	2	standby mode
	3	other mode
	All others are reserved.	

<n>=6, <n>=7, same as parameter in command:+CRNV

11. Audio

11.1 Audio Transmission Overview

The scope of audio transmission architecture is shown below.

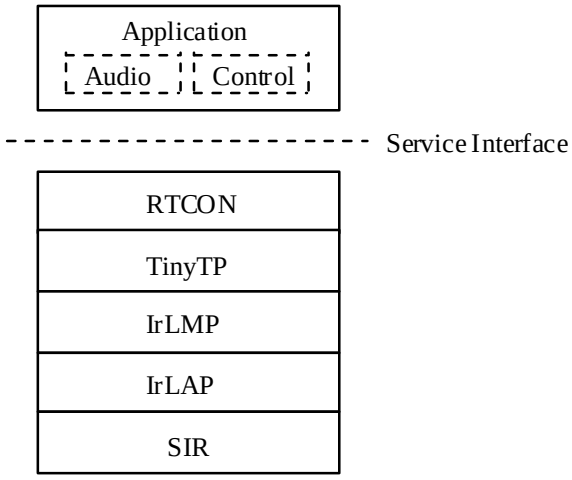


Figure 11.1-1 Audio Transmission Architecture

RTCON stands for Real-time Transfer Control Protocol, and it defines the method of transferring real-time voice data, and control service data, at the same time without jamming. The control service data, is up to 2400 bps. RTCON implementation in a device requires the parameters listed in 11.2.4 to be supported by this device. RTCON should not be built in non-compliant devices.

RTCON adopts the ITU-T G.726 32kbps ADPCM, as the common voice codec method for all devices supported by this specification (portable phones, cradles, PCs, etc.) and for link speeds up to 115.2 kbps (the common voice codec method to be adopted for higher speeds will be specified in the phase two document). In addition, RTCON should allow other speech codec methods. When another speech codec is used, RTCON changes its procedure to the other codec mode, which means pass-through from the radio interface to the IR interface. Therefore, it is necessary to discriminate between 32 kbps ADPCM and another codec. This document specifies the 32 kbps ADPCM transmission method and the way of discrimination between ADPCM and another codec mode. A detailed description of the other modes (Note) is not within the scope of this document.

Note: Other codec modes can include both audio and non-audio data. When RTCON passes through the data from the radio interface to the IR interface, the available data is not only audio data but can also be non-audio data (e.g. Data/Fax service data). RTCON can handle non-audio data as one type of codec data. For example, if it is needed to support IR communication between portable phone and Data/Fax adapter, this method may be useful.

11.2 Transmission Operation

The basic frame exchange for real-time audio data is described as follows.

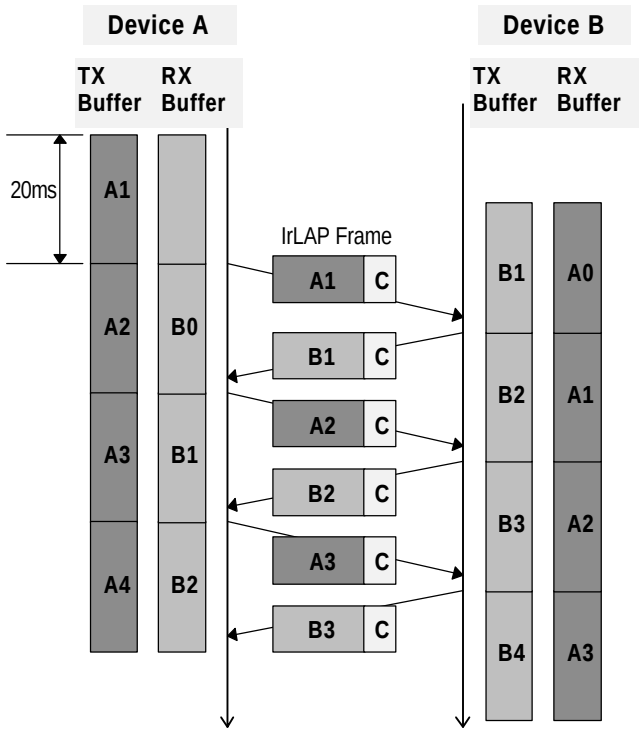


Figure 11.2-1 Normal Transmission Operation

In RTCON, both audio and control data are stuffed into one frame, and transferred to TinyTP.

11.2.1 Real-time Voice Data Management (RTCON operation*)* Overview

Phones have standby and talk modes. RTCON also has these modes. When a phone changes from talk mode to standby mode, RTCON also changes from talk mode to standby mode. RTCON is forced to change its mode by RTCON_status.request (refer to 11.4.7) from the upper application. The upper application issues RTCON_status.request when it detects the Call Control "+CTALK" or "+CTONE" command. RTCON is just a pipe-line for audio and control data, it does not decode or change the data that it transfers. So, RTCON mode is managed by the upper application.

[RTCON standby mode] In this mode, there is only control data in the RTCON link between the phone and the cradle/PC. When the upper application issues control data, RTCON immediately sends a frame, in which the control data is included.

[RTCON talk mode] In this mode, there is both audio and control data in the RTCON link, between the phone and cradle/PC. RTCON sends a frame, in which both audio and control data is included, every 20 ms. When the upper application issues control data, the control data is not sent immediately, but is delayed until the next 20ms slot.

In addition, RTCON should have a procedure for the occurrence of errors as described below.

11.2.2 Factors of Errors

There are three causes of error, first is CRC error detection, the second is clock slip and the third is instability of Indication arrival timing. When one of these errors occurs, there may be an overflow, or underflow of data

to/from the codec circuit. Overflow data should be discarded. In an underflow condition, the codec circuit should be given dummy data, to allow decoding to continue. The detailed RTCON procedure is described in 11.7.

(1) DETECTION OF FCS ERROR

If a CRC error is detected by the IrLAP layer, or if the physical layer SIR does not receive any data, then IrLAP does not have any audio or control data to transfer to RTCON. The audio and control data that is waiting to be sent by RTCON, cannot be transmitted, as its transmission is normally triggered by the reception of data from the other device.

Therefore, the codec circuit cannot be supplied data to decode, and the generated data by the codec circuit, cannot be sent during this error period.

(2) CODEC CLOCK SLIP

As the ADPCM sampling codecs between the primary and secondary stations are not synchronized with each other, clock slip is caused by the difference in the accuracy of their clocks.

- *In the case that the Secondary clock is faster than the Primary.*

The secondary will have more speech data to transmit, than the primary is expecting. Similarly, the secondary will use up the speech data faster than the primary transmits it.

In the transmission part of the secondary station, the secondary will be ready to transmit the data before it receives the indication to transmit from the primary. As the data is ready to transmit before it is transmitted, the delay between sampling the speech and the other end receiving it, is extended.

Eventually this will lead to the situation when there are two send-requests pending, when the indication to transmit is received from the primary. When this happens, the last send-request, might be dropped. Speech data may be lost. It is implementation specific how this error situation is handled.

If the difference between the clocks is small, then this will not happen very often.

In the reception part of the secondary station, the codec will be expecting data faster than the primary is sending it. Eventually the buffer which supplies the codec, will underflow. When this happens, dummy speech data should be sent to the codec. This occurs at the same rate as in the transmission part.

-*In the case that the secondary clock is slower than the primary.*

The primary will have more speech data to transmit, than the secondary is expecting. Similarly, the primary will use up the speech data faster than the secondary transmits it.

In the transmission part of the secondary station, the secondary will not be ready to transmit the speech data when it receives the indication to transmit from the primary. When this happens, the secondary IrLAP has to send an RR frame to the primary station. No speech data is sent.

When the primary sends the next indication to transmit, the speech data that was too late last time, will now be ready. Eventually the error condition will occur again.

If the difference between the clocks is small, then this will not happen very often.

In the reception part of the secondary station, the buffer which supplies the codec will not be empty when the next frame arrives. Eventually the buffer which supplies the codec, will overflow. When this happens, speech data may be lost. It is implementation specific how this error situation is handled.

This occurs at the same rate as in the transmission part.

(3) INSTABILITY OF INDICATION ARRIVAL TIMING

The IrLAP frame size is variable depending on the control data size (0-6 bytes). In addition, if the escape sequence code (C0, C1, 7D) is in the I field of the IrLAP frame, then the IrLAP frame size becomes longer due to

the transparent nature of IrLAP. Therefore, the Indication arrival timing is dispersed (it occurs an Indication arrives earlier than expected and an Indication arrives than expected at random.). This is the similar situation to the clock slip explained above and shortens the clock slip interval.

11.2.3 RTCON Primary / Secondary Role

When RTCON executes a primary procedure, its local IrLAP should act as primary station. Similarly, when RTCON is executing a secondary procedure, its local IrLAP should act as a secondary station.

If RTCON tries to execute a procedure which does not correspond with its local IrLAP role, it will not work. To deal with this case, it is recommended that RTCON has a primary/secondary exchange procedure as described in 11.6.1, in order to correct the IrLAP role.

11.2.4 Negotiation Parameters

- Baud rate	115200 bps (note 1)
- Max turn around time	(=>) 50 ms (note 2)
- Window size	1 (note1)
- Data Size	(=>) 128 byte (note 2)
- Additional BOFs	0
- Minimum turn around time	(<=) 0.5 ms (note 3)

note 1: When RTCON is on talk mode, these values should be set as shown.

note 2: It is possible to set longer values for these parameters, but in that case recovery takes more time when a CRC check error occurs.

note 3: This parameter is defined as the required time delay between the last byte of the last frame sent by a station and the point at which it is ready to receive the first byte of a frame from another station (refer to 6.6.8 in the IrLAP specification). The IrDA device (including CPU) may need a certain time for receipt-frame processing between receiving the frame at the Physical layer and sending an Indication to the upper layer at IrLAP. Therefore, in this document, Minimum turn around time means the period from receiving a frame at the Physical layer to sending a frame at IrLAP (including frame processing time).

The minimum turn around time has been reduced to 0.5ms, to support the timing requirements of full duplex audio. The IrDA frames for audio are 96 bytes long. This includes 80 bytes for the 32kbps ADPCM and 6 bytes of control data. The frame length may be extended with additional bytes added for byte stuffing. On average, this will be one extra byte. At 115200bps, it takes 8.4ms to transmit an average frame (96+1 bytes). To support full duplex speech, one frame must be transmitted and one frame received every 20ms. This leaves on average 1.6ms to turn the link around. The 0.5ms receiver latency allowance in the physical layer rev. 1.2a is required, so that the link would not fail, if more than the average 1 byte of byte stuffing was required.

It should be noted that the IrDA physical layer implementations using receiver latency allowances of greater than 0.5ms will not be able to support the audio feature. That is, 0.5ms is the maximum allowable value for the minimum turn around time.

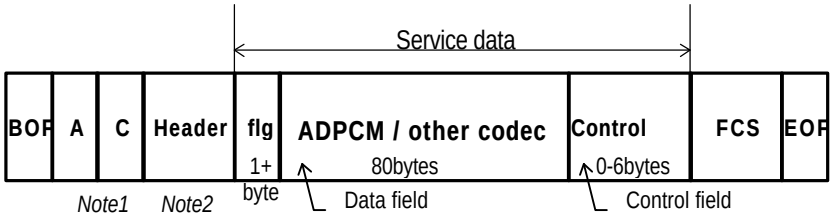
11.3 Frame Format

The IrLAP frame structure is shown as below.

The flag is at the beginning of the service data field. This gives information about the contents of the following data.

The data length of ITU-T G.726 32kbps ADPCM in the service data field is fixed (80 bytes). For other codecs the data field length depends on implementation (max. 80 bytes). The control data length is variable (0-6 bytes). During standby mode, there is only control data following the flag in the service data field.

IrLAP frame structure



Note1: indicates I frame

Note2: includes DLSAP, SLSAP, Credit

Figure 11.3-1

Flag (Flg) field

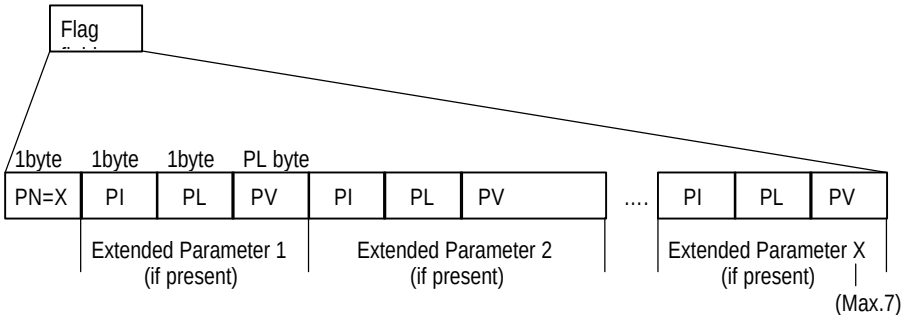


Figure 11.3-2

- 1st byte of Flag field

Bit	Function	Value	Description	Default
7	Data field	0 1	None Present	0
6	0	0	(Fixed)	0 (fixed)
5	Control filed type	0 1	Standard (Call Control data) Other (operator/manufacturer specific)	0
4-3	rsvd			0
2-0	PN	0-7	Number of Extended Parameters	0

- In the case that there is no DATA field in the frame, the Data field is “None”.

- In the case that the data in the Control field is not Call Control, Control field type is ‘Other’.

- If any other detailed information about IrLAP frames is needed, the detailed information can be indicated in Extended Parameters.

- **Extended Parameters**

DATA/Control field extended Common attribute

PI	PI name	PL	PV data type	PV description
0x00	Audio field length	1	0-255 byte	length
0x01 - 0x0F	rsvd			

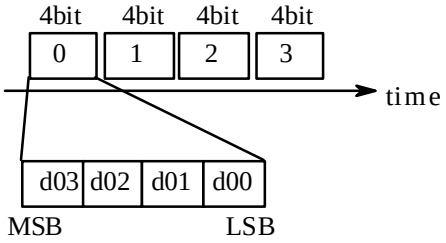
DATA/Control field Operator/Manufacturer specific attribute

PI	PI name	PL	PV data type	PV description
0x10 ~ 0xAF	(Operator category) 0x10=Japan	2+	(Operator code)	undefined
0xF0 ~ 0xFF	Manufacturer specific			undefined

NOTE: No octet can be set with a control sequence code such as C0, C1 and 7D.

ADPCM Bit Ordering

ADPCM sample



Bit Odering in RTCON

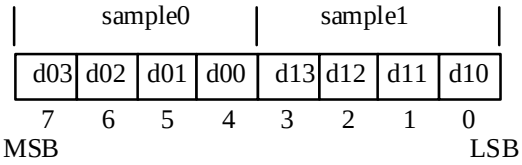
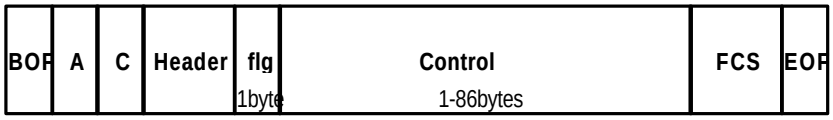


Figure 11.3-3

An ADPCM sample (4 bit) is generated in the ADPCM codec, and a series of two samples composes one byte. In one-byte data, the first sample is the upper 4 bits and the following sample is the lower 4 bits.

There are 5 patterns of service data field dependent on portable phone mode.

(1) STANDBY MODE

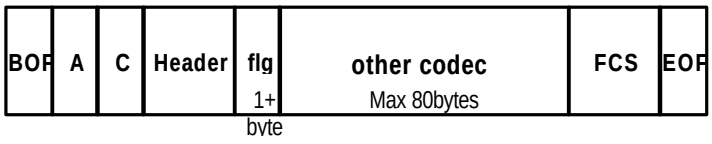


- Control Field: 1 ~ 86 bytes (variable)

(2) DURING TALK MODE, WITH ADPCM, WITHOUT CONTROL DATA

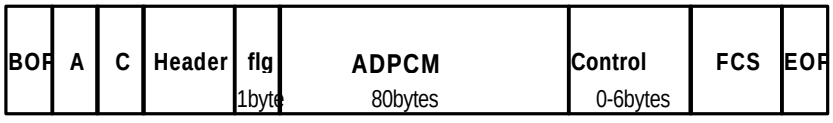
- DATA Field: 80 bytes (Fixed), 32 kbps ADPCM

(3) DURING TALK MODE, WITH AN OTHER CODEC, WITHOUT CONTROL DATA



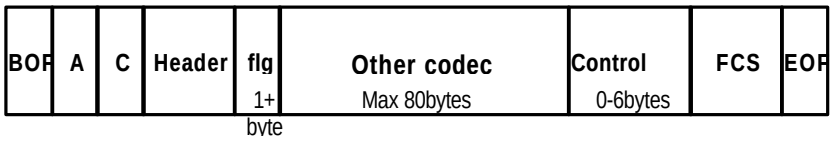
- DATA Field: maximum 80 bytes (dependent on codec or implementation)

(4) DURING TALK MODE, WITH ADPCM, WITH CONTROL DATA



- DATA Field: 80 bytes (fixed), 32k ADPCM
- Control Field: 0~6 bytes (variable)

(5) DURING TALK MODE, WITH AN OTHER CODEC, WITH CONTROL DATA



- DATA Field: maximum 80 bytes (dependent on codec or implementation)
- Control Field: 0-6 bytes (variable)

11.4 Service Interface Definition

11.4.1 Connect Service

RTCON_Connect.request(CalledLsap,Qos)

RTCON_Connect.indication(CallingLsap,Resultant Qos)

RTCON_Connect.response()

RTCON_Connect.confirm(Resultant Qos)

Description: The Connect services are used to establish an RTCON connection with a peer device.

Parameters:

CallingLsap, CalledLsap = TTPSAP address (LSAP address)

Qos = Quality of service parameters for IrLAP link

11.4.2 Disconnect Service

RTCON_Disconnect.request()

RTCON_Disconnect.indication()

Description: The Disconnect service is used to end the connection with RTCON.

11.4.3 Control Service

RTCON_Control.request(ControlData, [Type], [ExtendedParameter])

RTCON_Control.indication(Control, [Type], [ExtendedParameter])

Description: The Control service is used to transmit control data. The default type of control data is standard control.

Parameters:

Type = standard (Call Control, refer to section 10) | other

(Default Type is standard.)

ExtendedParameter corresponds to ‘Extended Parameter’ (refer to 11.3).

11.4.4 Audio Service

RTCON_Audio.request(AudioData, [ExtendedParameter])

RTCON_Audio.indication(AudioData, [ExtendedParameter])

Description: The Audio service is used to transmit audio data. This service is available when RTCON is in TALK mode.

Parameters:

ExtendedParameter corresponds to ‘Extended Parameters’ (refer to 11.3).

11.4.5 Non Audio (Non Voice) Service

RTCON_NonAudio.request(NonAudioData, [ExtendedParameter])

RTCON_NonAudio.indication((NonAudioData, [ExtendedParameter])

Description: The Non-Audio service is used to transmit non-audio data. This service is available when RTCON is in TALK mode.

Parameters:

ExtendedParameter corresponds to ‘Extended Parameters’ (refer to 11.3).

11.4.6 AudioMode Service

RTCON_AudioMode.request(Tone, CodecType, LengthAttribute, [Length])

RTCON_AudioMode.confirm(status)

Description: Default AudioMode is ADPCM. If the application transmits an other type of audio data, it should change RTCON audio mode during RTCON standby mode.

Parameters:

Tone = support | non support

CodecType = ADPCM | PCM64 |PDC VSELP (Full Rate) | PDC PSI-CELP (Half Rate) | IS54 VCELP | QCELP |GSM FR (Full rate based on RPE-LPT 13 Kbps) | GSM HR (Half rate based on VSELP 5.6 Kbps) | GSM EFR (Enhanced full rate based on ACELP 12.2 Kbps) | EVRC | MPEG Audio | Twin VQ | non-Audio | other

LengthAttribute = static | dynamic

Length (option) = RTCON data field length of IrLAP frame status

Status = complete | failure (Note)

Note:“complete” means RTCON is ready to work the requested codec procedure. “failure” means RTCON could not change its mode for the requested codec mode or RTCON doesn’t support the requested codec mode.

11.4.7 Status Service

RTCON_Status.request(RequestState)

RTCON_Status.indication(CurrentState)

Description: The request primitive is used to change RTCON status between Talk and Standby. The indication primitive is used to indicate current RTCON status.

Parameters:

RequestState = talk | standby

11.5 State Chart

11.5.1 Primary State

11.5.1.1 State Chart

State	Event	Action	Next State
IDLE	RTCON_Connect.Request	TTP_Connect.Request	SETUP
	TTP_Connect.Indication(CallingTTPSAP)	PeerSAP = TTPSAP RTCON_Connect.Indication	CONNECT
SETUP	TTP_Connect.Confirm	Codec = ADPCM /* Default */ RTCON_Connect.Confirm	STANDBY(P)
CONNECT	RTCON_Connect.Response	TTP_Connect.Response IrLAP_Primary.Request RTCON_Status.Indication(RoleExchange)	ROLE EXCHANGE (Note)
		Codec = ADPCM /* Default */	STANDBY(S)
	RTCON_Disconnect.Request	TTP_Disconnect.Response	IDLE
ROLE EXCHANGE	IrLAP_Primary.Confirm	Codec = ADPCM /* Default */ RTCON_Connect.Confirm	STANDBY(P)
	IrLAP_Primary.Confirm(deny)	TTP_Disconnect.Request TTP_Connect.Request(PeerSAP) RTCON_Status.Indication(ReConnect)	RECONNECT
RECONNECT	TTP_Connect.Confirm	Codec = ADPCM /* Default */ RTCON_Connect.Confirm	STANDBY(P)
STANDBY(P)	RTCON_Control.Request(ControlData)	TTP_Data.Request(ControlData)	STANDBY(P)
	TTP_Data.Indication(ControlData)	RTCON_Control.Indication(ControlData)	STANDBY(P)
	RTCON_Audio.Request	Empty /* Not Talk State */	STANDBY(P)
	TTP_Data.Indication(AudioData)	Empty /* Not Talk State */	STANDBY(P)
	RTCON_AudioMode.Request(Tone, CodecType, LengthAtr, [Length])	Codec = CodecType /* Apply Parameters */ RTCON_AudioMode.confirm(complete)	STANDBY(P)
	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE
	RTCON_Status.Request(talk)	Initialize all FIFO and Buffer	TALK(Codec) XMIT /* Primary Station */
STANDBY(S)	RTCON_Control.Request(ControlData)	TTP_Data.Request(ControlData)	STANDBY(S)
	TTP_Data.Indication(ControlData)	RTCON_Control.Indication(ControlData)	STANDBY(S)
	RTCON_Audio.Request	Empty /* Not Talk State */	STANDBY(S)
	TTP_Data.Indication(AudioData)	Empty /* Not Talk State */	STANDBY(S)
	RTCON_AudioMode.Request(Tone, CodecType, LengthAtr, [Length])	Codec = CodecType /* Apply Parameters */ RTCON_AudioMode.confirm(complete)	STANDBY(S)
	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE

	RTCON_Status.Request(talk)	Initialize all FIFO and Buffer	TALK(Codec) XMIT /* Secondary Station */

Note: If the primary/secondary exchange procedure (refer to 11.6.1) is implemented, the Next State after receiving RTCON.Connect.Response Event is ROLE EXCHANGE, otherwise, STANDBY.

11.5.1.2 State Definitions

IDLE

This is the initial state. No link is established.

SETUP

The RTCON user (application) has requested an RTCON connection and is waiting for confirmation from the peer.

CONNECT

RTCON has received an incoming connection request and is waiting for the response from the RTCON user (application).

ROLE EXCHANGE

RTCON has made an IrLAP primary request and is waiting for confirmation from IrLAP.

RECONNECT

RTCON is attempting to reconnect the IR link and is waiting for confirmation from the peer.

STANDBY

RTCON connection has been established. Control data can be sent and received.

TALK (TALK XMIT/RECV)

Control data, Audio data and Non-Audio data can be sent and received.

11.5.2 Secondary State

11.5.2.1 State Chart

State	Event	Action	Next State
IDLE	RTCON_Connect.Request	TTP_Connect.Request	SETUP
	TTP_Connect.Indication	RTCON_Connect.Indication	CONNECT
CONNECT	RTCON_Connect.Response	TTP_Connect.Response Codec = ADPCM /* Default */	STANDBY(S)
SETUP	TTP_Connect.Confirm	Start RoleExchangeWaitTimer	ROLE EXCHANGE WAIT
	TTP_Disconnect.Indication	RTCON_Disconnect.Indication	IDLE
ROLE EXCHANGE WAIT	IrLAP_Primary.Indication	IrLAP_Primary.Response Codec = ADPCM /* Default */ RTCON_Connect.Confirm	STANDBY(S)
	TTP_Disconnect.Indication	Start ReconnectWaitTimer	RECONNECT WAIT
	RoleExchangeWaitTimer expired	Codec = ADPCM /* Default */ RTCON_Connect.Confirm	STANDBY(P)
RECONNECT WAIT	TTP_Connect.Indication	TTP_Connect.Response Codec = ADPCM /* Default */ RTCON_Connect.Confirm	STANDBY(S)
	ReconnectWaitTimer expired	RTCON_Disconnect.Indication	IDLE
STANDBY(S)	RTCON_Control.Request(ControlData)	TTP_Data.Request(ControlData)	STANDBY(S)
	TTP_Data.Indication(ControlData)	RTCON_Control.Indication(ControlData)	STANDBY(S)
	RTCON_Audio.Request	Empty /* Not Talk State */	STANDBY(S)
	TTP_Data.Indication(AudioData)	Empty /* Not Talk State */	STANDBY(S)
	RTCON_AudioMode.Request(Tone, CodecType, LengthAtr, [Length])	Codec = CodecType /* Apply Parameters */ RTCON_AudioMode.comfirm(complete)	STANDBY(S)
	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE
	RTCON_Status.Request(talk)	Initialize all FIFO and Buffer	TALK(Codec) /* Secondary Station */
STANDBY(P)	RTCON_Control.Request(ControlData)	TTP_Data.Request(ControlData)	STANDBY(P)
	TTP_Data.Indication(ControlData)	RTCON_Control.Indication(ControlData)	STANDBY(P)
	RTCON_Audio.Request	Empty /* Not Talk State */	STANDBY(P)
	TTP_Data.Indication(AudioData)	Empty /* Not Talk State */	STANDBY(P)

	RTCON_AudioMode.Request(Tone, CodecType, LengthAtr, [Length])	Codec = CodecType /* Apply Parameters */ RTCON_AudioMode.comfirm(complete)	STANDBY(P)
	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE
	RTCON_Status.Request(talk)	Initialize all FIFO and Buffer	TALK(Codec) /* Primary Station */

11.5.2.2 State Definitions

IDLE

This is the initial state. No link is established.

SETUP

The RTCON user (application) has requested an RTCON connection and is waiting for confirmation from the peer.

CONNECT

RTCON has received an incoming connection request and is waiting for the response from the RTCON user (application).

ROLE EXCHANGE WAIT

RTCON is waiting for the request to exchange primary/secondary roles from the peer (portable phone).

RECONNECT WAIT

RTCON is waiting for the request to reconnect the IR link from the peer.

STANDBY

RTCON connection has been established. Control data can be sent and received.

TALK

Control data, Audio data and Non-Audio data can be sent and received.

11.6 Service Sequence Example

11.6.1 Primary / Secondary Exchange

Below sequence shows the cases that Portable phone is a primary station and Hand-set is a secondary station.

Case1: Hand-set supports IrLAP RoleExchange service.

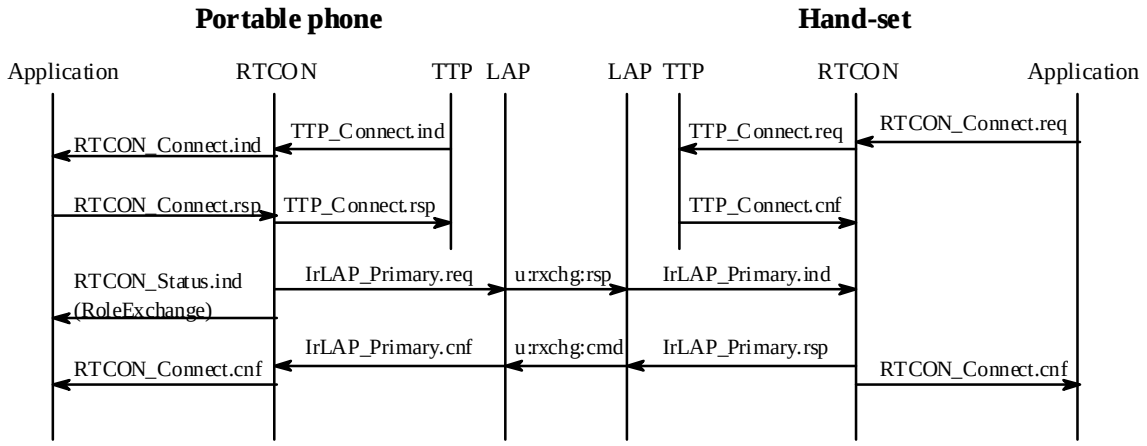


Figure 11.6-1

Case2: Hand-set does not support IrLAP RoleExchange service.

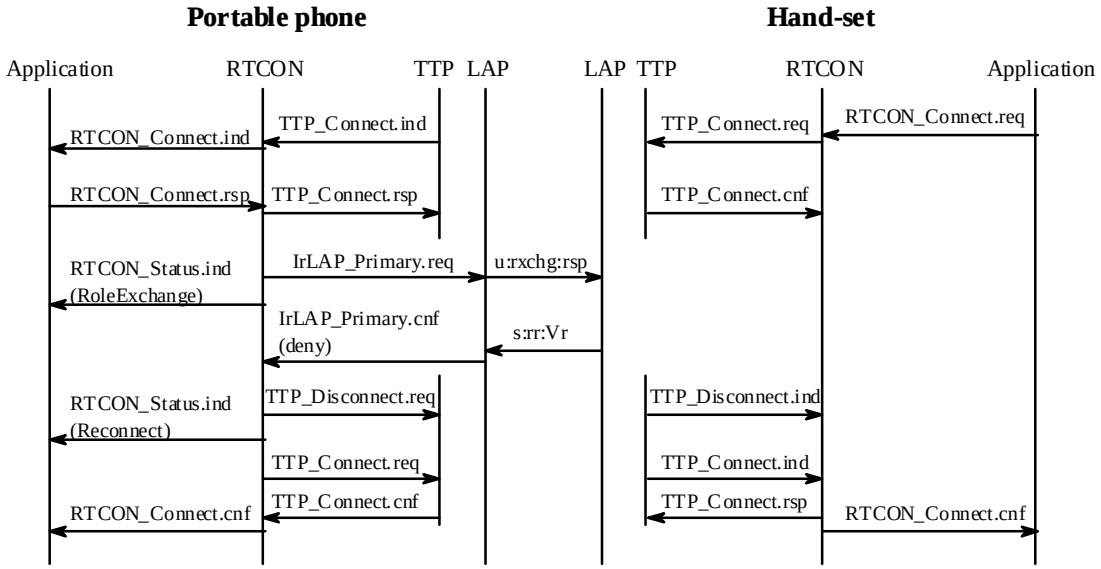


Figure 11.6-2

When the cradle/PC initiates IrLAP connection with portable phone and the portable phone acts as secondary station, the portable phone should exchange the roles of primary and secondary station. First, it tries to exchange primary/secondary roles, which is an option procedure of IrLAP (**case 1**), then if this fails, the portable phone shuts down the IrLAP connection and re-starts it up immediately (**case 2**).

11.6.2 State Change from Standby to Talk

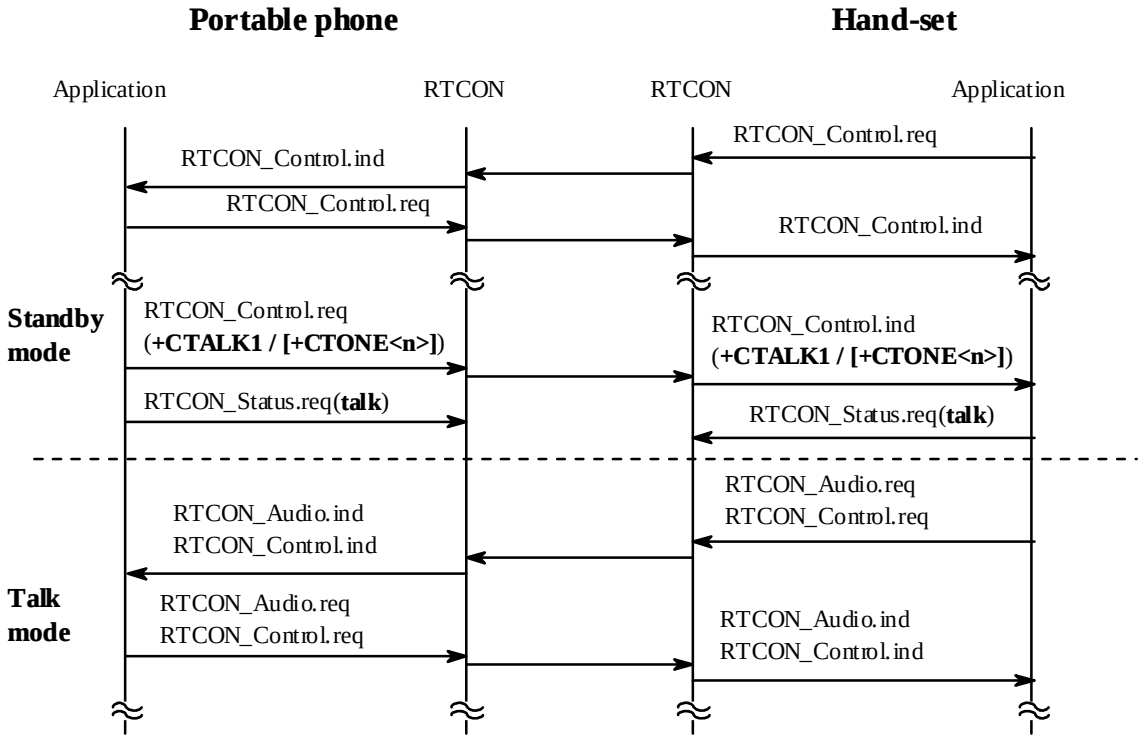


Figure 11.6-3

RTCON_Status.req is issued when either of the following responses in Call Control is detected by the application.

+CTALK1 for the case that codec type is other than 32 kbps ADPCM

+CTONE<n> for the case that codec type is 32 kbps ADPCM and the portable phone supports a tone generating function
n is not equal to 0

11.6.3 State Change from Talk to Standby

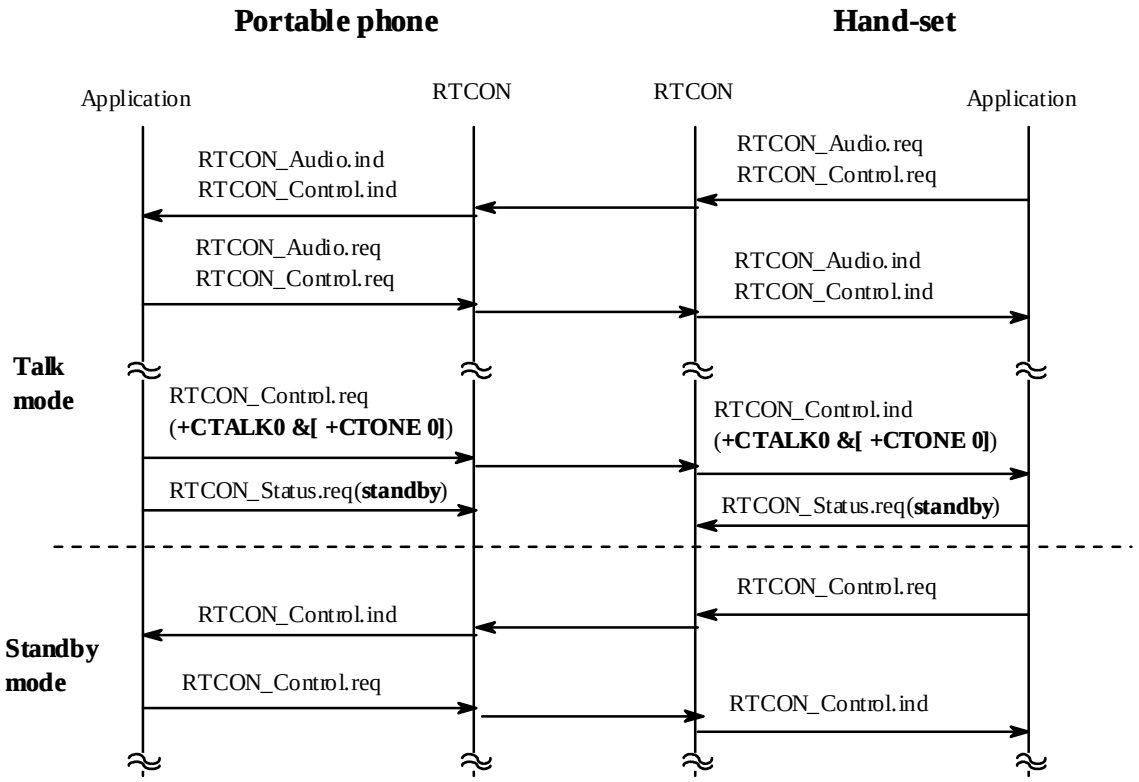


Figure 11.6-4

If the codec type is 32 kbps ADPCM and the portable phone supports a tone generating function, RTCON_Status.req is issued when both '**+CTALK 0**' and '**+CTONE 0**' are detected by application.

Otherwise, RTCON_Status.req is issued when '**+CTALK 0**' is detected.

11.6.4 Codec Mode Change

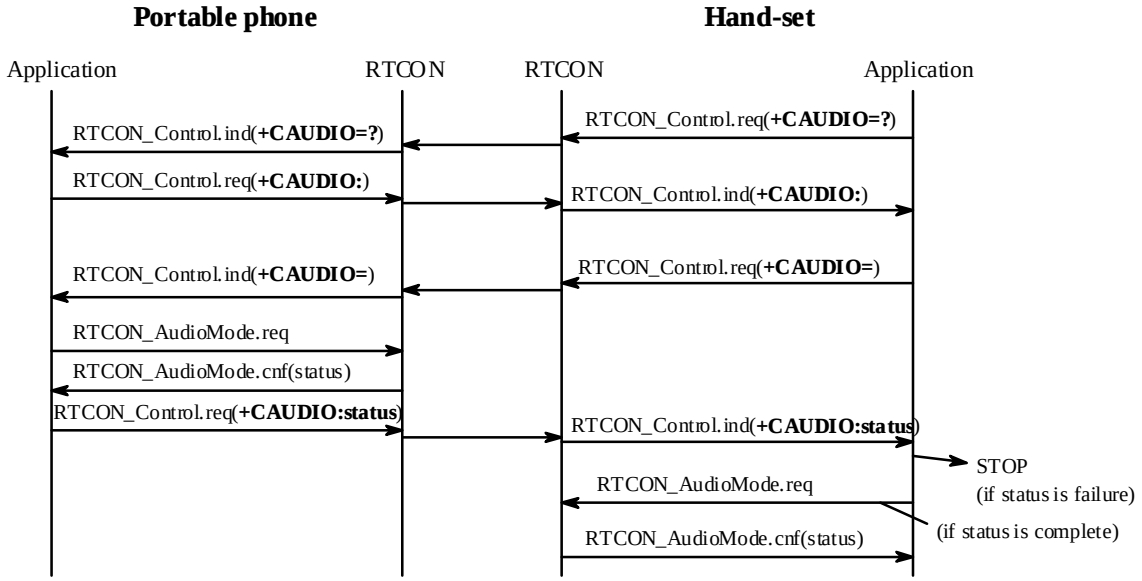


Figure 11.6-5

RTCON_AudioMode is issued when the following request in Call Control is detected by the application.

+CAUDIO for the case that codec type is other than 32 kbps ADPCM

11.7 Implementation Requirements / Recommendations

This chapter shows the desirable implementation of 32 kbps ADPCM mode, in order to make clear the audio transmission procedure and help manufacturers to implement it.

11.7.1 Basic Requirement

(1) 20 ms of 32 kbps ADPCM (80 bytes) is stuffed into one IrLAP frame.

(2) When RTCON contacts the codec circuit, it sends audio data, stuffed in the receipt buffer, to the codec and gets generated audio data from the codec at the same time. This access is repeated every 20 ms or a multiple of 20 ms.

(3) At the secondary station, when errors occur (full receipt buffer when Indication arrives or insufficient data in the transmission buffer when send-request is issued), overflow data should be discarded or dummy code ‘FF’ should be used to make up for the insufficiency. Then, the timing of access to the codec circuit should be changed.

(4) At the primary station, when the receipt buffer is empty (due to the reception of an RR frame) when access is made by the codec circuit, dummy code ‘FF’ should be used to make up for the insufficiency, if the codec circuit does not support the muting function which can suspend the generation of audio signals when there is no data to decode.

(5) If a echo control is needed, it is implemented in hand-sets.

11.7.2 Recommendation for Implementation Type

If RTCON cannot be implemented as both the primary and the secondary procedures within a Telecom device, then it is recommended that the RTCON in the portable phone executes the primary procedure, and the RTCON in the cradle/PC executes the secondary procedure. It is also recommended that the RTCON supports the primary/secondary exchange procedure described in 11.6.1 in order to correct the local IrLAP role.

The RTCON implementations of primary and secondary station are different. The primary station has one type of implementation. The secondary station has two types of implementation depending on the delay and performance of the hardware, as follows:

- Simple implementation; delay: 30-50 ms, for a general processor
- Delay reduced implementation; delay: 34-38 ms, for a high performance processor

The following chapter uses an example Audio transmission implementation to describe the two types of implementation in detail.

11.7.3 Simple Implementation

11.7.3.1 Normal Procedure Overview

Under IrLAP connection, each station acts as either primary or secondary station. The primary station is the initiating side and can send IrLAP frames as soon as a request is issued by the upper application upon generating 20 ms of ADPCM data. The secondary station is the answering side, and can send an IrLAP frame after receiving an Indication which informs it that the IrLAP frame from the primary station has arrived successfully, if there is a pending request issued by an upper application.

Therefore, the secondary station must always have a Pending-request when the Indication is received, otherwise IrLAP of the secondary station returns an RR frame, which means there is no data to send, and discontinuous data is decided in the primary station. Conversely, if the request to send is issued too early, it causes a delay, because the request is not sent until the Indication arrives.

In this case, the transmission delay from Secondary to Primary is 30-50 ms.

(buffering time in Secondary = 20 ms, pending period of request to send in Secondary = 0-20 ms, IrLAP transmission time = 8.3 ms, receiving process time in Primary = 1.7 ms)

The receipt frame’s data should be suspended until the codec circuit finishes decoding the previous frame’s data and requests the next frame’s data. The request interval is 20 ms, so the maximum suspended period is 20 ms.

Therefore, the delay from Primary to Secondary is 30-50 ms.

(buffering time in Primary = 20 ms, transmission time = 8.3 ms, receiving process time in Secondary = 1.7 ms, suspended period in Secondary = 0-20 ms)

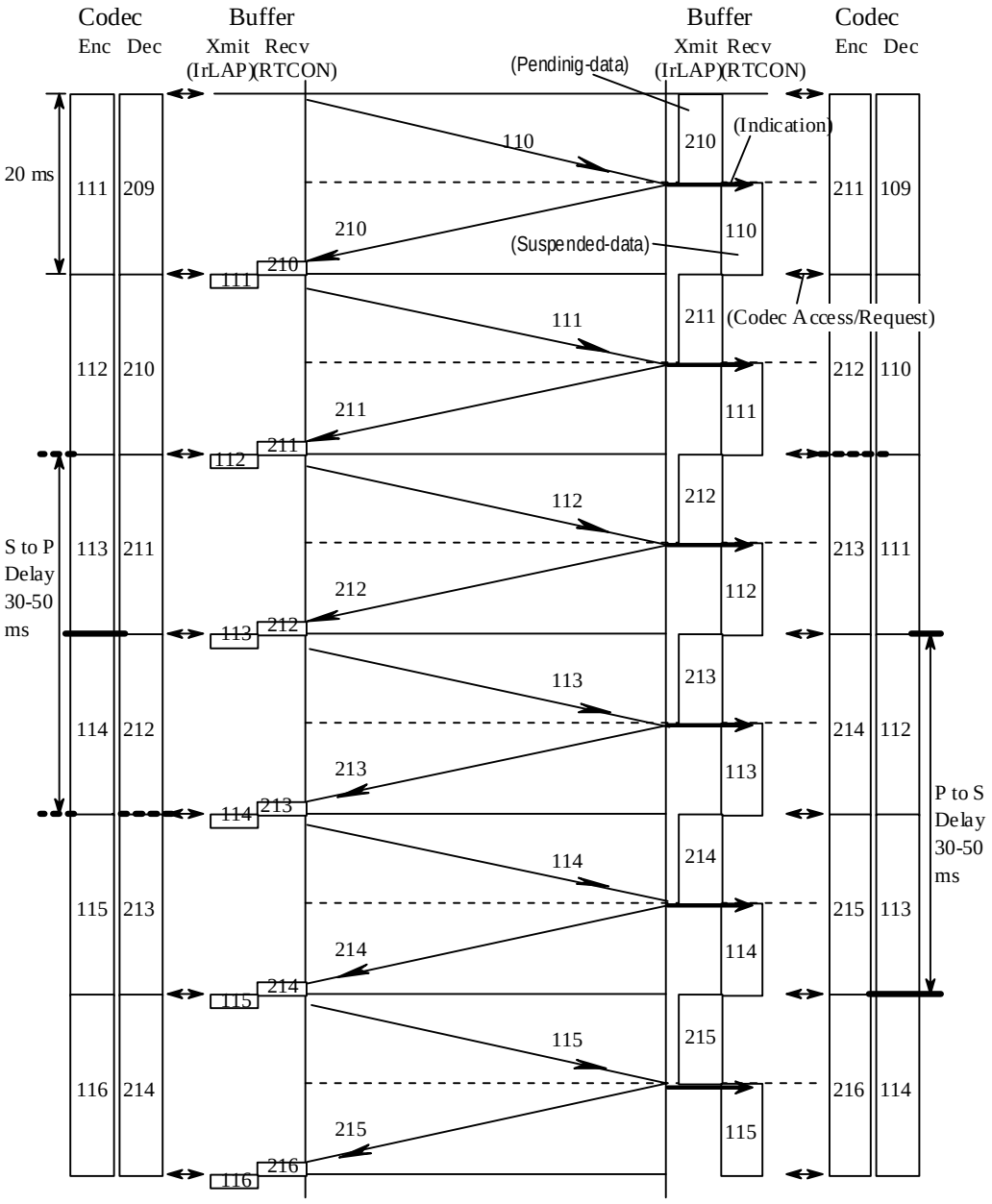


Figure 11.7-1 Normal Sequence

11.7.3.2 Overview of Procedure for Errors

Errors might cause overflow of audio data to be sent, or insufficient audio data for decoding, because the codec circuit is always generating and decoding audio data. Also see section 11.6.1.

Both ADPCM sampling clocks are not synchronized with each other. If the secondary station clock is faster than the primary station, it occurs that two Send-requests exist in one between an Indication and the next Indication, at certain intervals depending on the clock difference. Besides, the Indication is likely to arrive later than expected because the IrLAP frame length increases due to the escape-code -sequence. In this case, one of the two Send-requests should be discarded.

If the secondary station clock is slower than the primary station, it occurs that no Send-request (No-pending) exists in one period between an Indication and the next Indication at certain intervals depending on clock difference. Besides, Indication is likely to arrive earlier than expected because the IrLAP frame has no control data and so its length decreases. In this case, IrLAP in the secondary sends an RR frame, and then IrLAP in the primary station can't receive data to transfer to RTCON, so RTCON in the primary station should supply 'FF' code as dummy data to the codec circuit.

If the request from the codec circuit and the receipt of the Indication happen at nearly the same time, errors caused by instability of Indication arrival might occur for every Indication reception. To handle this case, when an erroneous procedure happens, the interval of requests from the codec circuit should be changed.

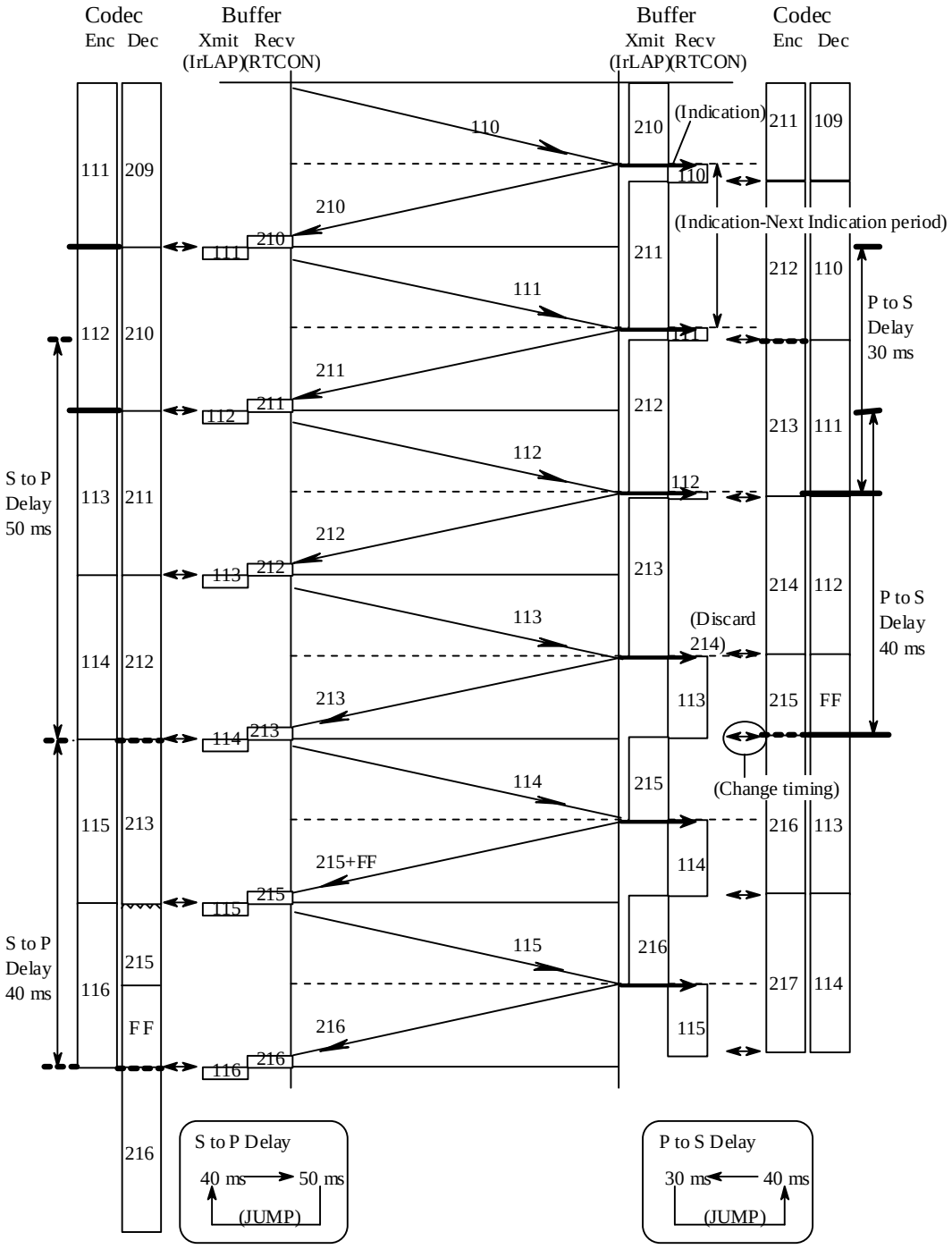


Figure 11.7-2 Secondary Clock Fast Sequence

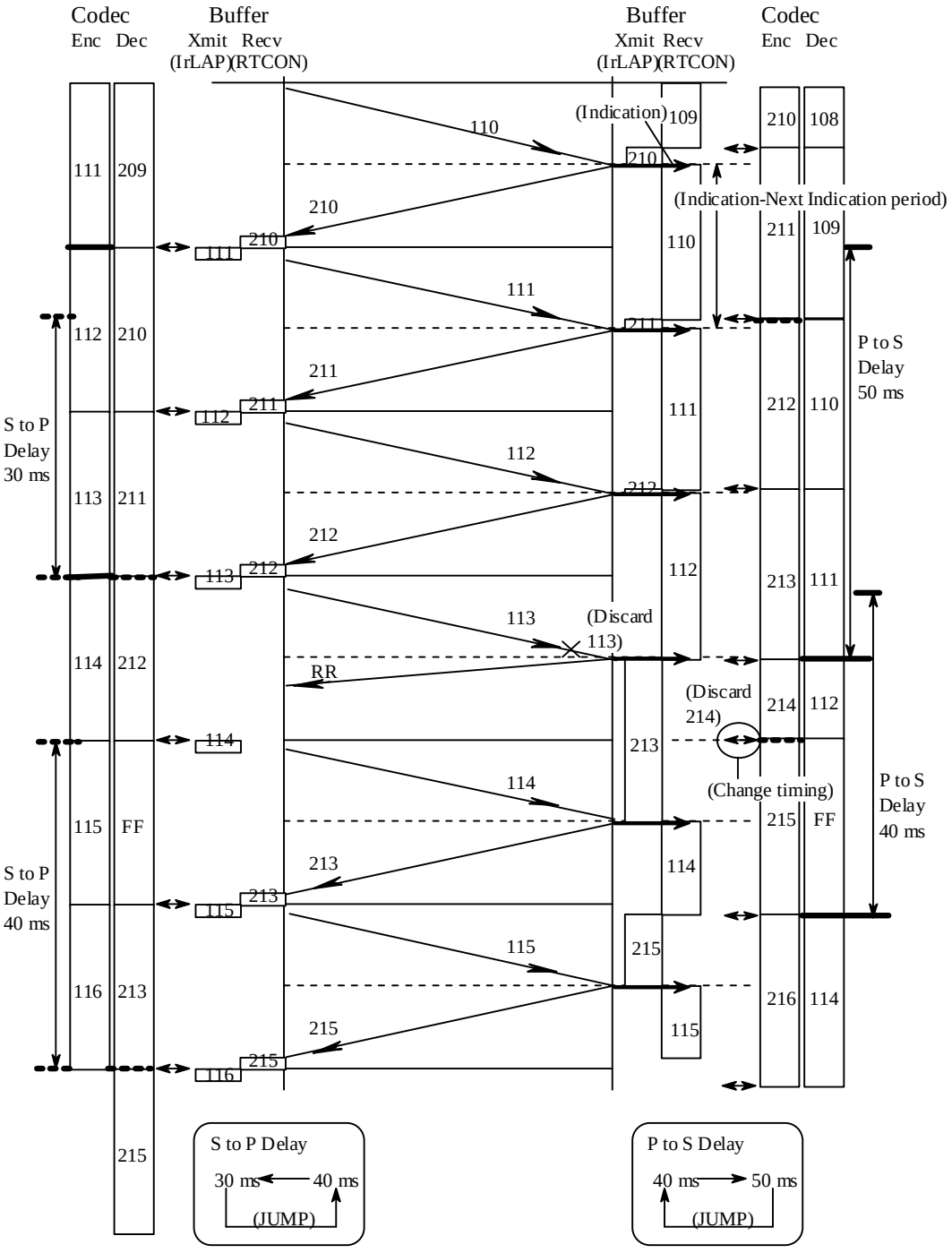


Figure 11.7-3 Secondary Clock Slow Sequence

11.7.3.3 State Chart

Primary Station

State	Event	Action	Next State
TALK (ADPCM) XMIT	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE(P)
	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize < AudioTxFIFOSize	Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize	TALK (ADPCM) XMIT
	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize = AudioTxFIFOSize	RTCON_Audio.Indication(AudioRxBufferData) Clear AudioRxBuffer /* fill with 0xff */ AudioRxBufferState = Empty Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize Pop AudioData from AudioTxFIFO /* All data in AudioTxFIFO */ AudioTxFIFOCount = 0 Pop ControlData from ControlTxFIFO /* not exceed Control field size */ TTP_Data.Request(AudioData + ControlData)	TALK (ADPCM) RECV
	RTCON_Control.Request(ControlData)	Push ControlData into ControlTxFIFO	TALK (ADPCM) XMIT
TALK (ADPCM) RECV	TTP_Disconnect.Indication	RTCON_Disconnect.Indication	IDLE(P)
	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize < AudioTxFIFOSize	Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize	TALK (ADPCM) RECV

	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize = AudioTxFIFOSize	RTCON_Audio.Indication(AudioRxBufferData) Clear AudioRxBuffer /* fill with 0xff */ AudioRxBufferState = Empty Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize Pop AudioData from AudioTxFIFO /* All data in AudioTxFIFO */ AudioTxFIFOCount = 0 Pop ControlData from ControlTxFIFO /* not exceed Control field size */ TTP_Data.Request(AudioData + ControlData)	TALK (ADPCM) RECV
	RTCON_Control.Request(ControlData)	Push ControlData into ControlTxFIFO	TALK (ADPCM) RECV
	TTP_Data.Indication(AudioData) ^ AudioRxBufferState = Empty	Store AudioData in AudioRxBuffer AudioRxBufferState = Full	TALK (ADPCM) XMIT
	TTP_Data.Indication(AudioData + ControlData) ^ AudioRxBufferState = Empty	Strore AudioData in AudioRxBuffer AudioRxBufferState = Full RTCON_Control.Indication(ControlData)	TALK (ADPCM) XMIT
	TTP_Data.Indication(AudioData) ^ AudioRxBufferState = Full	Empty /* discard incoming AudioData */	TALK (ADPCM) XMIT
	TTP_Data.Indication(AudioData + ControlData) ^ AudioRxBufferState = Full	/* discard incoming AudioData */ RTCON_Control.Indication(ControlData)	TALK (ADPCM) XMIT

Secondary Station

State	Event	Action	Next State
TALK (ADPCM)	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE(S)
	TTP_Disconnect.Indication	RTCON_Disconnect.Indication	IDLE(S)
	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize < AudioTxFIFOSize	Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize	TALK (ADPCM)
	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize = AudioTxFIFOSize ^ AudioTxBufferState = Empty ^ AudioRxBufferOverflow = false	RTCON_Audio.Indication(AudioRxBufferData) Clear AudioRxBuffer /* fill with 0xff */ AudioRxBufferState = Empty Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize Pop AudioData from AudioTxFIFO /* All data in AudioTxFIFO */ AudioTxFIFOCount = 0 Pop ControlData from ControlTxFIFO /* not exceed Control field size */ TTP_Data.Request(AudioData + ControlData) AudioTxBufferState = Full	TALK (ADPCM)

	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize = AudioTxFIFOSize ^ AudioTxBufferState = Empty ^ AudioRxBufferOverflow = true	RTCON_Audio.Indication(AudioRxBufferData by AudioRxBufferSize/2) Clear AudioRxBuffer /* fillwith 0xff */ AudioRxBufferState = Empty Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize Pop AudioData from AudioTxFIFO /* All data in AudioTxFIFO */ AudioTxFIFOCount = 0 Pop ControlData from ControlTxFIFO /* not exceed Control field size */ TTP_Data.Request(AudioData + ControlData) AudioTxBufferState = Full AudioRxBufferOverflow = false	TALK (ADPCM CHANGE)
TALK (ADPCM)	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize = AudioTxFIFOSize ^ AudioTxBufferState = Full	RTCON_Audio.Indication(AudioRxBufferData by AudioRxBufferSize/2) Clear AudioRxBuffer /* fillwith 0xff */ AudioRxBufferState = Empty Pop AudioData from AudioTxFIFO /* All data in AudioTxFIFO */ AudioTxFIFOCount = 0 /* discard AudioTxFIFO data */ AudioRxBufferOverflow = false	TALK (ADPCM CHANGE)
	RTCON_Control.Request(ControlData)	Push ControlData into ControlTxFIFO	TALK (ADPCM)
	TTP_Data.Indication(AudioData) ^ AudioRxBufferState = Empty	Store AudioData in AudioRxBuffer AudioRxBufferState = Full AudioTxBufferState = Empty	TALK (ADPCM)
	TTP_Data.Indication(AudioData + ControlData) ^ AudioRxBufferState = Empty	Strore AudioData in AudioRxBuffer AudioRxBufferState = Full RTCON_Control.Indication(ControlData) AudioTxBufferState = Empty	TALK (ADPCM)

	TTP_Data.Indication(AudioData) ^ AudioRxBufferState = Full	/* discard incoming AudioData */ AudioRxBufferOverflow = true AudioTxBufferState = Empty	TALK (ADPCM)
	TTP_Data.Indication(AudioData + ControlData) ^ AudioRxBufferState = Full	/* discard incoming AudioData */ AudioRxBufferOverflow = true RTCON_Control.Indication(ControlData) AudioTxBufferState = Empty	TALK (ADPCM)
TALK (ADPCM CHANGE)	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE(S)
	TTP_Disconnect.Indication	RTCON_Disconnect.Indication	IDLE(S)
	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize < AudioTxFIFOSize/2	Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize	TALK (ADPCM CHANGE)
TALK (ADPCM CHANGE)	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize >= AudioTxFIFOSize/2 ^ AudioTxBufferState = Empty	RTCON_Audio.Indication(AudioRxBufferData) Clear AudioRxBuffer /* fill with 0xff */ AudioRxBufferState = Empty Push AudioUnitData into AudioTxFIFO AudioTxFIFOCount += AudioUnitDataSize Pop AudioData from AudioTxFIFO /* All data in AudioTxFIFO */ AudioTxFIFOCount = 0 Pop ControlData from ControlTxFIFO /* not exceed Control field size */ TTP_Data.Request(AudioData + 40bytes 0xFF + ControlData) AudioTxBufferState = Full	TALK (ADPCM)

	RTCON_Audio.Request(AudioUnitData) ^ AudioTxFIFOCount + AudioUnitDataSize >= AudioTxFIFOSize/2 ^ AudioTxBufferState = Full	RTCON_Audio.Indication(AudioRxBufferData by AudioRxBufferSize) Clear AudioRxBuffer /* fillwith 0xff */ AudioRxBufferState = Empty Pop AudioData from AudioTxFIFO /* All data in AudioTxFIFO */ AudioTxFIFOCount = 0 /* discard AudioTxFIFO data */	TALK (ADPCM)
	RTCON_Control.Request(ControlData)	Push ControlData into ControlTxFIFO	TALK (ADPCM CHANGE)
	TTP_Data.Indication(AudioData) ^ AudioRxBufferState = Empty	Store AudioData in AudioRxBuffer AudioRxBufferState = Full AudioTxBufferState = Empty	TALK (ADPCM CHANGE)
	TTP_Data.Indication(AudioData + ControlData) ^ AudioRxBufferState = Empty	Strore AudioData in AudioRxBuffer AudioRxBufferState = Full RTCON_Control.Indication(ControlData) AudioTxBufferState = Empty	TALK (ADPCM CHANGE)
	TTP_Data.Indication(AudioData) ^ AudioRxBufferState = Full	/* discard incoming AudioData */ AudioTxBufferState = Empty	TALK (ADPCM CHANGE)
TALK (ADPCM CHANGE)	TTP_Data.Indication(AudioData + ControlData) ^ AudioRxBufferState = Full	/* discard incoming AudioData */ RTCON_Control.Indication(ControlData) AudioTxBufferState = Empty	TALK (ADPCM CHANGE)

11.7.4 Delay Reduced Implementation for Secondary

11.7.4.1 Normal Procedure Overview

In order to reduce the delay described above, the secondary station must adapt the timing of send-requests to the Indication receipt timing.

The codec circuit in the secondary station transfers a 4-ms-segment of ADPCM data to RTCON every 4 ms, and when RTCON recognizes the Indication, it starts to count the segments until the count reaches 4, it then stuffs the past 5 segments of data (total 20 ms of data) into one frame and issues a request. The pending period is reduced to 8 ms at most.

In this case, the transmission delay from Secondary to Primary is 34-38 ms

(buffering time in Secondary = 20 ms, pending period of request to send in Secondary = 4-8 ms, IrLAP transmission time = 8.3 ms, receiving process time in Primary = 1.7 ms).

4 ms of pending delay is needed to ensure that pending data exists at Indication reception.

The receipt frame's data should be suspended until the codec circuit finishes decoding the previous frame's data and requests next frame's data. The received-data suspension period is 4-8 ms due to the 4ms request interval from the codec circuit.

Therefore, the delay from the primary station to the secondary station is 34-38 ms

(buffering time in Primary = 20 ms, transmission time = 8.3 ms, receiving process time in Secondary = 1.7 ms, suspended period in Secondary = 4-8 ms).

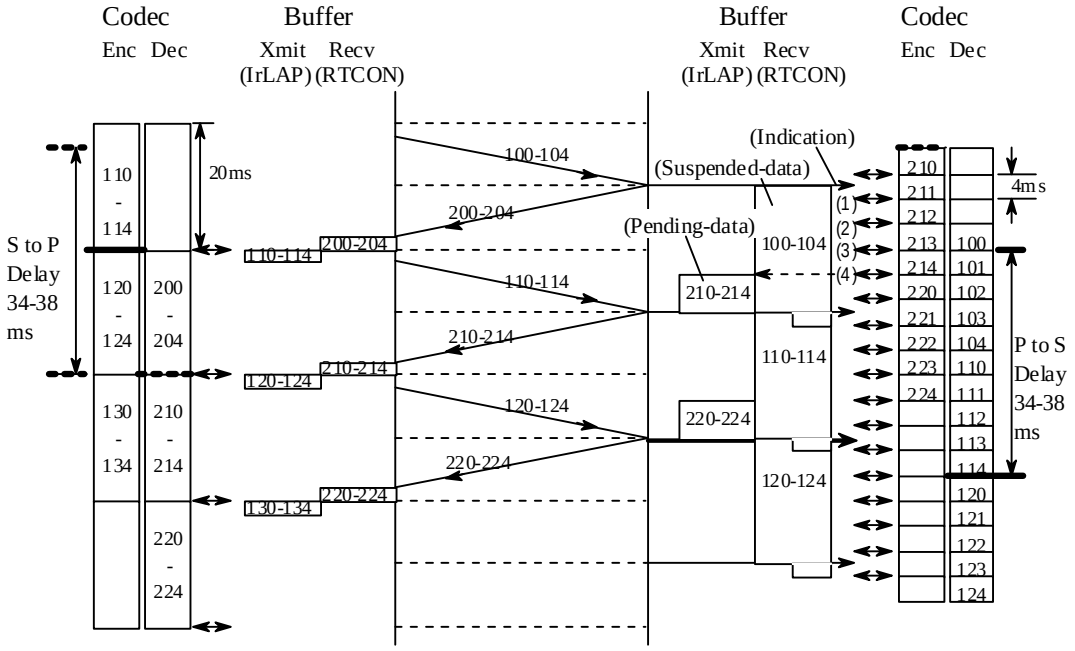


Figure 11.7-4 Normal Sequence

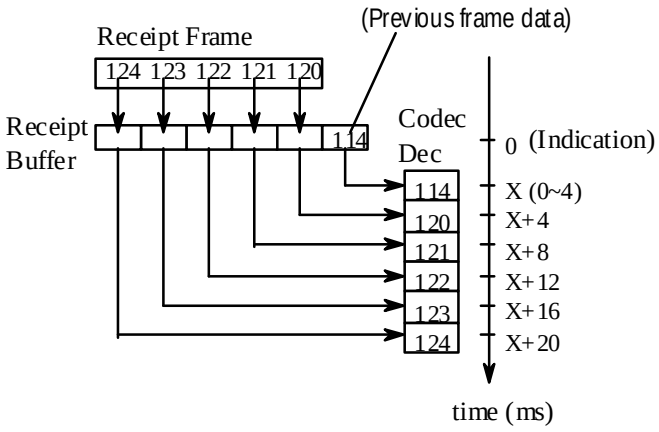


Figure 11.7-5 Suspended Period of One Frame Data in Receipt Buffer

11.7.4.2 Overview of Procedure for Errors

The errors might cause overflow of audio data to be sent or insufficient audio data for decoding. Also see section 11.6.1.

If the secondary station clock is faster than the primary station, or if the Indication arrives later than expected, it occurs that 6 segments exist in one period between an Indication and the next Indication. In this case, one segment of data (16 bytes) should be discarded.

If the secondary station clock is slower than the primary station, or if the Indication arrives earlier than expected, each period lacks one segment to send. In this case, 'FF' codes (16 bytes) are supplied as dummy data.

If the request from hardware and receipt Indication happen at nearly the same time, errors caused by instability of Indication arrival might occur for every receipt of Indication. To handle this case, when an erroneous procedure happens the interval of requests from the codec circuit should be changed.

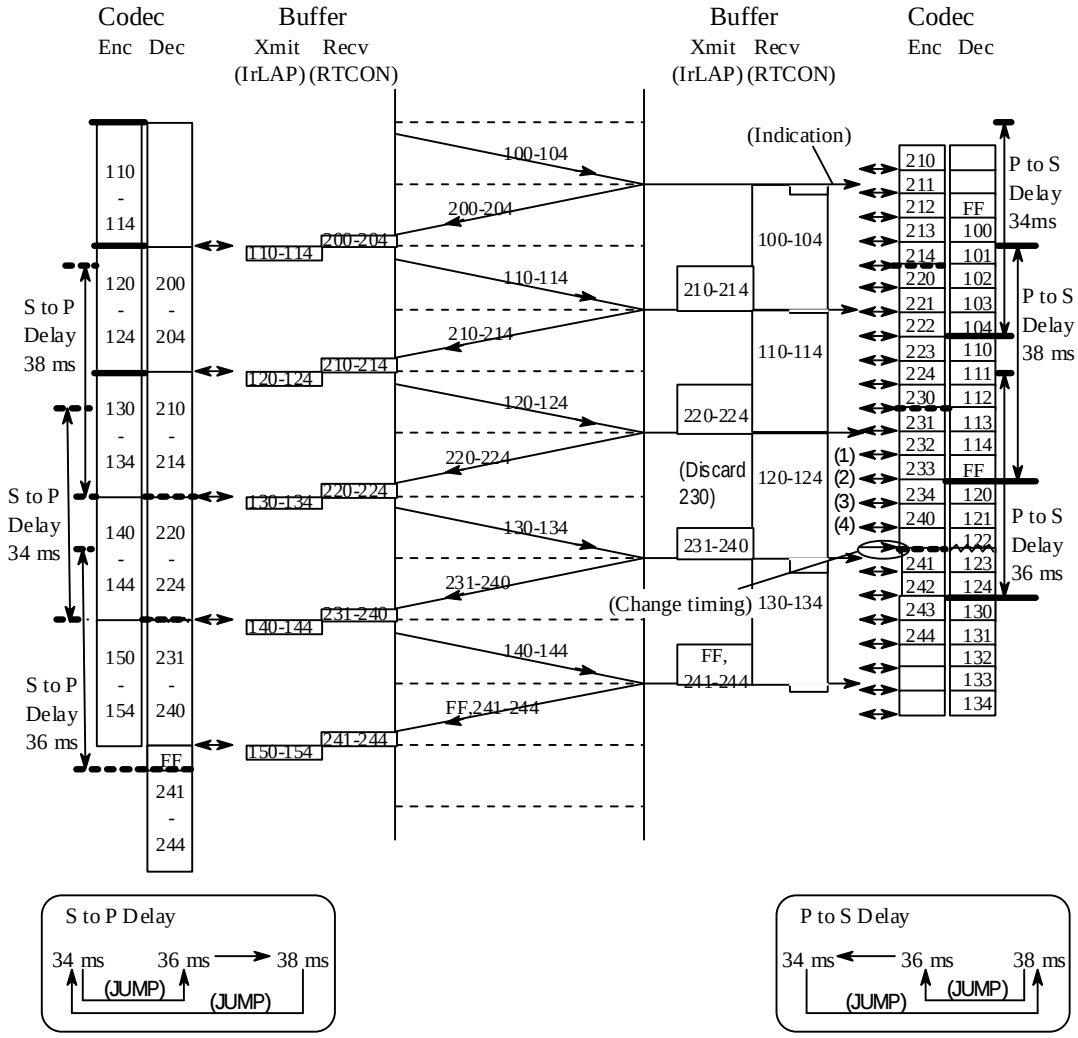


Figure 11.7-6 Secondary Clock Fast Sequence

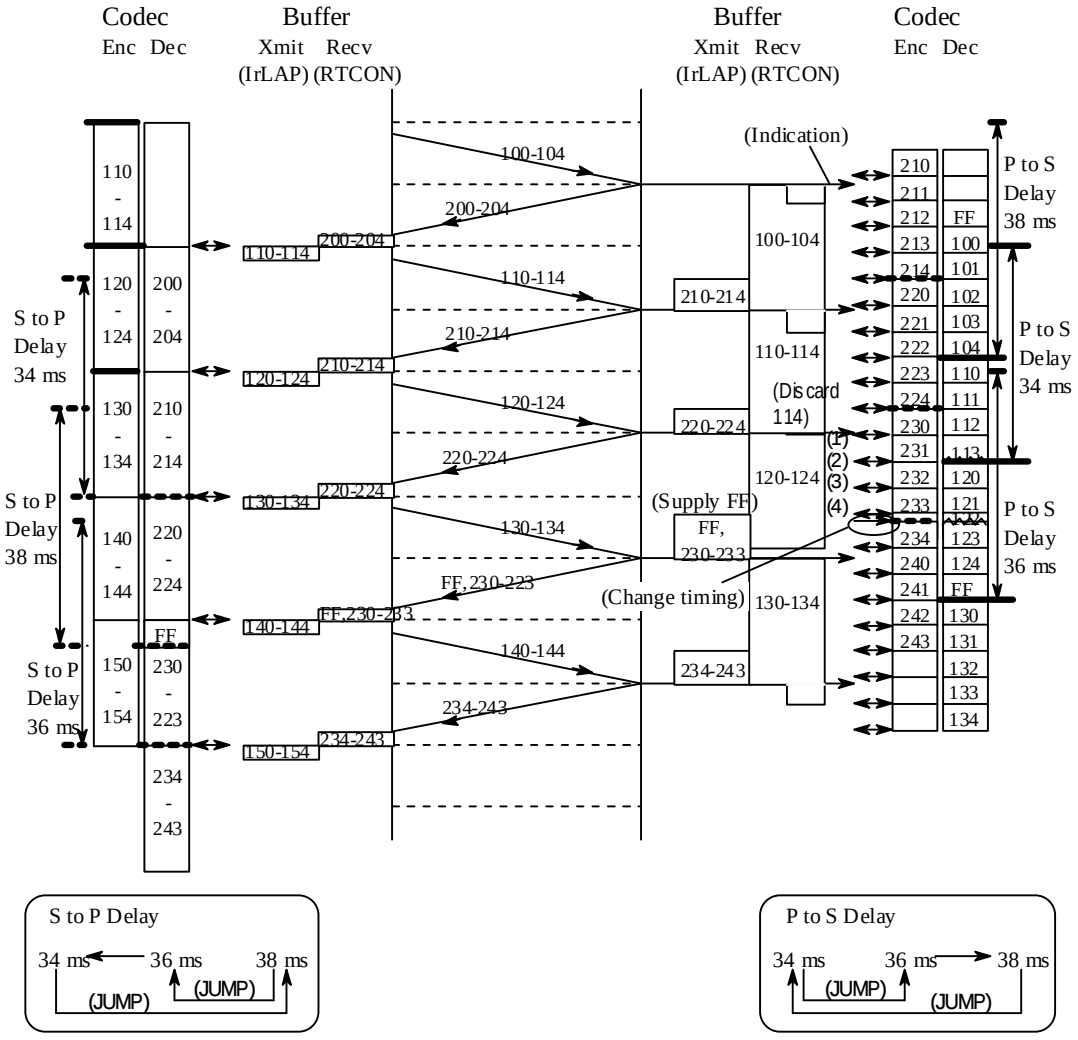


Figure 11.7-7 Secondary Clock Slow Sequence

11.7.4.3 State Definition and Transitions

Secondary Station

State	Event	Action	Next State
STAND BY	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE
	TTP_Disconnect.Indication	RTCON_Disconnect.Indication	IDLE
	RTCON_Mode.Request (Tone, CodecType, LengthAttribute, [Length]) ^ CodecType = ADPCM	AudioFrameSize = 80 bytes PrimaryAudioTxFIFOSize = PrimaryAudioRxFIFOSize = 80 bytes SecondaryAudioTxFIFOSize = SecondaryAudioRxFIFOSize = 16 bytes status = complete RTCON_Mode.Confirm(status)	STAND BY

	RTCON_Mode.Request
	(Tone, CodecType,
	LengthAttribute, [Length])
	^
	CodecType != ADPCM
		Changes Mode
		If (Mode Change is completed){
		status=completed}
		else{
		status = failure}
		RTCON_Mode.Confirm(status)
			STAND BY

	RTCON_State.Request (talk) ^ CodecType = ADPCM	Initialize RTCON Audio Buffer	ADPCM _RECV
	RTCON_State.Request (standby)		STAND BY
	RTCON_Control.Request (ControlData)	Push ControlData into SecondaryControlTxBuf TTP_Data.Request (Flag + (Pop from SecondaryControlTxBuf by Max86bytes))	STAND BY
STAND BY	TTP_Data.Indication (RTCONData)	SecondaryControlRxBuf = ControlPart of RTCONData	STAND BY

		RTCON_Control.Indication (SecodaryControlRxBuf) SecondaryControlRxBuf = Empty	
ADPCM_RECV	RTCON_State.Request (talk) ^ CodecType = ADPCM	Initialize RTCON Audio Buffer	ADPCM_RECV
	RTCON_Disconnect.Request	TTP_Disconnect.Request	IDLE
	TTP_Disconnect.Indication	RTCON_Disconnect.Indication	IDLE
	RTCON_State.Request (standby)		STAND BY
	RTCON_Audio.Request (AudioUnitData) ^ SecondaryAudioTxFIFOCount + AudioUnitDataSize < SecondaryAudioTxFIFOSize ^ ShortResetFlag=0 ^ DropResetFlag=0	Increase SecondaryAudioTxFIFOCount by AudioUnitDataSize Push AudioUnitData into SecondaryAudioTxFIFOData AudioUnitData= Pop from SecondaryAudioRxFIFOData by AudioUnitDataSize RTCON_Audio.Indication(AudioUnitData)	ADPCM_RECV

	RTCON_Audio.Request
	(AudioUnitData)
	^
	SecondaryAudioTxFIFOCount
	+ AudioUnitDataSize
	<
	SecondaryAudioTxFIFOSize
	^
	ShortResetFlag=1
		Increase SecondaryAudioTxFIFOCount
		by AudioUnitDataSize
		Push AudioUnitData
		into SecondaryAudioTxFIFOData
		If (SecondaryAudioTxFIFODataCount exceeds
		over a half size of SecondaryAudioTxFIFOSize){
		SecondaryAudioTxFIFOData = Empty
		ShortResetFlag=0}
			ADPCM_RECV

ADPCM_RECV	RTCON_Audio.Request (AudioUnitData) ^ SecondaryAudioTxFIFOCount	Increase SecondaryAudioTxFIFOCount by AudioUnitDataSize Push AudioUnitData into SecondaryAudioTxFIFOData	ADPCM_RECV

	+ AudioUnitDataSize
	\<
	SecondaryAudioTxFIFOSize
	^
	DropResetFlag=1
		If (SecondaryAudioTxFIFODataCount exceeds
		over a half size of SecondaryAudioTxFIFOSize){
		SecondaryAudioTxFIFOData = Empty
		DropResetFlag=0}	

	RTCON_Audio.Request
	(AudioUnitData)
	^
	SecondaryAudioTxFIFOCount
	+ AudioUnitDataSize
	>=
	SecondaryAudioTxFIFOSize
	^
	(n+1) x
	SecondaryAudioTxFIFOSize
	< AudioFrameSize
		Increase SecondaryAudioTxFIFOCount
		by AudioUnitDataSize
		Push AudioUnitData
		into SecondaryAudioTxFIFOData
		SecondaryAudioTxFIFOCount = 0
		SecondaryAudioTxBuf[0 to 3]
		= SecondaryAudioTxBuf[1 to 4]
		SecondaryAudioTxBuf[4]
		= SecondaryAudioTxFIFOData
		n++
		SecondaryAudioTxFIFOData = Empty
		If (SecondaryAudioRxBuf is Empty){
		Push "FFh" into SecondaryAudioRxFIFOData
		by SecondaryAudioRxFIFOSize}
		else{
		Push into SecondaryAudioRxFIFOData
		from SecondaryAudioRxBuf
		by SecondaryAudioRxFIFOSize}
		AudioUnitData =
		Pop from SecondaryAudioRxFIFOData
		by AudioUnitDataSize
		RTCON_Audio.Indication(AudioUnitData)
			ADPCM_RECV

	RTCON_Audio.Request (AudioUnitData) ^ SecondaryAudioTxFIFOCount + AudioUnitDataSize = SecondaryAudioTxFIFOSize ^ (n+1) x	Increase SecondaryAudioTxFIFOCount by AudioUnitDataSize Push AudioUnitData into SecondaryAudioTxFIFOData SecondaryAudioTxFIFOCount = 0 SecondaryAudioTxBuf[0 to 3] = SecondaryAudioTxBuf[1 to 4] SecondaryAudioTxBuf[4] = SecondaryAudioTxFIFOData	ADPCM_RECV

	SecondaryAudioTxFIFOSize
	>= AudioFrameSize
		n++
		SecondaryAudioTxFIFOData = Empty
		TTP_Data.Request
		(Flag
		+ SecondaryAudioTxBuf[0 to 4]
		+ (Pop from SecondaryControlTxBuf
		by Max6byte))
		If (DropFlag=1){DropResetFlag=1, DropFlag=0}
		If (ShortFlag=1){ShortResetFlag=1, ShortFlag=0}
		If (SecondaryAudioRxBuf is Empty){
		Push "FFh" into SecondaryAudioRxFIFOData
		by SecondaryAudioRxFIFOSize}
		else{
		Push into SecondaryAudioRxFIFOData
		from SecondaryAudioRxBuf
		by SecondaryAudioRxFIFOSize}
		AudioUnitData =
		Pop from SecondaryAudioRxFIFOData
		by AudioUnitDataSize
		RTCON_Audio.Indication(AudioUnitData)	

ADPCM_RECV	TTP_Data.Indication (RTCONData) ^ n = 5 */* Detecting the Faster Primary ADPCM Clock */*	SecondaryAudioRxBuf = AudioPart of RTCONData SecondaryControlRxBuf = ControlPart of RTCONData RTCON_Control.Indication (SecondaryControlRxBuf) SecondaryControlRxBuf = null n=1 ShortFlag=1 DropFlag=0	ADPCM_XMIT
ADPCM_RECV	TTP_Data.Indication (RTCONData) ^ n >= 7 */* Detecting the slower Primary ADPCM Clock or the	SecondaryAudioRxBuf = AudioPart of RTCONData SecondaryControlRxBuf = ControlPart of RTCONData RTCON_Control.Indication (SecondaryControlRxBuf)	ADPCM_XMIT

	*delay of Data Indication from Primary */	SecondaryControlRxBuf = null n=1 ShortFlag=0 DropFlag=1	
	TTP_Data.Indication (RTCONData) ^ n = 6	SecondaryAudioRxBuf = AudioPart of RTCONData SecondaryControlRxBuf = ControlPart of RTCONData RTCON_Control.Indication (SecondaryControlRxBuf) SecondaryControlRxBuf = null n=1 ShortFlag=0 DropFlag=0	ADPCM_XMIT
	RTCON_Control.Request (ControlData)	Push ControlData into SecondaryControlTxBuf	ADPCM_RECV

12. Telecom applications IAS entry and service hint bit

12.1 IAS Entries

The information needed for accessing the IrDA Telecom applications is included in the IAS class TELECOM. The following table defines the attributes associated with this class. For further information about IAS entries and access, please refer to the [IRDALMP].

Note that IAS should be the prioritary means of identifying the services supported by a device. There is no need for OBEX's "who" or "target" headers when dealing with Telecom related objects.

Class IrDA:TELECOM's parameters should be listed in ascending PI order, which means that PhoneBookVersion is to come first, then PhoneBookSupport, etc...Should new parameters be introduced later on, they would be listed at the end, after AudioCoding which is the last current parameter when sorted in ascending order.

Class IrDA:TELECOM

Attributes	
IrDA:TinyTP:LsapSel	Integer (0x01)
Parameters	Octet Sequence (0x02)

12.1.1 LsapSel

A link service access point selector is the address of an application. In the IrLMP LM-MUX interface device information, phone book, calendar and messaging services are accessed through the OBEX LSAP selector whose value is defined in the IAS class OBEX. For further information about this IAS class, please refer to the OBEX specification.

Audio and call control services are accessed through TinyTP and it's IrLMP LM-MUX interface. The IrDA:TinyTP:LsapSel attribute of the IrDA:TELECOM class indicates the LSAP selector at which the audio and call control services are located. This selector value must be unique and within the range of [0x01...0x6F]. There should be only one LSAP selector for the audio and call control services, otherwise it may be impossible for an incoming connection to decide which LSAP to connect to.

12.1.2 Parameters

The Parameters attribute uniquely identifies the Telecom services provided by a device. All Telecom service information is packed into this one attribute to allow a single IAS GetValueByClass query.

Parameters attribute is an octet sequence, which consists of one or more 3-tuples with the following format:

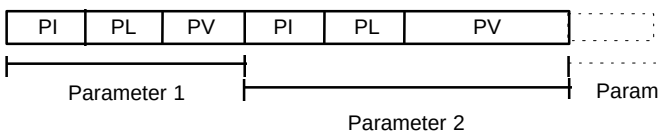


Figure 12.1-1 The structure of the parameters attribute

The fields in the 3-tuples are

Field	Value Type	Description
PI - Parameter Identifier	UINT8	A unique parameter name. If bit 7 is set, the parameter is regarded as critical. Critical parameters are used to identify special services that only work properly with peers.
PL - Parameter Length	UINT8	The length of the PV field in bytes.
PV - Parameter Value	UINT8 sequence	Value, whose meaning depends on the PI.

The Parameters of the various services of the IrDA:TELECOM class are listed in the following sections 12.1.2.1 - 12.1.2.5. If a service is not supported the service parameters may be omitted.

Parameter identifier values in the range of 0x60-0x7F and 0x83 -0xFF are reserved for future use.

12.1.2.1 Phone Book

PI	PI name	PL	PV data type	PV Description	Default Value
0x00	PhoneBookSupport	1	0x01 0x02 0x03	Access support Index support Synchronization support	0
0x01	PhoneBookOptional	1	byte (bit mask) bit 0 bit 1 bit 2 bit 3-7	 <phone-book-incoming-call-history-object> supported <phone-book-outgoing-call-history-object> supported <phone-book-missed-call-history-object> supported Reserved	0
0x02	PhoneBookVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,0)

PHONEBOOKSUPPORT

The PhoneBookSupport parameter is used to indicate the level of the service supported. Default value is 0 for no phone book service available. If a connection oriented phone book service is implemented, this parameter is mandatory.

There is no parameter value for the minimum phone book service support. This is because the minimum service is a connectionless service and no IAS queries are to be expected during a connectionless data exchange. It

should be noted that each of the higher support levels automatically indicates the support of also the lower level services including the connectionless service.

PHONEBOOKOPTIONAL

The PhoneBookOptional parameter is used to indicate the support of the incoming, outgoing and missed call history objects. This parameter is optional and the default value is 0 indicating no support of these objects.

PHONEBOOKVERSION

The PhoneBookVersion parameter is used to indicate the version number of the Telecom specification supported.

This two octet field is provided for compatibility with future versions. The first octet is the major version number and the second octet is the minor version number. This parameter is optional.

12.1.2.2 Calendar

PI	PI name	PL	PV data type	PV Description	Default Value
0x10	CalendarSupport	1	0x01 0x02 0x03	Access support Index support Synchronization support	0
0x11	CalendarVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,0)

CALENDARSUPPORT

The CalendarSupport parameter is used to indicate the level of the service supported. Default value is 0 for no calendar service available. If a connection oriented calendar service is implemented, this parameter is mandatory.

There is no parameter value for the minimum calendar service support. This is because the minimum service is a connectionless service and no IAS queries are to be expected during a connectionless data exchange. It should be noted that each of the higher support levels automatically indicates the support of also the lower level services including the connectionless service.

CALENDARVERSION

The CalendarVersion parameter is used to indicate the version number of the Telecom specification supported.

This two octet field is provided for compatibility with future versions. The first octet is the major version number and the second octet is the minor version number. This parameter is optional.

12.1.2.3 Messaging

PI	PI name	PL	PV data type	PV Description	Default Value
0x20	MessageSupport	1	0x01 0x02 0x03	Access support Index support Synchronization Support	0
0x21	MessageOptional	1	byte (bit mask) bit 0 bit 1-7	 <phone-book-missed-call-history-object> supported Reserved	0
0x22	MessageVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,0)

MESSAGESUPPORT

The MessageSupport parameter is used to indicate the level of the service supported. Default value is 0 for no message service available. If a connection oriented message service is implemented, this parameter is mandatory.

There is no parameter value for the minimum message service support. This is because the minimum service is a connectionless service and no IAS queries are to be expected during a connectionless data exchange. It should be noted that each of the higher support levels automatically indicates the support of also the lower level services including the connectionless service.

MESSAGEOPTIONAL

The MessageOptional parameter is used to indicate the support of the missed message history object. This parameter is optional and the default value is 0 indicating no support of this object.

MESSAGEVERSION

The MessageVersion parameter is used to indicate the version number of the Telecom specification supported.

This two octet field is provided for compatibility with future versions. The first octet is the major version number and the second octet is the minor version number. This parameter is optional.

12.1.2.4 Audio

PI	PI name	PL	PV data type	PV Description	Default Value
0x40	AudioCoding	2+	byte 1 (bit mask) bit 0 bit 1 bit 2 bit 3 bit 4 bit 5 bit 6 bit 7 byte 2 bit 0 bit 1 bit 2 bit 3 bit 4 bit 5-6 bit 7	 32k ADPCM (mand.) PCM64 PDC VSELP PDC PSI-CELP IS54 VCELP QCELP GSM FR Extension GSM HR GSM EFR EVRC MPEG Audio Twin VQ Reserved Extension	0
0x41	AudioVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,0)
0x42	AudioDeviceType	1	byte 1 (bit mask) bit 0 bit 1 bit 2-7	 Primary role Secondary role Reserved	

AUDIOCODING

The AudioCoding parameter is used to indicate the voice coding scheme used. Default value is 0 for no Audio service supported. 32k ADPCM is a mandatory coding method for all Telecom devices and the only method currently described. In order to implement support for any of the other CODECs, formal definition of the service must be included first in this document.

AUDIOVERSION

The Version parameter is used to indicate the version number of the Telecom specification supported.

This two octet field is provided for compatibility with future versions. The first octet is the major version number and the second octet is the minor version number. This parameter is optional.

AUDIODEVICETYPE

The AudioDeviceType parameter is used to indicate the primary/secondary role of the device providing the audio service. This parameter is mandatory for all devices supporting audio.

12.1.2.5 Call Control

PI	PI name	PL	PV data type	PV Description	Default Value
0x50	CallControlVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,0)
0x51	ME/TE Identification	1	0x01 0x02	0:Equipment acts as ME 1:Equipment acts as TE	

CALLCONTROLVERSION

The Version parameter is used to indicate the version number of the Telecom specification supported.

This two octets field is provided for compatibility with future versions. The first octet is the major version number and the second octet is the minor version number. This parameter is optional.

ME/TE IDENTIFICATION

The call control application cannot be used between two MEs or TEs. The ME/TE Identification parameter indicates the category of the equipment.

12.2 Service Hint Bit

The Telephony service hint bit (bit 8) of the IrLMP service hints is used in the device discovery to inform about the Telecom application capabilities of the device. [IRDALMP]

It should be noted that the Telephony bit does not indicate the type of the device in question. It only points out that the device supports some of the Telecom services, for example calendar, call control or audio. Consequently, all PCs, pagers and other non-phone devices are also expected to indicate their Telecom capability with the Telephony bit.