Infrared Data Association

Specifications for Ir Mobile Communications (IrMC)

Version 1.1

March 01, 1999

Working Group Convenors:
Robert K. Lockhart, rob.lockhart@mot.com (Motorola, Inc.)
James Scales, james.scales@nmp.nokia.com (Nokia Mobile Phones, Ltd.),

Editor:
John Stossel, jstossel@drycreek.com (Dry Creek Software)

Authors:

Dave Suvak, davesu@countersys.com (Counterpoint)
Doug Kogan, dkogan@countersys.com (Counterpoint)
John Stossel, jstossel@drycreek.com (Dry Creek Software)
Lars Novak, lars.novak@ecs.ericsson.se (Ericsson)
Patrik Olsson, patrik.olsson@ecs.ericsson.se (Ericsson),
Jörgen Birkler, jorgen.birkler@ecs.ericsson.se (Ericsson)
Robert K. Lockhart, rob.lockhart@mot.com (Motorola)
James Scales, james.scales@nmp.nokia.com (Nokia Mobile Phones, Ltd.)
Philippe Borges, philippe.borges@nmp.nokia.com (Nokia Mobile Phones, Ltd)
Kazuya Anzawa, anzawa@mdev.nttdocomo.co.jp (NTT DoCoMo)
Roy Feague, rfeague@starfish.com (Starfish)

Contributors:

H. Chad Johnson (Ericsson)
Estella Martinez (Ericsson)
Eugene Leung, eleung@waccess.com (Glenayre)
Kohei Akiyama (Hewlett-Packard Japan, Ltd.)
Ray Chock (Calibre)
Raymond Quek (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.)
Heli Helanummi, heli.helanummi@nmp.nokia.com (Nokia Mobile Phones, Ltd.)
Petri Nykänen, petri.nykanen@nmp.nokia.com (IrDA IrMC SIG Chair, Nokia Mobile Phones, Ltd.)
Katsunori Hamada, hamada@mdev.nttdocomo.co.jp (NTT DoCoMo)
Kiyohito Nagata (NTT DoCoMo)
Glen Walant
Barbara Bancroft, bbancroft@pumatech.com (Puma Technology)
Steve Rybicki, srybicki@pumatech.com (Puma Technology)
Scott Simon, ssimon@pumatech.com (Puma Technology)
Tadami Tanabe (SHARP)
Perry Tobin, ptobin@starfish.com (Starfish)
Stéphane Bouet

Document Status: **Version 1.0 - First Release, October 15, 1997**
Version 1.0.1 - Title Correction, January 10, 1998
Version 1.1 - Sync Additions, Existing Errata Merge, March 01, 1999

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. 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 IrMC specification.

1.2 Scope

This document describes the infrared interface between devices adhering to the phase 1 IrDA IrMC 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, short text notes 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 IrDA IrCOMM specification.

This document is intended to be a companion document to the IrDA IrPHY, IrLAP, IrLMP, Lite, Ultra and IrOBEX specifications (see *Chapter 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 1.3, 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
OBEX-AUTH	OBEX Authentication Errata approved in January 1999 Meeting to be incorporated into the OBEX 1.2 specification due for approval in April 1999.
OBEX-ARHDR	OBEX App Response Header Errata approved in January 1999 Meeting to be incorporated into the OBEX 1.2 specification due for approval in April 1999.
ULTRA	Guidelines for Ultra Protocols, Version 1.0, October 15, 1997 (p/o IrMC Specifications Package)
RFC822	RFC #822 - Standard for the Format of the Arpa Internet Text Messages
RFC2045/	RFC 2045 – Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
RFC2047	RFC 2047 – MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text
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) plus the IrDA Telecom Extensions to the IMC vCard Format, Version 1.0, October 15, 1997 (p/o IrMC Specifications Package)
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)

References to the above mentioned documents will be in brackets, e.g., [VCARD], [VCALENDAR]

1.4 Definitions

Calendar Application An application on an IrMC Device that provides a means to add, edit or modify Calendar Objects.

Calendar Object An IrMC Object that provides detailed information about a calendar item. For example, start time, description of the event and end time.

Change Counter A whole number counter that is incremented for each change that is made to a particular Object Store. This counter, a 32 bit unsigned value, must be able to hold large values since it is used by sync engines to keep track of the number of changes since the last database sync. It may wrap to zero as needed. For more information see Sync Anchor

Change Log An object stored on an IrMC Server that provides detailed information about the changes that have occurred to a particular Object Store. Usually the Change Log contains the LUIDs of the objects that have changed, a description of the change (e.g., Modified or Deleted) and a sync anchor indicating when the change occurred.

Database see Object Store

Database ID (DID) A unique 32 bit identifier, used to identify a particular Object Store.

Device Sync Anchor The Sync Anchor of items in a Stored Change Log.

Device Information Object An IrMC Object that provides detailed information about the IrMC device characteristics.

Entry see IrMC Object.

Fast Sync A sync in which Change Logs are used so that only objects that have changed will be synchronized.

Hard&Soft delete To provide the most optimum synchronization the Object Store within a server may support a distinction between deletes. A hard delete is a delete made by a user because that entry is to be removed, a soft delete would be made by a synchronization engine because memory is limited on the server's object store.

Inbox A generic Object Store on an IrMC Device that can hold objects of various formats, such as contact records, calendar appointments and messages. It is typically used as a temporary holding area for objects received from other IrMC Devices. Often, the user can inspect the items in the Inbox, and file them away or delete them.

IrMC Object An object stored on an IrMC device. Examples of IrMC Objects include phone book entries, calendar items, device information and Change Logs. Equivalent to a "record"

IrMC Client The IrMC Client is the device that initiates communication with an IrMC Server. Typical IrMC Clients are PCs. However, in some cases, PDAs, pagers and phones may also be IrMC clients. IrMC Clients can only communicate with IrMC servers. For instance, an IrMC client may request a particular Phone Book record from an IrMC Server.

IrMC Device A device that contains at least an IrMC Server. Examples include phones, pagers, and PCs.

IrMC Server The IrMC Server listens for requests from IrMC clients. Typical Servers are pagers, phones and PDAs. IrMC Servers can not initiate communication, and can only communicate with IrMC clients.

IrMC Static Server An IrMC Server that supports Level 2 and Level 3 Information Access using static indexing only.

IrMC Unique Server An IrMC Server that supports Level 2 and Level 4 Information Access using unique indexing.

Level 1 Information Access A means by which an IrMC Client can send a single IrMC Object to an IrMC Server.

Level 2 Information Access A means by which an IrMC Client can send or retrieve an entire IrMC Object Store to an IrMC Server.

Level 3 Information Access A means by which an IrMC Client can send or retrieve a single IrMC Object to an IrMC server by using static indices.

Level 4 Information Access A means by which an IrMC Client can send or retrieve a single IrMC Object using unique indices. Also defines a way to retrieve a list of IrMC Objects that have changed in a given IrMC Object Store since a previous point in history.

Log Information Object An IrMC Object that contains detailed information about a specific IrMC Object Store. For instance, a Phone Book Log Information Object would contain information such as the number of phone book Objects, the type of Phone Book Object format that is supported, and the space available for additional entries.

LUID (Locally Unique Identifier) A UID (see UID definition) that is unique locally, i.e., to a particular IrMC Server, but may be present on other IrMC Servers.

Message Application An application on an IrMC Device that provides a means to add, edit or modify Message Objects.

Message Object An IrMC Object that provides detailed information about a message item. For example: the name of the sender, a subject field and a detailed message description. Message Objects are different from Note Objects, in that Message Objects are sent from one entity to another. Examples of Message Objects include e-mail objects, or SMS messages.

Modification Log see Change Log

Note Application An application on an IrMC Device that provides a means to add, edit or modify Note Objects.

Note Object An IrMC Object that provides detailed information about a note item. For example: the title of the Note and a description. Note Objects are different from Message Objects in that Note Objects are usually self-authored memos, while Message Objects are sent from one entity to another.

Object Store A database consisting of one or more IrMC objects.

Phone Book Application An application on an IrMC Device that provides a means to add, edit or modify Phone Book Objects.

Phone Book Object An IrMC Object that provides detailed information about a phone book item, also known as a "contact" item. For example: the first name, last name, address, and phone numbers.

Record Equivalent to an IrMC Object or entry

Slow Sync A form of synchronization in which all entries in one or more Object Stores are compared with each other on a field-by-field basis.

Static Indexing Indexing is used for IrMC Object identification within an IrMC Object Store. Static Index values are defined as a finite set of sequential whole numbers, beginning at zero. As objects are created, they are assigned one of the unused index numbers. As objects are deleted, their index numbers can be reused.

Stored Change Log The Change Log stored on an IrMC Server.

Sync (Synchronization) A means by which two IrMC Devices can exchange Change Logs and IrMC Objects to ensure that the data contained within their respective Object Stores is made identical. For example, an IrMC PDA might synchronize its Phone Book Object Store with an IrMC desktop PC's Phone Book Object Store, thereby ensuring that each IrMC device has identical information in its respective Phone Book Object Store.

Sync Anchor A value in a change log that indicates when a record was changed. It some cases, this is a Change Counter, in some cases it is a Timestamp, in some cases it is both.

Sync Engine Anchor The Sync Anchor saved by the Sync Engine, and used as basis for subsequent Change Log requests and sync operations.

Sync Engine A software application that manages the synchronization process between two or more disparate databases and/or devices.

Timestamp A value that represents a specific date and time. In the context of Sync Anchors, it refers to the last time a record was modified.

Transmitted Change Log The Change Log sent to the IrMC Client.

Unique Indexing Indexing is used for IrMC Object identification within an IrMC Object Store. Unique indexing requires that the index numbers consist of a finite set of whole numbers. Unique indexing does not require that those numbers be sequential or start at zero. As objects are created, they are assigned a unique index value that has not been used previously.

UID Unique Identifier, a number assigned to an object in an Object Store. UID values are never reused. They are unique forever. Note that in practice, numbers do not have to be unique *forever*, they must only be unique as long as some sync engine remembers the UIDs. (also see LUID).

1.5 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.

1.6 Common Elements

CHARACTER DEFINITIONS

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

<*carriage-return*> ::= ‘Character with ASCII value 13 decimal.’

<*CRLF*> ::= <*carriage-return*><*line-feed*>

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

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

<*default-alpha-char*> ::= ‘Any alphabetic character in the default character-set., e.g., A, B, C, D’

<*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**”

<*alphanumeric*-char> ::= <*common-digit*> **|** <*default-alpha-char*> | **“+”** | **“-“**

<*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)’

2. IrMC Framework

The IrDA IrMC specification defines the rules for utilization of IR in wireless communications equipment, e.g., in mobile handsets, PDAs, PCs, notebook computers and pagers.

2.1 IrMC Applications

The IrMC specification enables IrMC Object exchange between a variety of applications. For instance, the IrMC specification enables easy exchange of:

- business cards between Phone Book Applications
- text messages between Messaging Applications
- calendar and todo items between Calendar Applications
- short notes between Note applications

The IrMC Call Control specification enables call control of mobile handsets. The IrMC Audio Specification enables real time audio transmission respectively. These two specifications are for communication between a handset and PC or a handset and car cradle.

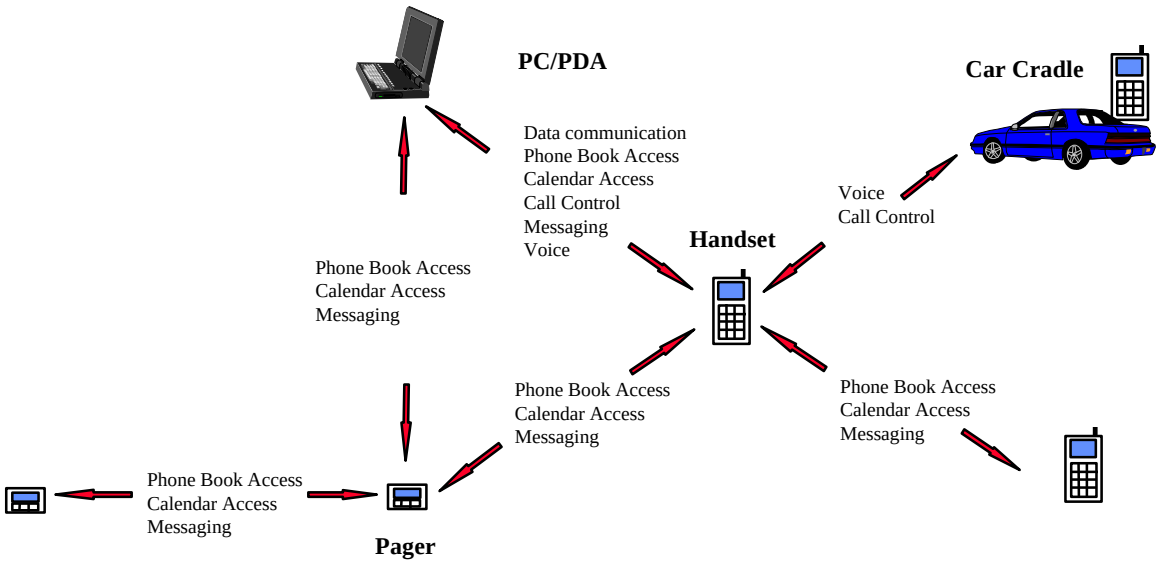


Figure 2-1 Sample IrMC Applications

2.2 Security

Secure information 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.

2.3 Data Transmission

The IrMC 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 IrMC devices.

2.4 Physical Layer Considerations

Because the power consumption requirements of the IrMC 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

IrMC devices to be capable of 20 cm link distance from IrMC device to IrMC device and 30 cm from IrMC device to a standard IrDA device, is optional and the original IrDA-SIR values may be used in all IrMC device implementations. Despite the shorter communication range of some IrMC 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.

2.5 Infrared Functionality

The IR functionality of an IrMC 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

2.5.1 Atomic information exchange

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

Note: Atomic information exchange does not imply that only single IrMC objects are being exchanged. A number of objects can be combined into a single "stream" of IrMC Objects which can then be exchanged during an Object Exchange. This type of "streaming" should not be confused with the "stream-oriented information exchange" defined in section 2.5.2.

2.5.2 Stream-oriented information exchange

Data services of cellular networks typically fall into this category. For instance, the ability to use a cellular phone as a modem to connect to the Internet. 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 IrDA IrComm Specification. The IrMC specifications do not redefine these services although they are important for any device implementing stream-oriented information exchange.

Note: The stream-oriented information exchange referred to in this section should not be confused with the stream of IrMC Objects which can be exchanged as defined in Chapter 2.5.1.

2.5.3 Time-bounded information exchange

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

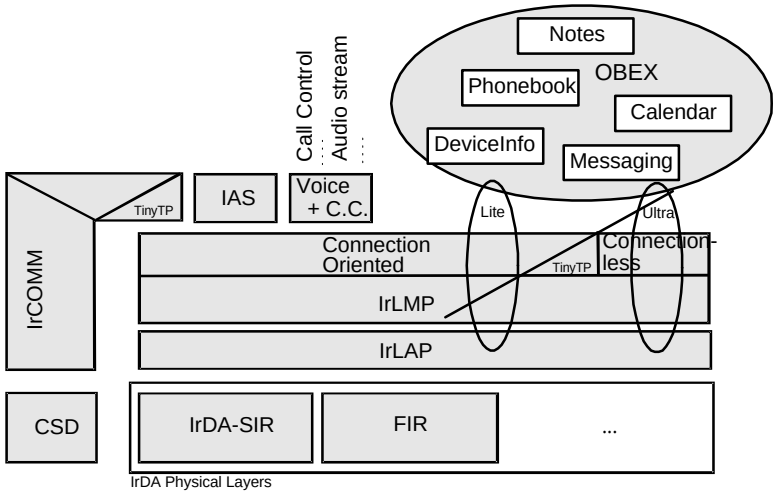


Figure 2-2 The IrDA IrMC Framework

2.6 Supported IrMC Object Stores and Object Formats

The IrMC Specification 1.1 defines a specific object formats for each of the supported applications. Other object formats can be added in the future. It is also possible to use non-supported formats. The Object Store and Object Formats defined by this v1.1 IrMC Specification include:

1. **Phone Book Object Store: vCard 2.1 format** as defined in [VCARD]. Phone Book Objects and the vCard 2.1 support considerations are described in Chapter 7 of this specification.
2. **Calendar Object Store: vCalendar 1.0 format** as defined at [VCALENDAR] . Calendar Objects and the vCalender 1.0 support considerations are described in Chapter 8 of this specification.
3. **Message Object Store: vMessage format**. Message Objects and the vMessage format are defined in Chapter 9 of this specification. There are actually three Message Object Stores: In Box, Out Box and Sent Box. *Note: vMessage is a format defined in this specification and can not be found at www.imc.org.*
4. **Note Object Store: vNote format.** Note Objects and the vNote format are defined in Chapter 10 of this specification. *Note: vNote is a format defined in this specification and can not be found at www.imc.org.*
5. **Change Log Object Store** as defined in Chapter 5 of this specification.
6. **Information Log Object Stores** as defined in Chapters 7, 8, 9, 10 of this specification for Phone Books, Calendar, Message and Note Object Stores, respectively.
7. **Device Information Object** which contains property information about the device. This object is defined in Chapter 6 of this specification.
8. **Change Counter Object** containing the last change counter used as defined in Chapter 5 of this specification.
9. **Real Time Clock Object:** which contains the current date and time of the device. This object is defined in Chapter 4 of this specification.

2.7 Obex Information Exchange

All atomic information exchange is based on the use of OBEX-operations. All IrMC Object Stores and IrMC Objects are accessible through OBEX and identified with the IrMC-naming convention, i.e. <*phone-book-stream-object-name*>, <*calendar-indexed-object-name*>, etc. OBEX Data Exchange is defined in more detail in Chapter 4 of this specification.

2.8 Atomic Information Exchange Levels

Figure 2-3 shows the four possible levels of atomic information exchange that are detailed in this specification. The way an Object Store is organized and accessed depends on the level of support implemented in the IrMC Server. An IrMC Server implementing any of the higher support levels also has to provide support of the lower levels, with the exception that those devices supporting Level 4 do not have to support Level 3. The level of the support must be unambiguously identified in the Information Log (see Chapter 2.9) and in the IAS (see Chapter 13)

	LEVELS **Level 1 Server**	**REQUIRED** **Level 2 Server**	**LEVELS** **Level 3 Server**	**REQUIRED** **Level 4 Server**
LEVEL OF SUPPORT IDENTIFIED IN INFORMATION LOG				
Level 1 (Minimum Level) OBEX PUT Object in Inbox	MUST IMPLEMENT			
Level 2 (Access Level) OBEX GET/PUT Entire Object Store	MUST IMPLEMENT	MUST IMPLEMENT		
Level 3 (Index Level) OBEX GET/PUT Objects by Static Indices	MUST IMPLEMENT	MUST IMPLEMENT	MUST IMPLEMENT	
Level 4 (Sync Level) OBEX GET/PUT Objects by Unique Indices Change Log Support Change Counter Support	MUST IMPLEMENT	MUST IMPLEMENT	OPTIONAL	MUST IMPLEMENT

Figure 2-3 Information Exchange Table

2.8.1 Level 1 Information Exchange (Minimum Level)

This level provides a means by which an IrMC Client can send a single IrMC Object to an IrMC Server. When received, the IrMC Server stores the object in a generic IrMC Inbox with the original name of the object. The name of the pushed objects is specified as *<object-store-minimum-support-object-name>* and the object is specified as *<object-store-minimum-support-object>*. It is recommended that no automatic actions should be taken based on the name of the object, e.g. receiving a new Phone Book vCard object should not cause that object to be automatically stored in the Phone Book Object Store. It is recommended that new objects should be moved to the Phone Book Object only when the user intervenes.

All IrMC Servers must be able to receive objects using Level 1 Information Exchange. It is not required that all IrMC Clients be capable of sending objects using Level 1.

It should be noted that the names of the IrMC objects may vary between the client and the server due to various system specific naming rules.

Level 1 Information Exchange can occur using connectionless data transfer, such as Ultra, or using connection-oriented data transfer, such as IrLMP. For more information on the use of these protocols, and rules for switching between the two, refer to Chapter 3.3 Connection vs. Connectionless Switching.

Objects received as connectionless data (e.g., over Ultra) will be automatically stored in the Inbox. Those Objects received over connections will be stored in the Inbox if the pathname is not one of the reserved names which start with “telecom/”. For example, if an object with the name of “1.vcf” is received, it shall be stored in the Inbox. If the object name is preceded by a Level 3 pathname, e.g., “telecom/pb/1.vcf”, it shall be stored in the Phonebook as a Level 3 object. If the object name is preceded by a Level 4 pathname, e.g., “telecom/pb/luid/1.vcf”, it shall be stored in the Phonebook as a Level 4 Object. *NOTE: It is recommended that the names of objects destined for the Inbox should not contain characters such as / \ or : which might be interpreted as pathname delimiters by the server receiving the objects.*

Some devices may not have strict pathname checking. In order to maximize the probability of interoperability, it is recommended that the OBEX name of the object not be a number if the intent is for the Object to placed in the Inbox. For example, “james.vcf” or “1james.vcf” will be stored in the Inbox. “1234.vcf” might be treated as indexed objects and stored in the Phone Book by devices that do not have strict pathname checking. (See Level 3 Information Exchange).

The Inbox is intended to handle OBEX frames which contain a single Object. The IrMC Inbox can optionally be designed to handle OBEX frames which contain multiple Objects, e.g., a string of vCards. For those clients that send OBEX frames consisting of multiple objects, it should be noted that those Level 1 Servers which can only store a single Object in the Inbox may lose one or more of the Objects in such an OBEX frame. Before sending an OBEX frame containing multiple objects to an IrMC Server, the IrMC client can request the Device Information Object (See Chapter 6) in order to verify that the server can support such frame:

(a) if the server does not support the Device Information Object OR if the INBOX property is not present in the Device Information Object OR if the INBOX property in the Device Information Object has a value of SINGLE, then the server can not successfully parse an OBEX frame containing multiple objects.

(b) if the INBOX property value is MULTIPLE, then the server can parse an OBEX frame containing multiple objects. Note that the objects must be of the same type, e.g., all vCard Objects, or all vCalendar objects.

The way that the Inbox is cleared is implementation specific. The easiest way to delete an Object from the Inbox is when the user selects whether to accept or reject the Object.

Level 1 (Minimum Level) Example: Four vCard Phone Book objects 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 Objects received by a remote push]
Yoshiko.vcf
Alex.vcf
Home.vcf
John Smith.vcf

2.8.2 Level 2 Information Exchange (Access Level)

A means by which an IrMC Client can send or retrieve an entire IrMC Object Store. In some cases, the stream may be an entire IrMC Object Store.

Access Level support provides a read-all/write-all functionality. Servers with the Access Level support create or act upon an object stream containing all the objects in an Object Store. Note that inbox objects are not taken into account when building this stream of objects. This stream is named <*object-store-stream-object-name*> and its format is defined as <*object-store-stream-object*>.

Level 2 Information Exchange requires connection-oriented services. Furthermore, Level 2 Servers must support Level 1 also.

For example, the Phone Book Objects as a vCard stream are shown in *Figure 2-4 A Stream of All vCards in a Phone Book*.

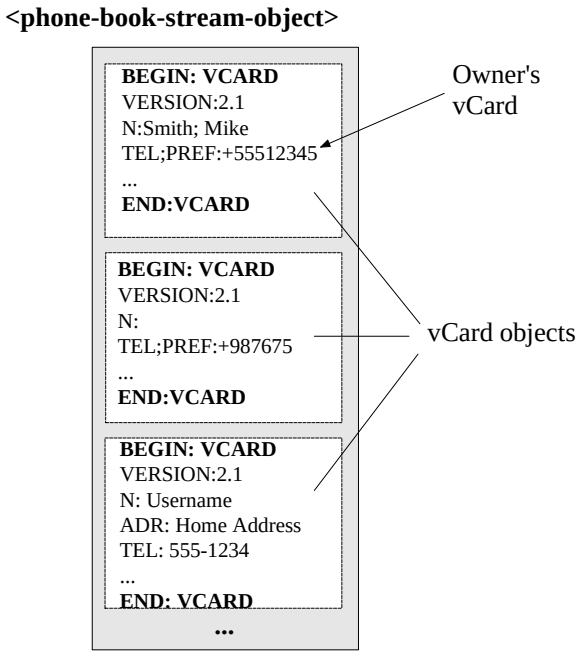


Figure 2-4 A Stream of All vCards in a Phone Book

Level 2 (Access Level) Example: In this example, a device supports Level 2 Information Exchange. A Phone Book stream object "telecom/pb.vcf" allows easy Level 2 GET/PUT access to the entire Phone Book Object Store.

telecom/pb.vcf [vCard stream object listing all the Objects in the Phone Book Object Store.]

2.8.3 Level 3 Information Exchange (Index Level):

A means by which an IrMC Client can send or retrieve a single IrMC Object to and from an IrMC Server's Object Store. This level provides a means to read, write, modify and delete individual objects in an object store. Each IrMC Object is assigned a static index by the IrMC Client and named as *<object-store-indexed-object-name>*. The format of these objects is defined as *<object-store-indexed-object>*.

The indices are static and expressed as whole numbers starting from 0. The indices do not necessarily 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 Object Store. Index "0" is a special index. In the case of the Phone Book application, it is assigned to hold the Phone Book Object of the owner of the device.

Level 3 Information Exchange relies on connection-oriented services. Furthermore, Level 3 Servers must support Level 1 and Level 2 Information Exchange.

A sample <*object-store-stream-object*> consisting of statically indexed vCards is shown in Figure 2-5.

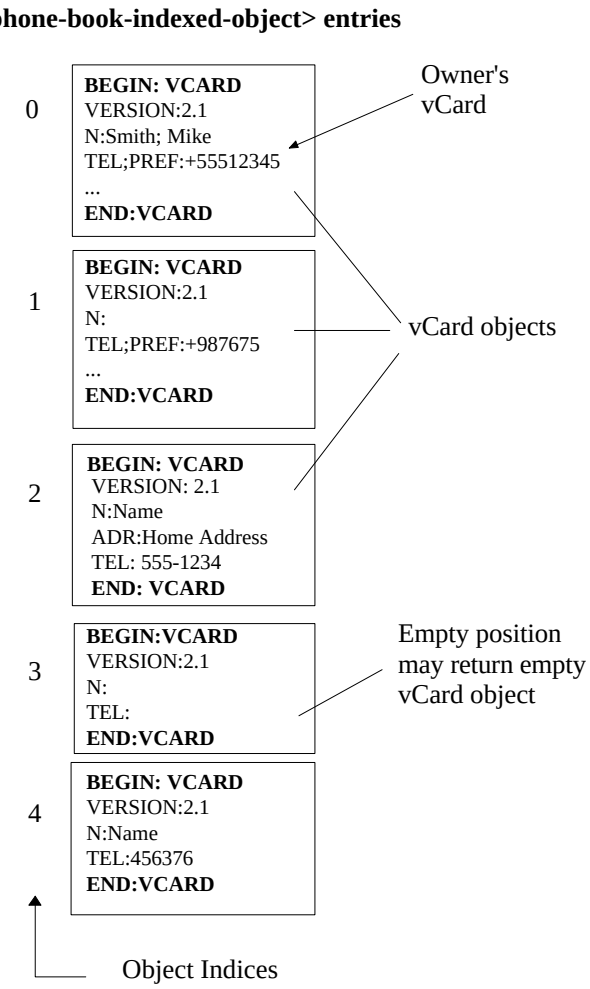


Figure 2-5 vCard entries with indices

Level 3 (Index Level) Example: In this example, a device supports Level 3 Information Exchange. For Level 3 each vCard in the phone book is assigned an individual static index.

telecom/pb/0.vcf [vCard of the owner of the device]
telecom/pb/1.vcf [phone book objects with individual indices]
telecom/pb/2.vcf
telecom/pb/3.vcf
telecom/pb/4.vcf

2.8.4 Level 4 Information Exchange (Sync Level):

A special form of information exchange by which an IrMC Client can retrieve a list of IrMC Objects that have changed in a given IrMC Object Store since a previous point in history. This information is used primarily as a part of a data synchronization operation.

Level 4 also defines and mandates a new type of indexing – unique indexing. For more information on unique indexing, see Section 4.1.2 - Unique Indexing. It should be noted that index "0" is a special index. In the case of the Phone Book application, it is assigned to hold the Phone Book Object of the owner of the device.

An additional log object named <*object-store-change-log-object-name*> is to be implemented. This object, defined as <*object-store-change-log-object*>, is to hold information about all changes in the any of the <*object-store-indexed-object*>. Each time an object is added, modified or deleted in an Object Store, an entry must be made in the Change Log.

Level 4 Information Exchange relies on connection-oriented services. Furthermore, Level 4 Servers must also support Level 1 and Level 2 Information Exchange. They do not have to support Level 3.

Level 4 has two types of mechanisms for identifying which records to sync. These mechanisms are called Sync Anchors. The first type of anchor is a Change Counter and the second one is Timestamps. All synchronization clients are required to support both types of Sync Anchors. Sync Servers are required to support at least one type of anchor. If only one can be supported by a server, the Change Counter is preferred.

There are a couple of very important differences between using Change Counters and Timestamps. First, when using Timestamps the database on the Server has to be locked for other interactions during synchronization. This means that simultaneous multiple sync scenarios are prevented, as is the ability for users to enter data during a synchronization. Furthermore, serious errors in synchronization can occur if the time on the Server is changed backwards. If the device hosting the sync server does not support UTC time it is strongly recommended not to use timestamps.

A detailed description of data synchronization and Change Logs is provided in Chapter 5 of this specification.

Level 4 (Synchronization Level) Example: In this example, a device supports Level 4 Information Exchange. For Level 4, individual Phone Book objects are stored in the telecom/pb/luid/ path. The Phone Book Change log object contains information about all changes that have been made to the phone book. The Phone Book Change Counter object contains the most recently assigned Change Counter value, if Change Counters are supported by the device. For more details on Level 4, please see Chapter 5 Synchronization.

telecom/pb/luid/0.vcf	[phone book objects of owner]
telecom/pb/luid/189a3.vcf	[phone book objects with individual indices]
telecom/pb/luid/813cd.vcf	[phone book objects with individual indices]
telecom/pb/luid/cc.log	[phone book Change Counter object]
telecom/pb/luid/XXXX.log	[phone book Change Log object where XXXX is a change counter value]

2.9 Information Logs

Information Logs are IrMC objects that contain general information about specific Object Stores. For instance, the Phone Book Information Log contains information such as the type of data formats supported by the Phone Book Object Store, the type of indexing supported, the highest access level supported, and the number of unused entries in the Phone Book.

The Information Log must be supported by all devices that support Level 2, 3 and 4 Information Exchange. Fields within the Information Log may be optional at certain levels. The Information Log must contain the following fields:

2.9.1 Total-Records

Mandatory for Level 3 and 4. The total number of Objects (or records) within the Object Store.

2.9.2 Last-Used-Index

Mandatory if the device supports Level 3 information exchange. This does not apply to Level 4. Indicates the last-used static index within the Object Store. This value indicates that newly added records can be added after this last used index location without risk of overwriting existing records.

2.9.3 Maximum-Records

Mandatory for Level 2, 3 and 4. The maximum number of objects (or records) that can be stored in the Object Store. If there is no particular limit, then an asterisk "*" will be used in place of a number.

2.9.4 IEL (Information Exchange Level)

Mandatory for Level 4. The level of Information Exchange supported by the Object Store. For the sake of backwards compatibility, this parameter is not required for IrMC version 1.0 Level 2 and 3 devices. If the Access-Level parameter is not available, the IAS values should be used to determine the Access Level supported for the Phone Book, Calendar and Message applications. The acceptable values of Access-Level are outlined in a table below:

Value	Description
0x01	Level 1 (Minimum Level) Only
0x02	Level 1 and 2 (Minimum and Access Levels) Only
0x04	Level 1, 2 and 3 (Minimum, Access, Index) Only – implies static index support
0x08	Level 1, 2, and 4 (Minimum, Access, Sync) Only – implies unique index support
0x10	Level 1, 2, 3 and 4 (Minimum, Access, Index, Sync) – implies support of static and unique index

2.9.5 HD (Hard delete)

Mandatory at Level 4. This parameter indicates whether the device supports a distinction between hard and soft deletes. Acceptable values are YES and NO. YES indicates that the object store is capable of making a distinction between hard and soft deletes. NO indicates that the object store treats all deletes the same way.

2.9.6 SAT (Sync-Anchor-Type)

Mandatory at Level 4. This parameter indicates which type of Sync Anchor is supported. Either the device supports a change counter only (“CC”), or a Timestamp only (“TS”) or both (“CT”).

2.9.7 (SAI) Sync-Anchor-Increment

Mandatory if Sync-Anchor-Type is “TS” or “CT”. This parameter indicates whether the timestamp values are guaranteed to increment, i.e., the reference clock can never be set to an earlier time. Acceptable values are “YES”

2.9.8 SAU (Sync-Anchor-Unique)

Mandatory if Sync-Anchor-Type is “TS” or “CT”. This parameter indicates whether the timestamp values are guaranteed to be unique. Acceptable values are “YES” or “NO”.

2.9.9 DID (Database ID)

Mandatory at Level 4. This parameter indicates the database ID that is assigned to the Object Store. Acceptable values are any random string.

2.9.10 X-IRMC-Fields Definition

IrMC Object Stores may support various subsets of field properties, parameters and types defined by Versit or other supported standards. There is a need for an IrMC Client to understand the total fields supported within a specific Server Object Store, as well as the size, i.e., length, of the supported fields without accessing all data records.

Without the ability of an Object Store to provide an overview of all the fields supported in that store, an IrMC Client would have to read all the records, process all the properties, parameters and types supported in all the records and derive a list of field support. This would negatively affect performance, especially for synchronization.

The IrMC Object Store must indicate to an IrMC Client what field properties, parameters and types it supports.

Both the vCard and vCalendar specifications provide for Extension usage - vCard extensions are supported in vCard Version 2.1 in section 2.8.1 and vCalendar extensions are supported in vCalendar Version 1.0 in Section

2.4.1. Following the guidelines in these standards for the creation of the extension property, the extension property for dynamic field mappings usage is to be defined as X-IRMC-FIELDS.

X-IRMC-FIELDS is mandatory for Level 4 support, and is highly recommended for Level 2 and 3.

The rules for the X-IRMC-FIELDS creation can be stated in summary as: If the X-IRMC-FIELDS extension property is supported, it must list for each supported property of the standard format (vCard, vCalendar, vMessage, vNote) a property name followed by square brackets with indexes for the enumerated positional values if applicable, using semicolons as delimiters and followed by a list of supported values for the TYPE parameter. If a size restriction is placed on a supported field element, the size will be represented by an equal sign followed by a numeric value. Each new property within the X-IRMC-FIELDS extension property must be terminated by the use of a colon ":". The end of each line in the X-IRMC-FIELDS definition must contain a line feed as represented by <CRLF>. More specifically, the rules for the creation of the X-IRMC-FIELDS property is as follows:

- The X-IRMC-FIELDS extension property will represent all the supported field properties, property parameters and types utilized within the IrMC Object Store.
- The existence of the X-IRMC-FIELDS extension property in the Information Log for the specified Object Store, indicates the fields supported for the remainder of the processing session.
- Records with differing field support from the fields specified by the X-IRMC-FIELDS property can be ignored.
- If the X-IRMC-FIELDS extension property does not exist, then the fields supported may be any of those supported within the specified Versit standard, or other applicable standard for the Object Store.
- After the tag definition indicating support of X-IRMC-FIELDS, the beginning of the X-IRMC-FIELDS syntax must contain the tag “<Begin>”
- The ending of the X-IRMC-FIELDS definition must contain the tag “<End>”
- Each new property/parameter pairing will exist on individual lines ending with a <*CRLF*>.
- The X-IRMC-FIELDS extension property must contain all the utilized property names which are supported within an Object Store, including mandatory property names as determined by the IrMC Specification.
- All Property Types must start with the syntax TYPE=. All supported property types must be indicated and must use the Field Delimiter character ";" (ASCII decimal 59) to separate types. For example, a vCard Delivery Address which supports the Delivery Address Types DOM and HOME would be represented in an X-IRMC-FIELDS extension as follows: TYPE=DOM;HOME:
- Each new property within the X-IRMC-FIELDS extension property must be terminated by the use of a colon ":".
- Properties which contain positional fields must be represented by enumerated positional values separated by the Field Delimiter character ";" (ASCII decimal 59), and be enclosed in brackets "[" and "]". For example, the Delivery Address Property example in vCard (Section 2.3.1) indicates the following example: ADR;DOM;HOME:P.O. Box 101; Suite 101; 123 Main Street; Any Town; CA; 91921-1234 This example would be represented as a X-IRMC-FIELDS extension as follows:

  X-IRMC-FIELDS:<*CRLF*>
  <Begin><*CRLF*>
  ADR[1;2;3;4;5;6;];TYPE=DOM;HOME:<*CRLF*>
  <End><*CRLF*>
- The length for each field item will be specified by the use of the equal sign “=” followed by a numeric value. For those fields which do not have length restrictions, the size indicator can be omitted. Furthermore, for those devices that store the data in a character set that is different than the character set used for data transmission, the length must be omitted.

Below is an example of the applied rule for X-IRMC-FIELDS support for enabling dynamic field mapping in an IrMC Object Store which supports vCard. The information is carried in the Information Log for the specified Object Store and is represented as follows:

X-IRMC-FIELDS:
<Begin>
Version:
N:=20
UID:=4
ADR[1=20;2;6;7]:
TEL;TYPE=HOME;WORK:

<End>

The field support indicated by this example is broken down as follows:

- The presence of this extension identifies that there are fields which are intended to be dynamically mapped, as indicated by - X-IRMC-FIELDS:
- Each line supports a <CRLF>
- The syntax of the X-IRMC-FIELDS item in the Information log contains a <Begin> and <End> delimiter
- The extension supports a Version Number field, as indicated by -Version:
- The extension supports a Name field, as indicated by - N: with a maximum supported length of 20 bytes indicated by - =20
- The extension supports a UID field as indicated by - UID: with a maximum supported length of 4 bytes indicated by - =4
- The extension supports the Delivery Address property, as indicated by - ADR[1;2;6;7]: and the first positional field element supports the maximum length of 20 as indicated by – [1=20;2;6;7]
- The positional fields supported for this property are indicated by an enumeration of the supported positions:
  - 1; indicates support for Post Office Address
  - 2; indicates support for Extended Address
  - 6; indicates support for Postal Code
  - 7 indicates support for Country

The extension supports the Telephone Property for both HOME and WORK, as indicated by - TEL;TYPE=HOME;WORK:

For more information on X-IRMC usage, pronunciation and the TEL parameter, please see Chapter 7.8.

2.9.11 ICL (Incoming-Call-Log)

Mandatory at Level 4 for devices with Phone Book Object Stores. This indicates whether the device supports an Incoming Call Log for the Phone Book Object Store. It is only present in the Phone Book Information Log. Acceptable values are Y (Yes) and N (NO.)

2.9.12 OCL (Outgoing-Call-Log)

Mandatory at Level 4 for devices with Phone Book Object Stores. This indicates whether the device supports an Outgoing Call Log for the Phone Book Object Store. It is only present in the Phone Book Information log. Acceptable values are Y (Yes) and N (NO.)

2.9.13 MCL (Missed-Call-Log)

Mandatory at Level 4 for devices with Phone Book Object Stores. This indicates whether the device supports a Missed Call Log for the Phone Book Object Store. It is only present in the Phone Book Information log. Acceptable values are Y (Yes) and N (NO.)

2.9.14 MMHL (Missed-Message-History-Log)

Mandatory at Level 4 for devices with Message Object Stores. This indicates whether the device supports a Missed Message History Object for the Message Information Store. It is only present in the Message Information Log. Acceptable values are Y (Yes) and N (NO.)

2.9.15 Sample Phone Book Log

In this example:

- The log object indicates the Total Number of Records in the Phone Book is 4;
- the Last Used Index is 5;
- the Maximum Number of Records that can be stored is unlimited
- the device supports Level 3 information Exchange, and therefore uses static indexing
- the device can not distinguish between hard and soft deletes
- it does not support the Timestamp field in the Change Log.
- It does not support a database ID.
- The X-IRMC-FIELDS property example is explained in Chapter 2.9.10 X-IRMC-Fields Definition.

Finally, the device supports Incoming, Outgoing and Missed Call Logs.

An example of the Information Log for Phone Book is as follows:

Total-Records:4
Last-Used-Index:5
Maximum-Records:*
IEL:3
HD:NO
SAT:CC
DID:*
X-IRMC-FIELDS:
<Begin>
Version:
N:=20
UID:=4
ADR[1=20;2;6;7]:
TEL;TYPE=HOME;WORK:
<End>
ICL:YES
OCL:YES
MCL:YES

In this example:

- The log object indicates the Total Number of Records in the Phone Book is 4;
- the Maximum Number of Records that can be stored is 10
- the device supports Level 4 Information Exchange, and therefore uses unique indexing
- the device supports hard and soft deletes
- it supports the Timestamp field in the Change Log. All timestamps are guaranteed to increment and are guaranteed unique.
- It has a database ID of 3edfeg7898dae.
- The X-IRMC-FIELDS property example is explained in Chapter 2.9.10 X-IRMC-Fields Definition.

Finally, the device does not support the Incoming, Outgoing or Missed Call Logs.

An example of the Information Log for Phone Book is as follows:

Total-Records:4
Maximum-Records:10
IEL:4
HD:YES
SAT:TS
SAI:YES
SAU:YESDID:3edfeg7898dae
X-IRMC-FIELDS:
<Begin>
Version:
N:=20

UID:=4
ADR[1=20;2;6;7]:
TEL;TYPE=HOME;WORK:
<End>
ICL:NO
OCL:NO
MCL:NO

2.9.16 Formal Definition of info.log

<information-log> ::= {

<total-records> *<CRLF>*

[*<last-used-index>* *<CRLF>*] ; only required if static indexing is supported

<maximum-records> *<CRLF>*

[*<information-exchange-level>* *<CRLF>*] ; mandatory for IEL values of 4 or 5

[*<hard-soft-delete-support>* *<CRLF>*] ; mandatory for IEL values of 4 or 5

[*<sync-anchor-type>* *<CRLF>*] ; mandatory for IEL values of 4 or 5

[*<sync-anchor-increment>* *<CRLF>*] ; mandatory if sync-anchor-type is "CT" or "TS"

[*<sync-anchor-unique>* *<CRLF>*] ; mandatory if sync-anchor-type is "CT" or "TS"

[*<databaseID>* *<CRLF>*] ; mandatory for IEL values of 4 or 5

[*<x-irmc-fields-object>*]

[*<object-store-specific-objects>*] ; any other items that are necessary for a particular object store

}*<total-records>* ::= "**Total-Records:**" *<digits-or-not-supported>*

<last-used-index> ::= "**Last-Used-Index:**" *<digits-or-not-supported>* ; mandatory for IEL values of 3 or 5

<maximum-records> ::= "**Maximum-Records:**" *<digits-or-empty>* | "*****"

<information-exchange-level> ::= "**IEL:**" {"**1**" | "**2**" | "**3**" | "**4**" | "**5**" }

<hard-soft-delete-support> ::= "**HD:**" "**YES**" | "**NO**"

<sync-anchor-type> ::= "**SAT:**" **"CC**" | **"TS**" | "**CT**"

<sync-anchor-increment> ::= "**SAI:" YES:**" | **"NO**"

<sync-anchor-unique> ::= "**SAU:" YES:**" | **"NO**"*<databaseID>* ::= "**DID:**" *<default-string-or-not-supported>*

<x-irmc-fields-object> ::= {

"**<Begin>**" *<CRLF>*

<x-irmc-property-object>$^{+}$

"**<End>**" *<CRLF>* }

<x-irmc-property-object> ::= {

<x-irmc-property-name>

["**[**" *<x-irmc-positional- list>* "**]**"]

["**;TYPE=**" *<x-irmc-type-name-list>*]

"**:**"

["**=**" *<common-digit>*$^{+}$]

<CRLF }

<x-irmc-property-name> ::= ; any prop or field name defined in the object content spec, e.g., vCard name

<x-irmc-type-name-list> ::= *<x-irmc-type-name>* ["**;**" *<x-irmc-type-name>*]* ;

<x-irmc-type-name> ::= ; any type name defined in the object content spec, e.g., vCard *knowntype*

<x-irmc-positional-list> ::= *<x-irmc-position>* ["**;**" *<x-irmc-position>*]*

<x-irmc-position> ::= *<common-digit>*$^{+}$ ["**=**" *<common-digit>*$^{+}$]

3. Data Transmission Services

IrMC Objects are accessible both through a connection-oriented service and a connectionless service. Establishment of a connection is required 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 the OBEX Specification. If Connection Oriented Services are implemented in the device, Level 1, 2, 3 and 4 Information Exchange must support these services.

Connection Oriented services are located by using the appropriate LSAP selector for establishing the connection. See Section 12.1.1 for information regarding the use of LSAP selectors for IrMC services.

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 OBEX Connect or OBEX Disconnect procedures. Devices implementing the connectionless service thus only need to support the OBEX PUT operation. Devices that implement connectionless services must provide Level 1 Information Exchange.

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 IrMC 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 octets. The Segmentation and Re-assembly (SAR) byte must exist in all Ultra packets. By definition OBEX requires SAR even if the data will fit in one 60 octet packet. The OBEX PID of 0x01 implies an SAR byte. There is no IrDA IrMC specific limitation on the number of frames in the sequence, i.e., up to 15 fragments (900 octets) can be sent.

To guarantee a successful transfer of data over Ultra, the OBEX object should be kept to less than or equal to 255 bytes. This includes the OBEX name header etc. The size should be minimized by only sending the OBEX name. The OBEX length, time and type fields should be omitted. The data should be contained in one OBEX packet. i.e., the first and only frame contains the PUT_LAST opcode.

The 255 byte or less limit, was added as an errata on the original document, so the limit is only a recommendations.

IrMC implementations are not required to receive OBEX packets larger than 255 bytes. Possible ways that IrMC implementations can deal with Ultra OBEX packets larger than 255 bytes is as follows:

1. Ignore the packet completely
2. Only accept the first 255 bytes
3. Accept the whole frame.

Normal IrLAP 1.1 MAC rules apply to IrDA IrMC 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 shall never include more than one IrMC Object. Multiple IrMC Objects shall not be combined in one 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.

3.3 Connection vs. Connectionless Switching

A client supporting both connection less (Ultra) and connection oriented who wants to do a level 1 PUT to a server may adhere to the following guidelines:

1. Discovery. The device should follow normal discovery procedures. If a correct discovery response is not received from the server after running the discovery process a third time, the client shall assume that there is no connection oriented server present, wait at least 500 ms and send the object connection less (Ultra). According to the media access rules there shall be delay before the sending of the ultra frame, this will mean that the total delay from the last XID frame to the first Ultra frame will be about 1 s.
2. If the server returns a proper discovery response the client will try to connect to the OBEX server. If no OBEX server can be found, the object is not sent using connectionless either. This is because the client assumes that server does not support OBEX connectionless if it also does not supported connection oriented. A server implementing OBEX and supporting connection oriented services is required to support OBEX connection-oriented.
3. Connection. A connection is established to the server. The IAS is then queried for the OBEX server, if an OBEX server is not present, the client disconnects from the server. If an OBEX server is present a connection is established and the object is transferred.

Example: The client receives a proper discovery response to its discovery command from the server side. It then sends a SNRM command frame to the server. If the server decides to accept, it sends an UA frame as response to the SNRM command. The client will then connect to the IAS LSAP of the server, searching for a suiting IAS entry. If such an entry is not found it means that the server does not support this service. The client should then disconnect from the IAS LSAP, followed by a disconnect from the physical link.

4. OBEX Information Access and Indexing

4.1 Indexing

OBEX GET and PUT operations are used for exchanging IrMC Objects. The access schemes for each of these object types is similar. With the exception of Level 1 Information Exchange, whereby the IrMC Objects are pushed into a device inbox, the object names passed to OBEX GET and PUT operations shall always include the path information.

IrMC Devices are required to store items with an object identifier. This identifier is used in Level 2, 3 and 4 Information Exchange. This object identifier is referred to as an "index". There are two types of indexing supported by IrMC Devices: static and unique.

4.1.1 Static Indexing

In this mode, the IrMC device defines a finite range of available indices starting at 0, e.g., 0 to 65535. Each IrMC Object in a given Object Store is assigned one of these values. Index values can be assigned by the IrMC Client or the IrMC Server. When objects are deleted, the index values can be reused. During an OBEX session between two IrMC devices, the index entry positions will be preserved for that session, up until the time the OBEX session is discontinued.

4.1.2 Unique Indexing

In this mode, the IrMC device assigns to each Object a locally unique, non-reusable identifier, or LUID. Unlike Static Indices, which can be assigned by the IrMC Client, LUIDs are assigned by the IrMC Server. They are unique per device and per application.

The Unique Indexing mode defines an IrMC specific property called X-IRMC-LUID. The X-IRMC-LUID property shall contain a locally unique identifier, the LUID.

The LUID must only include alphanumeric characters that can be represented by 7-bit ASCII. LUID "0", i.e., "zero", is a valid LUID but is reserved for special use by each content format. It is recommended that the name be kept short because the entire name, including path, is contained in the OBEX Header, and that header must fit in the first OBEX packet.

Devices which implement Level 4 Information Exchange must support Unique Indexing.

If the Object Store supports Change Counter Sync Anchors, then to protect changes made by another client (if they are accessing the database at the same time), the client submits a maximum expected change counter with each modification or delete. When receiving the request the server compares the submitted maximum expected Change Counter with either the Change Counter for the affected object (if the Change Counter is stored in each object) or with the current Change Counter (if the Change Counter is only stored as a single object) if the maximum expected Change Counter is smaller the request is denied by returning a failure code "conflict".

4.1.2.1 Formal Definition of X-IRMC-LUID

<*x-irmc-luid-object*> ::= "**X-IRMC-LUID:**" <*luid*>

<*luid*> ::= <*alphanumeric-char*>$^{+}$; must be no more than 50 Unicode characters (100 octets) in length

4.2 Read Access

For reading IrMC Objects and Object Stores an OBEX GET Request/Response is used. Each GET Request carries the NAME of the object to be read. The name consists of the pathname, the name of the object and the extension of the NAME indicates the format of the object.

> **Static Index Note**: When performing a Level 3 GET operation, NAME shall be set to the static index assigned to the object, e.g., telecom/pb/0.vcf for the vCard object at index 0.
>
> **Unique Index Note**: When performing a Level 4 GET operation, NAME shall be set to the LUID assigned to the object, e.g., telecom/pb/luid/12ac3f.vcf for the vCard object with LUID 12ac3f.

An IrMC Server 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. In some cases, the BODY of the response may contain data.

SUCCESS

If the requested object was found and read successfully, the GET response carries a "success" code 0xA0. The contents of the requested IrMC Object or Object Store are included in the BODY part of the response.

> **Static Index Note**: When static indexing is supported, the organization of the Object Store in the Level 2 <object-store-stream-object> must reflect the organization of the single <object-store-indexed-object> entries. During an OBEX session, the objects associated with particular index values will be preserved for that session, up until the time the OBEX session is discontinued. After that time, they may be assigned to new index values. In addition, empty entries must be present in the <object-store-stream-object>. Finally, objects with restricted access should be listed as empty entries.
>
> If the device supports both Static and Unique Indexing, and the Level 2 GET Request did not contain the IRMC-SYNC Target Header, then the Level 2 GET Response shall adhere to the instructions in the previous paragraph.
>
> **Unique Index Note**: When performing a Level 2 GET of the entire Object Store, the IrMC Server shall respond with <object-store-stream-object>. The order of entries in the <object-store-stream-object> is implementation dependent. The X-IRMC-LUID property is mandatory for each object in <object-store-stream-object>. It should not be included in Level 4 GET Responses since the LUID can be derived from the Object name. Restricted entries shall not be included in the <object-store-stream-object>. Finally, only used (non-empty) entries shall be present in <object-store-stream-object>.
>
> If the device supports both Static and Unique Indexing, and the Level 2 GET Request contains the IRMC-SYNC Target Header, then the Level 2 GET Response shall adhere to the instructions in the previous paragraph.
>
> Use of the IRMC-SYNC Target Header implies that the Client believes it is communicating with a unique indexed Object Store and that the Server Object Store supports unique indexing.

EMPTY ENTRY

If the requested object was found, but it's contents were empty, the GET response carries a "success" response code 0xA0 with no body in the response.

RESTRICTED ACCESS

Some objects may have a limited read access. If an unauthorized GET request is made to one of these objects, the GET response is to indicate failure with failure code "unauthorized" 0xC1. No body is included in the response.

INDEX NOT FOUND

If a read request is made to a non-existent object, the response is to carry a failure code of "not found" 0xC4. No BODY is included in the response. For more information about the OBEX GET, please refer to [OBEX].

Phone Book Read Example (Static): Consecutive read requests to a Phone Book Object Store with static range of indices [0...7]. The Object Store contains five vCard 2.1 Objects, the third object is empty and the IrMC Client is not authorized to access the fifth object. The following results would be returned:

	Object Requested in GET		**Result**	**Object Returned in GET Response**
	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]

*Note: telecom/pb/ is the path of the individual phone book object. vcf is the extension for vCard 2.1 objects. The objects are named **n**.vcf where **n** is the static index of the Object.*

Phone Book Read Example (Unique): Consecutive read requests to a Phone Book Object Store with unique indices. The Object Store contains three vCard 2.1 Objects and the IrMC Client is not authorized to access the fourth object. The following results would be returned:

	Object Requested in GET		**Result**	**Object Returned in GET Response**
	telecom/pb/luid/0.vcf	- >	Success	[0xA0] + contents of the entry
	telecom/pb/luid/12ac3.vcf	- >	Success	[0xA0] + contents of the entry
	telecom/pb/luid/3a465.vcf	- >	Success	[0xA0] + contents of the entry
[limited access]	telecom/pb/luid/25432.vcf	- >	Unauthorized	[0xC1]
	telecom/pb/luid/25433.vc	- >	Not Found	[0xC4]

*Note: telecom/pb/luid is the path of the individual phone book object. vcf is the extension for vCard 2.1 objects. The objects are named **LUID**.vcf where **LUID** is the locally unique identifier of the Object.*

4.3 Write Access

For writing IrMC Objects and Object Stores, an OBEX PUT Request/Response is used. Each PUT Request carries the NAME, a LENGTH, an APPLICATION COMMAND HEADER and the BODY of the object to be written. The name indicates the name of the IrMC Object to which the write operation is to be applied. The name consists of the pathname, the name of the object and the extension of the NAME indicates the format of the object.

The length field in the PUT-request indicates the length of the object to be written as specified in [OBEX], and is optional. The contents of the IrMC Object or Object Store are included in the BODY of the request.

The application command header in the PUT request carries application information about the write access command. This header is mandatory for level 4 devices, optional for level 1, 2 and 3 devices. The application command header carries information about hard/soft deletes. It also carries information about the 'maximum-expected-change-counter' if the device supports Change Counter Sync Anchors. The application command header syntax follows the OBEX Application Response Header syntax which is specified in [OBEX-ARHDR].

Maximum Expected Change Counter (only Modifies and Deletes):

AppCommand-Tag: 0x11

AppCommand-Value-Length: 'The length of the value field'

AppCommand-Value: The Maximum Expected Change Counter represented in 7 bit ASCII, e.g., "5678"

Hard delete flag (only at Deletes)

AppCommand-Tag: 0x12:

AppCommand-Value-Length: 0

AppResponse-Value: - empty -

OBEX PUT can be used to add or modify objects. Objects may be deleted on the IrMC Server by sending an OBEX PUT Request without a BODY.

> **Static Index Note**: When performing a Level 3 Modify or Delete operation, Name shall be set to the static index assigned to the object, e.g., telecom/pb/0.vcf for the vCard object at index 0.
>
> When performing a Level 3 Add, the IrMC Client must assign a static index. The IRMC Server must accept that index assignment. The only exception is if it is a restricted entry, in which case the client is not allowed to modify the entry.
>
> When performing a Level 2 PUT, internal organization of the object entries may have significance to the application user, therefore no unauthorized changes in the entry order shall be made.
>
> **Unique Index Note**: When performing a Level 4 Delete or Modify operation, Name shall be set to the LUID assigned to the object, e.g., telecom/pb/luid/12ac3f.vcf for the vCard object with LUID 12ac3f.
>
> When performing a Level 4 Add operation, Name shall be set to NULL, e.g., telecom/pb/luid/.vcf.

An IrMC Server 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 IrMC Object was written successfully to the Object Store, the response carries a "success" code 0xA0.

> **Static Index Note**: For Level 2 PUT operations on Static Index objects, it is up to the implementation whether all empty locations are indicated in the <object-store-stream-object>. This is recommended in order to allow straightforward access to single objects based on the information given by a read-all operation.
>
> When performing a Level 2 PUT, the organization of the Object Store in the Level 2 <object-store-stream-object> must reflect the organization of the single <object-store-indexed-object> entries. Accordingly, any changes in any of the indexed entries must be updated for future reads or writes of the stream object.
>
> If the device supports Database Ids, when performing a Level 2 PUT of the entire Object Store, a new database ID must be assigned to the Object Store by the Server.
>
> **Unique Index Note:** Upon a successful Level 4 Add operation, the IrMC Server response includes the newly assigned Unique Index value and the Sync Anchor using the Unique Response Format below.
>
> Upon a successful Level 4 Modify operation, the IrMC Server responds with the LUID of the entry that was modified and the Sync Anchor using the Unique Response Format below
>
> Upon a successful Level 4 Delete operation, the IrMC Server responds with the LUID of the entry that was deleted and the Sync Anchor using the Unique Response Format below.
>
> When performing a Level 2 PUT of the entire Object Store, the IrMC server shall
>
> (1) delete all objects that are currently in the Object Store
>
> (2) Clear the Change Log and set the Change Counter, if supported, to 0
>
> (3) Assign a new database ID to the Object Store
>
> (4) Add each entry from the <*object-store-stream-object*> to the new Object Store. Each object must be assigned a new LUID.
>
> **Unique Response Format:** When the IrMC Server responds to Level 4 ADD, MODIFY or DELETE operations it must respond with the LUID of the object and the Sync Anchor(s) associated with the operation. The LUID and the Sync Anchor must be included in an OBEX AppResponse Header in the Obex PUT Response. The OBEX AppResponse Header is defined in [OBEX-ARHDR].

LUID:

AppResponse-Tag: 0x01

AppResponse-Value-Length: 'The length of the value field'

AppResponse-Value: The LUID represented in 7 bit ASCII, e.g., "3A4C5345"

Change Counter:

AppResponse-Tag: 0x02

AppResponse-Value-Length: 'The length of the value field'

AppResponse-Value:

The Change Counter represented in 7 bit ASCII, e.g., "5678" if the device supports Change Counters as Sync Anchors.

Timestamp:

AppResponse-Tag: 0x03

AppResponse-Value-Length: 'The length of the value field'

AppResponse-Value:

The Timestamp of the last modification in <common-date> format, "e.g., 19980302T100003Z" if the device supports Timestamps as Sync Anchors.

EMPTY ENTRY

If the OBEX PUT request has no body or the body header is empty, the write action is regarded as a DELETE of the IrMC Object or Object Store. Any previous information is cleared.

Note: Some devices make a distinction between erasing an IrMC Object and creating an empty IrMC Object. In this case, an OBEX PUT request of an IrMC Object with an empty body header (an empty body header is 3 bytes. 1 byte for the Type plus a 2 byte Length field.) implies the creation of an empty entry; an OBEX PUT request of an IrMC Object with no body implies the IrMC Object is to be erased.

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 IrMC Server rejects the deletion, a response indicating failure with failure code "unauthorized" 0xC1 shall be returned. No body is included as part of the response.

If the entry delete is accepted by the IrMC Server and is successful, the response carries a "success" response code 0xA0 with no body is included as part of the response.

RESTRICTED ACCESS

Some IrMC Objects and Object Stores may have limited write access. If an unauthorized PUT request is made to one of these objects, the PUT response is to indicate failure with failure code "unauthorized" 0xC1. No body is included as part of the response.

INDEX NOT FOUND

If a write request is made to a non-existent IrMC Object, the response is to carry a failure code of "not found" 0xC4. No BODY is included as part of the response. For more information about the OBEX PUT please refer to [OBEX].

DATABASE LOCKED

If a write request is made to a locked Object Store, the response is to carry a failure code of "database locked" 0xE0. No BODY is included as part of the response. For more information about OBEX PUT refer to [OBEX].

DATABASE FULL

If a write request is made to an Object Store that is out of storage space, the response is to carry a failure code of "database full" 0xE1. No BODY is included as part of the response. For more information about OBEX PUT refer to [OBEX].

CONFLICT
If the device supports Change Counter Anchors, and if the Change Counter for the affected entry or the current Change Counter is bigger than submitted <maximum-expected-change-counter> the OBEX server will response will carry a failure code of "conflict" 0x49.

Calendar Write Example (Static): Five consecutive write operations to a statically indexed Calendar Object Store with static range of indices [0...4]. The first object and third objects are rejected and reported as "unauthorized." For more information on restricted access, please see section 4.3 of this specification. The following results would be returned:

	Object Written in PUT				**Result**	**PUT Response**
[limited access]	telecom/cal/0.vcs	&	no body	- >	Unauthorized	[0xC1]
	telecom/cal/1.vcs	&	vCalendar body	- >	Success	[0xA0]
	telecom/cal/2.vcs	&	no body	- >	Success	[0xA0]
[delete rejected]	telecom/cal/3.vcs	&	no body	- >	Unauthorized	[0xC1]
	telecom/cal/4.vcs	&	vCalendar body	- >	Not Found	[0xC4]

*Note: telecom/cal/ is the path of the individual calendar object. vcs is the extension for vCalendar 1.0 objects. The objects are named **n**.vcs where **n** is the static index of the Object.*

Calendar Write Example (Unique): The following example shows how to add, modify and delete objects in the Calendar Object Store using Unique Indexing and Change Counter Sync Anchors.

[addition of a new object]
Client request: PUT telecom/cal/luid/.vcs & vCalendar body
Server response: Success [0xA0], AppResponse Header [0x01,0x03,"801", 0x02,0x03,"867"]

Note: telecom/cal/luid/ is the path where the client wants to add a new calendar object. vcs is the extension for vCalendar 1.0 objects. Note that the object does not have a LUID before it has been added to the data store. The object was given a LUID, 801, by the IrMC Server. The addition resulted in the change counter becoming 867.

[modification of an existing object]
Client request: PUT telecom/cal/luid/801.vcs & vCalendar body
App Command Header [0x11,0x03,"867"]
Server response: Success [0xA0], AppResponse Header [0x01,0x03,"801", 0x02,0x

Note: telecom/cal/luid is the path of the individual uniquely indexed calendar object. vcs is the extension for vCalendar 1.0 objects. The object name is the LUID, 801. The modification resulted in the change counter becoming 868.

[deletion of an existing object]
Client request: PUT telecom/cal/luid/801.vcs & no body
App Command Header [0x11,0x03,"868", 0x12,0x00]
Server response: Success [0xA0], AppResponse Header [0x01,0x03,"801", 0x02,0x03,"869"]

Note: telecom/cal/uid is the path of the individual uniquely indexed calendar object. vcs is the extension for vCalendar 1.0 objects. The object name is the LUID, 801. The deletion resulted in the change counter becoming 869. The delete was a hard delete.

The Unauthorized [0xC1], Not Found [0xC4], Database Locked [0xE0], Database Full [0xE1] and Conflict Response [0x49] response codes are used in the same way as for static indexing.

4.4 Information Access Examples

4.4.1 Ultra Write

Two devices with limited connection establishment capabilities. Device1 pushes a phone book object to Device2 using the Ultra protocol. As shown in the Figure 4-1 no connection establishment is required, the OBEX PUT command is used to originate the data exchange and all user data is carried in unnumbered frames. No acknowledgment of the received information is sent or expected.

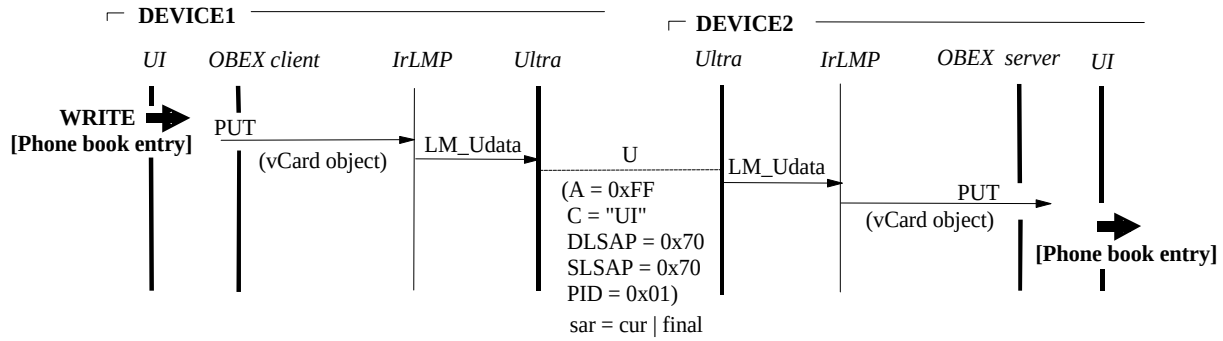


Figure 4-1 Object push with Ultra

4.4.2 Connection Oriented Read

Two devices which support Level 2 Information Exchange of Calendar Object Stores. Device1 requests a Calendar Object Store read operation as shown in the Figure 4-2. 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 Object Store stream. GET-request is confirmed by Device2 with a GET-response which carries the requested Calendar Object Store stream. All user data is transmitted in IrLAP information frames. Once the object read is complete the connection is disconnected with OBEX DISCONNECT-command.

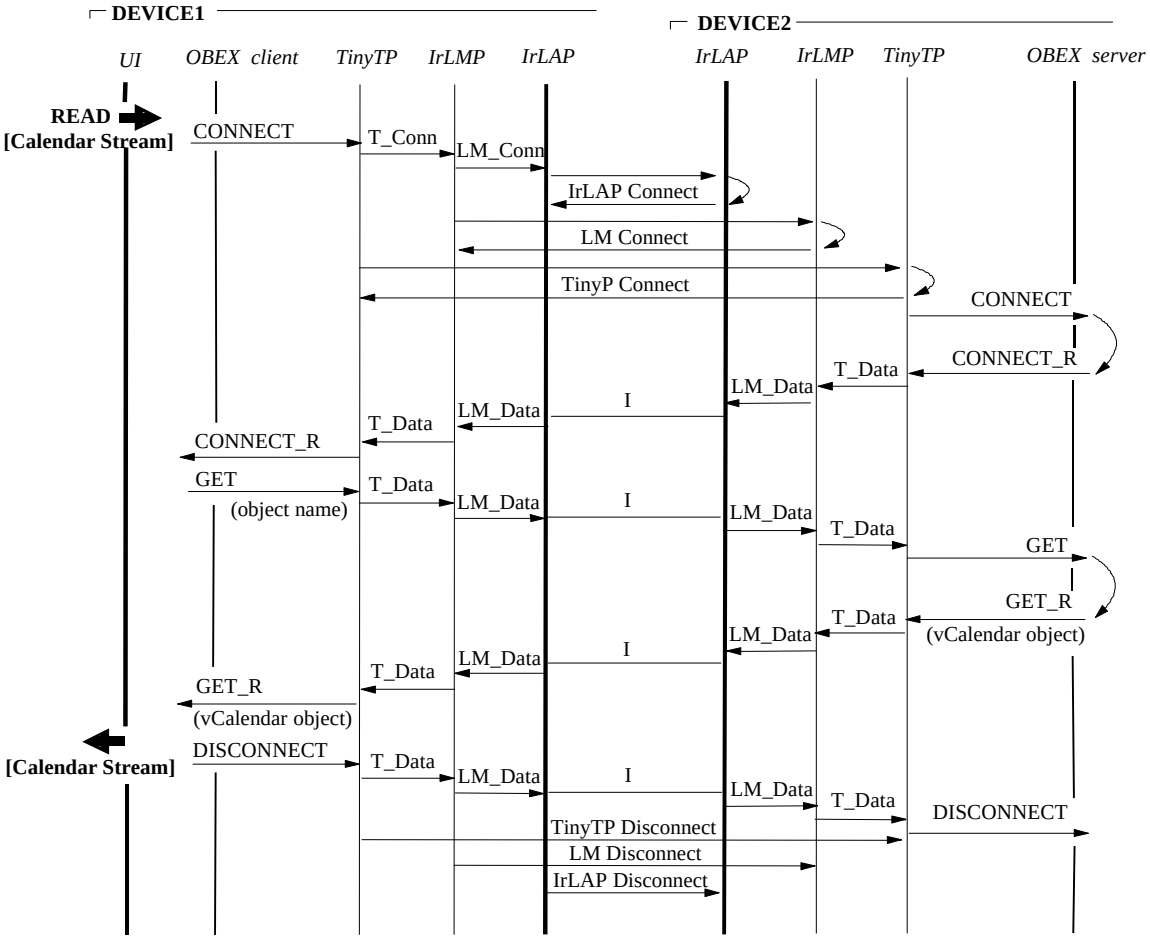


Figure 4-2 Object read

4.5 Real Time Clock

The Real Time Clock Object contains the current date and time of the device. A Client can read and write the value of this object. The OBEX return values are the same as in the case of other objects (phonebook or calendar items) read through IrOBEX. Attempts to write the Real Time Clock can be challenged.

WRITE DATE EXAMPLE

OBEX_PUT(NAME = “telecom/rtc.txt”, BODY = <*common-date*>)

READ DATE EXAMPLE

<*common-date*> OBEX_GET(NAME = “telecom/rtc.txt”)

4.5.1 Formal Definition of Real Time Clock

<*rtc-object-name*> ::= “**telecom/rtc.txt**”

<*rtc-object*> ::= <*common-date*>

4.6 Restricted Access

Read/write access to IrMC objects may be restricted depending on the device implementation and the information to be accessed. OBEX provides an Authentication Protocol (refer to [OBEX-AUTH]) which can be used in one of two ways:

1. The client can do a regular Inbox connect with no target header and no authentication, but if the client does any OBEX requests to level 2 or 3 the OBEX Server can (it does not have to) respond with UNAUTHORIZED and AUTHENTICATE_CHALLENGE to request a PIN.
2. The Client can do an OBEX Connect with target set to "IRMC-SYNC ". The server can (it does not have to) respond with UNAUTHORIZED and AUTHENTICATE_CHALLENGE to request a PIN. The connection will be given a Connection ID by the server. The connection ID header is required in the first OBEX packet of each operation for this connection. Use of the IRMC-SYNC Target Header implies that the Client believes it is communicating with a unique indexed Object Store and that the Server Object Store supports unique indexing.

4.7 Avoiding "Race Condition"

Simultaneous IrMC Object 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 be 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.

4.8 Line Termination Characters

Some objects transferred with OBEX will include <*CRLF*> termination characters, e.g., 0x0D, 0x0A. An alternative termination is simply <*line-feed*>. All IrMC Devices must be able to receive both types of line termination. However, all IrMC devices must always transmit the full <*CRLF*> termination. This applies to all objects within the IrMC specification.

5. Synchronization

5.1 Data Synchronization Overview

In its simplest form, synchronization provides a means to compare two Object Stores, analyze their differences and then modify each Object Store so that the two are identical. In the context of IrMC, a synchronization operation provides a means to synchronize data on two IrMC Devices. Furthermore, a synchronization operation may also be able to sync an Object Store on an IrMC device with an Object Store on a desktop PIM. It is not necessary that the IrMC Object Store and the desktop PIM Object Store use identical Object formats. For purposes of this specification, a Sync Engine is assumed to reside on a PC and it emulates an IrMC Device. The descriptions in this Chapter assume that an IrMC Device (IrMC Server), e.g., a mobile phone, will be synchronized with a PC-PIM through a PC-Based sync engine (IrMC Client).

For synchronizing similar type Object Stores on different devices, information about any changes, additions or deletions in the Object Store is to be stored in a Change Log <*object-store-change-log-object*>. Starting from the latest modification, the Change Log lists in chronological order all the changed objects in a particular Object Store.

Synchronization is most effective when there are means to uniquely identify each Object in the Object Store. Because of this, IrMC Devices that wish to support synchronization must support Level 4 Information Exchange and must implement Unique Indexing. See Chapter 4.1.2 - OBEX Information Access and Indexing.

NOTE: Sync Engines should implement client support for Level 3 (Static Indexing) in addition to the mandatory Level 4 (unique indexing) if they intend to sync to IrMC 1.0 devices/servers

There are two basic types of synchronization: Slow Sync and Fast Sync. There are variants, but this Chapter concentrates on these two types.

A “Slow Sync” occurs when two devices synchronize for the first time, or when the changes to one or both of the device Object Stores is to numerous to process efficiently. A typical “Slow Sync” operation proceeds as follows:

- The IrMC Client (the Sync Engine) performs a Level 2 GET of the entire Object Store on the Device.
- The Sync Engine performs analysis and processing of the data that is to be synchronized.
- The Sync Engine writes any changes to the IrMC Device using Level 4 (or Level 2) Information Exchange.

On subsequent synchronizations, the IrMC device can provide a Change Log that describes only the changes that have occurred in the Object Store. This enables “Fast Sync”. A typical “Fast Sync” operation proceeds as follows:

- The IrMC Client (the Sync Engine) requests a Change Log consisting of all Objects that have changed since Sync Anchor X. X is the value of the Sync Anchor saved by the Sync Engine at the time of its last synchronization with the IrMC Device.
- The Sync Engine requests each of the Objects listed in the Change Log using a Level 4 GET.
- The Sync Engine performs analysis and processing of the data that is to be synchronized.
- The Sync Engine writes any changes to the IrMC Device using Level 4 PUT.

A more detailed explanation of Slow and Fast Sync, along with examples, is provided later in this Chapter.

5.2 Sync Anchors

There are two types of Sync Anchors which can be used by IrMC implementations – Change Counters and Timestamps. IrMC Clients must support both types of anchors. IrMC Server Object Stores must implement at least one of the anchors. The ideal implementation would include both. If that is not possible, then a Change Counter is preferred. For some Object Stores, a Change Counter is not practical so a Timestamp anchor can be used.

5.2.1 Change Counters

One choice for Sync Anchor is the Change Counter. IrMC 1.1 recommends the use of a Change Counter, as opposed to a timestamp, to alleviate problems that can occur when synchronizing with timestamp anchored entries. The Change Counter is a 32-bit unsigned integer holding values from 0 to 4294967295. The Change Counter is set to 0 each time the IrMC device is reset and each time an Object Store is replaced in a Level 2 PUT

operation. The Change Counter is increased by 1 each time any change is made to an Object Store. A change is defined as an Add, Modify or Delete operation on a single object within the Object Store.

A last Change Counter used by an Object Store can be requested by an IrMC Client by issuing an OBEX GET operation similar to the following: GET telecom/XX/luid/cc.log where XX is the name of the Object Store. For instance, GET telecom/pb/luid/cc.log returns an object containing the last used Change Counter object of the Phone Book application.

5.2.2 Timestamps

IrMC also allows the use of Timestamps as Sync Anchors. Specifically, Timestamps represent the time that the modification or deletion occurred for the particular object. Timestamps are represented in <common-date> format, e.g., Jan 3, 1999 1:05:16 PM UTC would be represented as 19990103130516Z. Since there are a few variables affecting timestamps, there are three properties in the information log which must be present if timestamps are used. See Chapter 2.9.

It is highly recommended that all timestamps be guaranteed to have a value greater than all previously recorded timestamps. It is possible for timestamps to have a value less than previously recorded timestamps. For instance, if the device automatically adjusts the time for different time zones when traveling, if the user changes the time. The easiest way to achieve this is to use UTC time as the timebase AND to set the time automatically through a network. The Information Log contains a property called <sync-anchor-increment> which specifies whether this Object Store guarantees that all timestamps will have values greater than previous timestamps. Note that basing times on UTC does not guarantee that all values will increment. The time must also be set automatically by the network for this to be true.

It is highly recommended that timestamps be guaranteed unique values, i.e., two records can not be modified at precisely the same time. It some cases this is not practical. The Information Log contains a property called <sync-anchor-unique> which specifies whether this Object Store guarantees unique values.

Please note that when using Timestamps, the database must be locked when an OBEX connection is established using the IRMC-SYNC Target Header.

5.3 Database IDs

When the IrMC Device creates an Object Store, it also creates a Database ID which is associated with the Object Store. The Database ID is a random string. The Database ID is used by a Sync Engine to determine if the Object Store has been reset, or replaced, since the last synchronization. As such, a new Database ID is generated whenever the device is reset, or upon a Level 2 PUT to the device. In addition, if the device uses a Change Counter Sync Anchor, and the Change Counter rolls over from 4294967295 to 0, this must be treated like a device reset and a new Database ID must be generated. Since a different Database ID must be generated each time the Object Store is created, the Database ID must be unique.

The Database ID is returned in the second line of every Transmitted Change Log. It is also stored in the Object Stores Information Log to provide the same reset detection for Level 2 and 3 Information Exchange.

5.4 Hard and Soft Deletes

A Sync client (e.g., a PC Based Sync Engine) may delete an object in the IrMC Server for two reasons: because the user has deleted it from the PC; or to make room for other objects in the device. If synchronization were to occur and the object were deleted the stage has been set for future problems. For instance, if no distinction can be made between these two types of deletes the a second sync client synchronizing to the IrMC device will have no way of knowing if the deleted object was actually removed or just deleted to make room for other objects.

Similarly, an end user can delete a record in a device for two reasons: because the record is no longer required and the user wants to delete it; or the user wants to delete the record to make more room on a memory-constrained device. Any Sync Client will be unable to ascertain whether the IrMC device deleted the object because the user wanted to remove it permanently, or because the user was trying to free up memory for other objects.

To solve these problems, an IrMC device may support the concept of Hard and Soft Deletes. When the sync client deletes objects it either does a soft delete (to make room for other objects) or a hard delete (removes the object and implies that it should be permanently removed from any synchronizing clients.)

5.5 Change Log

The Change Log is used in Level 4 Information Exchange as a tool for synchronization. It provides a simple way for a Sync Client to obtain the device serial number, database ID, maximum number of records and total records used.

It also provides a list of the newest changes that have been made to a particular Object Store since a particular point in time. The Change Log is chronological: newest changes are listed first, oldest changes are listed last. Each entry in the Stored Change Log corresponds to a change to an Object in an Object Store. Each record contains a Modified Action Property, indicating that the object was Modified or Deleted. Each Change Object in the Change Log consists of:

- **Action**: The Modified Action Property. M=Modified, D= Deleted or H=Hard Deleted. Only one of these can be supported on a per record basis in the Change Log.
- **Change Counter**: The Change Counter for the change. This field is optional but recommended. If this field is not present, then the Timestamp field must be present.
- **Timestamp**: The Timestamp for the change. This field is optional. If this field is not present, then the Change Counter field must be present.
- **LUID**: The LUID form of the name of the modified Object, e.g., “13a4b.vcf”

NOTE: It is not always necessary to transmit the entire change log. To avoid confusion, Stored Change Log refers to the Change Log stored on the IrMC Device. Transmitted Change Log refers to the Change Log transmitted to the IrMC Client The device doesn’t actually have to Store the Stored Change Log, it can build the Transmitted Change Log dynamically.

A Change Log can be requested for each of the following Object Stores: Phone Book, Calendar, Message and Note. The Change Log will hold <n> entries (<n> is implementation specific). If new entries are added and the Stored Change Log is full then the oldest entry will be deleted. Entry 1 holds the latest (newest) change. Entry <n> holds the oldest change recorded in the change log.

The Stored Change Log should be large enough to hold all changes between syncs. It should be noted that multiple Sync Engines may synchronize with a single IrMC Device. For example, a mobile phone may be synchronized with a PC at work and a PC at home. In any event, the Stored Change Log should be as large as possible, but not so large that it becomes larger than sending the changed Object Store itself. Of course, if the Transmitted Change Log is built dynamically this is not an issue.

5.5.1 Requesting a Change Log

A Change Log object can be requested by an IrMC Client by issuing an OBEX GET operation similar to the following: **GET telecom/XX/luid/#########.log** where XX is the name of the Object Store, and ######### is the value of a Sync Anchor. For instance if Change Counter Sync Anchors are used, telecom/pb/luid/123.log identifies that the Client would like a Change Log containing all Change Records whose Change Counter is greater than 123. For instance, telecom/pb/luid/19980101T120000Z.log identifies that the Client would like a Change Log containing all Change Records whose timestamp is greater than 12pm on January 1, 1998 UTC. Note that the first time a Change Log is requested by a client, it should specify telecom/XX/luid/0.log and 19000101000000T.log for Change Log with Sync Anchors using Change Counters and Timestamps, respectively. It is up to the Server to determine which type of Sync Anchor is present in the change log request. The simplest way to do this is to search for the letter “T” in the Sync Anchor value.

As stated above, the entire Stored Change Log is not sent to the IrMC Client. When requesting a Transmitted Change Log, a Sync Engine submits its last known Sync Anchor, a value that was stored the last time it synced with the Device. The IrMC Server returns a Transmitted Change Log consisting of a Database ID and those objects whose Sync Anchor is greater than the one requested.

If the Sync Anchor submitted by the Sync Engine is smaller than the Sync Anchor of the oldest record in the Stored Change Log, i.e., Change Objects have been pushed out of the Stored Change Log, then all hard deletes plus a ‘*’ is returned. This situation is usually due to storage space limitations and means that the whole Object Store must be synchronized using Slow Sync techniques.

Also, if the Sync Anchor submitted by the Sync Engine is larger than the Sync Anchor of the newest record in the Stored Change Log then a ‘*’ is returned.

When the Transmitted Change Log is returned the Device Serial Number is on the first line and the Database ID is on the second line in the Change Log. Using this scheme the sync engine can easily and quickly detect if the Device has been reset or if the entire Object Store has been replaced (restored) by a Level 2 PUT since the last successful synchronization.

Note: The Change Log must be empty upon a reset of the device, or after a Level 2 PUT Operation has been performed. In addition, the Change Log must be empty if a Change Counter Sync Anchor rolls over from 4294967295 to zero…an unlikely occurrence.

5.5.2 Change Log Object Definition and Samples

When any change is made to an Object in an Object Store, a Change Record must be added to the Stored Change Log. The first line of the object contains the device serial number. The second line contains the Database ID named <*databaseid*>. There can be 0 or more Change Records. The Change Log Object Format defined as:

<*change-modify*>::=”**M**”

<*change-delete*>::=”**D**”

<*change-hard-delete*>::=”**H**”

<*change-counter*>::= ’ASCII coded 32 bit unsigned integer, 0 to 4294967295’

<*timestamp*> ::= <*common-date*>

<*change-log-entry-x*>::={

{<*change-modify*>|<*change-delete*>|<*change-hard-delete*>}

”**:**” [<*change-counter*>] ; if the change counter is not included, then timestamp must be

”**:**” [<*timestamp*>] ; if the timestamp is not included, then the change counter must be

“**:”**<*luid*>

<*CRLF*>

}

<*change-log-hard-delete-entry-x*>::={

{<*change-hard-delete*>}

”**:**” [<*timestamp*>] ; if the timestamp is not included, then the change counter must be

“**:”**<*luid*>

<*CRLF*>

}

<*change-log-hard-delete-entries*>::=<*change-log—hard-delete-entry*>*

<*change-log-entries*>::=<*change-log-entry*>*

<*change-log-full*>::=<*change-log-hard-delete-entries*>”*****” <*CRLF*>

<*change-log-object-name*>::= “<*change-counter*> “**.**” <*extension*>

<*change-log-object*>::= {

<*device-info-serial-number-field*> <*CRLF*>

<*databaseid*><*CRLF*>

<*total-records*><*CRLF*>

<*maximum-records*><*CRLF*>

{<*change-log-entries*> | <*change-log-full*>} }

Example 1 (Change log with Change Counter Sync Anchors only requested for change counter 45)

```
SN:1218182THD000001-2
DID:dsfhjg3hg4jf
Total-Records:4
Maximum-Records:50
M:46::19970401-080045-40000F192713-0052
D:47::19790327-080045-40000F192713-0053
M:48::19740715-080045-40000F192713-0123
M:49::19981207-080045-40000F192713-2252
```

This example indicates that:

- The Device Serial Number is: 1218182THD000001-2
- The Database ID is: dsfhjg3hg4jf
- The Total Records in the Object Store are 4
- The Maximum Number of records that can be stored in the Object Store is 50
- Record with LUID 19970401-080045-40000F192713-0052 has been modified, and the Change Counter associated with this add is 46. There is no timestamp in the change log.
- Record with LUID 19970401-080045-40000F192713-0053 has been deleted, and the Change Counter associated with this delete operation is 47. There is no timestamp in the change log.
- Record with LUID 19970401-080045-40000F192713-0123 has been modified, and the Change Counter associated with this modification is 48. There is no timestamp in the change log.
- Record with LUID 19970401-080045-40000F192713-2252 has been modified, and the Change Counter associated with this add is 49. There is no timestamp in the change log.

Example 2 (Sync Anchor requested was too old, Stored Log has pushed out the needed Objects)

```
SN:1218182THD000001-2
DID:dsfhjg3hg4jf
Total-Records:4
Maximum-Records:50
*
```

Example 3 (Empty Change Log)

```
SN:1218182THD000001-2
DID:dsfhjg3hg4jf
Total-Records:4
Maximum-Records:50
```

Example 4 (Change Log Sample With Timestamp Sync Anchors Only requested for 19990104T180000)

```
SN:1218182THD000001-2
DID: 03df30423
Total-Records:4
Maximum-Records:50
M::19990104T180000Z:0A456566
M::19990114T180100Z:0FED4101
H::19990222T000320Z:133DEFDE
```

This example indicates that:

- The Device Serial Number is: 1218182THD000001-2
- The Database ID is: 03df30423
- The Total Records in the Object Store are 4
- The Maximum Number of records that can be stored in the Object Store is 50
- Record with LUID 0A456566 has been modified and the time of that change was 19990104T180000Z
- Record with LUID 0FED4101 has been modified and the time of that change was 19990104T180100Z.
- Record with LUID 133DEFDE has been hard deleted and the time it was deleted was 19990222T000320Z.

Example 5 (Change Log Sample With Change Counter and Timestamp Sync Anchors)

```
SN:1218182THD000001-2
DID: 03df30423
Total-Records:4
Maximum-Records:50
M:123:19990104T180000Z:0A456566
M:124:19990114T180100Z:0FED4101
D:126:19990222T000320Z:133DEFDE
```

This example indicates that:

- The Device Serial Number is: 1218182THD000001-2
- The Database ID is: 03df30423
- The Total Records in the Object Store are 4
- The Maximum Number of records that can be stored in the Object Store is 50
- Record 0A456566 has been modified, the Change Counter associated with this add is 123 and the time of the change was 19990104T180000Z.
- Record 0FED4101 has been modified, the Change Counter associated with this modification is 124 and the time of the change was 19990114T1800100Z.
- Record 133DEFDE has been deleted, the Change Counter associated with this add is 126 and the time of the deletion was 19990222T000320Z.

Note that Change Counter 125 is missing. This Change Object indicated a modification of 133DEFDE.vcf, but since that record was deleted in change 126, there was no need to store or transmit the superfluous modification. (see Reduced Change Log section below).

Example 6 (Change log full, with hard delete support)

```
SN:1218182THD000001-2
DID: 03df30423
Total-Records:4
Maximum-Records:50
H:123:0A456566
H:124:0FED4101
*
```

This example indicates that:

- The Device Serial Number is: 1218182THD000001-2
- The Database ID is: 03df30423
- The Total Records in the Object Store are 4
- The Maximum Number of records that can be stored in the Object Store is 50
- Record 0A456566 has been hard deleted, and the Change Counter associated with this hard delete is 123.
- Record 0FED4101 has been hard deleted, and the Change Counter associated with this hard delete is 124.
- Stored change log did not contain all changes since submitted change counter

5.5.3 Reduced Change Log (Optional)

In order to minimize the size of the Stored and Transmitted Change Log, IrMC implementers may choose to use the following reduction scheme. Before adding a Change Object to the Stored Change Log, the Stored Change Log is searched for entries with the same NAME. If such an entry is found it can be deleted from the Stored Change Log.

Although implementation of a reduced Change Log is optional on behalf of the IrMC Server, all IrMC Sync Engine Clients must be capable of understanding such a reduced Change Log when received.

The action of the new entry will be set according to the table below:

Old entry in change log	Action in database	New entry in change log
- none -	Add	Modify
- none -	Modify	Modify
- none -	Delete	Delete
Modify	Modify	Modify
Modify	Delete	Delete

5.6 Sync Protocol

The following table details message exchanges that can occur during a synchronization operation between a Sync Engine and a single Object Store. Synchronization Engines are not restricted to this type of message exchange. It is provided for illustrative purposes only. The examples do not incorporate the hard and soft delete notion (the example device has no support for hard and soft delete distinction). The examples do not incorporate the notion of timestamps as Sync Anchors.

	Sync Engine	Device
1	Get the Change Log for this device	
2		If the device Stored Change Log does not contain all changes since the submitted Sync Anchor or if the submitted Sync Anchor is larger than the Device Sync Anchor then return a Transmitted Change Log with required fields (Serial Number, Database ID, etc.) + '*', i.e., "too many changes". If the Stored Change Log does contain all changes since the submitted Sync Anchor, return the required fields (Serial Number, Database ID, etc.) + Change Objects with Sync Anchors greater than the submitted Sync Anchor. Objects in the Transmitted Change Log are in chronological order with oldest changes first.
3	Retrieve Device Serial Number and Database ID from the Change Log. If the Serial Number and Database ID are known, the Change Log contains anything other than a "*" then go to step 15 (Fast Sync). If the Serial Number is known, but the Change Log contains a "*" or the Database ID has changed then go to Step 9. (Semi-slow sync). Otherwise this is the first time the databases have been synchronized OR the device has been reset. Go to Step 4. (Slow Sync)	
4	**SLOW SYNC BEGINS HERE** If you don't recognize the Serial Number or Database ID then retrieve the Devinfo.txt Object, otherwise go to Step 9.	
5		Return the Device Information Object
6	Get the Information Log	
7		Return the Information Log for the Object Store.
8	Store the static information from the info.log for future use.	
9	**SEMI SLOW SYNC BEGINS HERE** If the device uses Change Counter Sync Anchors, retrieve the current Device Change Counter, otherwise proceed to step 12.	
10		Return the Change Counter for the Object Store.
11	Store the Device Change Counter as the Stored Sync Anchor	
12	Get all entries in the Device Object Store.	

	Sync Engine	Device
13		Return all entries in database.
14	Go to step 19.	
15	**FAST SYNC BEGINS HERE** Process the Change Objects in the Transmitted Change Log. Starting with the oldest changes first. If Transmitted Change Log is empty go to Step 19.	
16	If the Object was an Add or Modify, get it from Device.	
17		Return Object requested by sync engine.
18	Update the Stored Sync Anchor with the Sync Anchor in the current Change Object. If there are more unprocessed Change Objects go to step 15.	
19	Get all changes from the Sync Client Object Store. Perform Synchronization Analysis. Apply any Adds, Modifies or Deletes to the Sync Client Object Store.	
20	Add, modify or delete Objects in Device Object Store.	
21		If Change Counters are used, increase the Device Change Counter for every change to the database. Return LUID and current Sync Anchor for each object add, modify or delete operation that is performed.
22	If change counters are used, and the difference between the returned Device Change Counter and the Stored Sync Anchor is greater than one go to step 1 because a record has been added, modified or deleted while synchronization was proceeding.	
23	Update the Stored Sync Anchor to the returned Sync Anchor. If there are more objects to Add, Modify or Delete in Client Object Store go to step 20.	
24	Finished.	

5.7 Sync Examples

In the following examples, a mobile IrMC Device (IrMC Server) is to synchronize its Object Store with an Object Store on a PC. The PC is the IrMC Client and the host of the Sync Engine and a non-IrMC Object Store, e.g., Microsoft Outlook. It is assumed that the Sync Engine tries to hold the databases in a one-to-one relationship, for example if an object is deleted in the mobile it will also be deleted on the PC. In the following examples, the Stored Change Log on the Mobile Device can only store two Change Objects. Furthermore, the device has chosen to use Change Counters as Sync Anchors. The Device Change Counter Sync Anchor is an integer between 0 and 4294967295. (32 bit integer). The Device can only store 20 entries in its Phone Book.

5.7.1 First Sync (Slow Sync)

The first sync always involves a slow sync. The PC and Mobile have the following Objects in their Object Stores.

PC PIM Object Store	
Record	**UID**
"A"	456
"D"	98567
"E"	10403

Mobile IrMC Object Store	
Object	**LUID**
"A"	567
"B"	7456
"C"	34345

The mobile has the following Stored Change Log:

	Action	Device Change Counter	LUID	Data (for reference only)
Latest Change	Mod	3	34345	"C"
Oldest Change	Mod	2	7456	"B"
	Mod	1	567	"A"

Sync Engine	Obex Command	Obex Response	Device
Get Change Log	GET “telecom/pb/luid/0.log”		
		SN: xyz1243 DID:12345678 Total-Records: 3 Max-Records: 20 *	Return Change Log
Compare serial number, sync engine realizes that this is a new device and starts a new sync job.			
Get Devinfo.txt	GET “telecom/pb/devinfo.txt”		
		devinfo.txt	Returns Devinfo.txt
Get current change counter	GET ‘telecom/pb/luid/cc.log’		
		3	Return current change counter.
Store ‘3’			
Get the info.log	GET ‘telecom/pb/info.log’		
		info.log data	Return the info log
Store the data from the info.log			
Get all entries	GET ‘telecom/pb.vcf’		
		“A”,X-IRMC-LUID=567 ”B”,X-IRMC-LUID=7456 ”C”,X-IRMC-LUID=34345	Return the whole phone book!
Slow sync. “A” already present on both sides. Do Nothing			
Record D needs to be added to device	PUT ‘telecom/pb/luid/.vcf’ “D”		
		OK, LUID=12345, CC=4 using AppResponse Header	Add “D” and send response
Record E needs to be added to device	PUT ‘telecom/pb/luid/.vcf’ “E”		
		OK, LUID=789, CC=5 using AppResponse Header	Add “E” and send response.

And the result is:

PC PIM Object Store	
Record	**UID**
“A”	456
“D”	98567
“E”	10403
“B”	55002
“C”	40603

Mobile IrMC Object Store	
Object	**LUID**
“A”	567
“B”	7456
“C”	34345
“D”	12345
“E”	789

Sync Engine Table	
PC UID	**Mobile LUID**
456	567
98567	12345
10403	789
55002	7456
40603	34345

The mobile device now has the following Stored Change Log:

	Action	**Change Counter**	**LUID**	Data (for reference only)
Latest Change	Mod	5	789	“E”
Oldest Change	Mod	4	12345	“D”
	Mod	3	34345	“C”
	Mod	2	7456	“B”
	Mod	1	567	“A”

5.7.2 Sync Without Changes (Fast Sync)

In this example, it is assumed that the IrMC Object Store has had no changes applied since the last synchronization. Hence, the Transmitted Change Log is empty. The PC PIM Object Store also has had no changes, hence there are no Objects to be Added, Modified or Deleted from the IrMC Object Store as a result of the synchronization.

Sync Engine	Obex Command	Obex Response	Device
Get Change Log	GET "telecom/pb/luid/5.log"		
		SN: xyz1243 DID:12345678 Total-Records: 5 Max-Records: 20 (Empty change log)	Return the change log. It is empty.
Ok we recognize this device! But there are no changes so we are done.			

5.7.3 Sync with Changes That Fit the Change Log (Fast Sync)

One new entry in the mobile ("Jörgen") and one new entry in the PC ("Ericsson").

PC PIM Object Store	
Record	**UID**
"A"	456
"D"	98567
"E"	10403
"B"	55002
"C"	40603
"Ericsson"	**234**

Mobile IrMC Object Store	
Data	**LUID**
"A"	567
"B"	7456
"C"	34345
"D"	12345
"E"	789
"Jörgen"	**899**

Sync Engine Table	
PC UID	**Mobile LUID**
456	567
98567	12345
10403	789
55002	7456
40603	34345

The mobile has the following change log:

	Action	**Change Counter**	**LUID**	Data (for reference only)
Latest Change	Mod	6	899	"Jörgen"
Oldest Change	Mod	5	789	"E"
	Mod	4	12345	"D"
	Mod	3	34345	"C"
	Mod	2	7456	"B"
	Mod	1	567	"A"

Sync Engine	**Obex Command**	**Obex Response**	**Device**
Get Change Log	GET "telecom/pb/luid/5.log"		
		SN: xyz1243 DID:12345678 Total-Records: 6 Max-Records: 20 M:6::899	Return the change log.
Inspect SN. OK, I know you!			
Get the new entry	GET 'telecom/pb/luid/899.vcf'		
		"Jörgen", X-IRMC-LUID=899	Return object requested
Store '6' as the latest change number.			
Add "Ericsson" to phone	PUT 'telecom/pb/luid/.vcf' "Ericsson"		
		OK, LUID=814,CC=7 using AppResponse Header	Add "Ericsson" and send response
Store '7' as the latest change number. We are done.			

It now looks like this:

PC PIM Object Store	
Record	**UID**
“A”	456
“D”	98567
“E”	10403
“B”	55002
“C”	40603
“Ericsson”	234
“Jörgen”	**1**

Mobile IrMC Object Store	
Object	**LUID**
“A”	567
“B”	7456
“C”	34345
“D”	12345
“E”	789
“Jörgen”	899
“Ericsson”	**814**

Sync engine	
PC UID	**Mobile LUID**
456	567
98567	12345
10403	789
55002	7456
40603	34345
234	**814**
1	**899**

The mobile has the following change log:

	Action	Change #	LUID	Data (for reference only)
Latest Change	Mod	7	814	“Ericsson”
Oldest Change	Mod	6	899	“Jörgen”
	Mod	5	789	“E”
	Mod	4	12345	“D”
	Mod	3	34345	“C”
	Mod	2	7456	“B”
	Mod	1	567	“A”

5.7.4 Sync with Full Change Log (Semi Slow Sync)

One deleted (“C”), one modified (“B” to “Lars”) and one added (“Patrik”).

It looks like this:

PC	
Data	**UID**
“A”	456
“D”	98567
“E”	10403
“B”	55002
“C”	40603
“Ericsson”	234
“Jörgen”	1

Mobile	
Data	**UID**
“A”	567
“Lars”	7456
“D”	12345
“E”	789
“Jörgen”	899
“Ericsson”	814
“Patrik”	2

Sync engine	
PC UID	**Mobile UID**
456	567
98567	12345
10403	789
55002	7456
40603	34345
234	814
1	899

The mobile has the following change log:

	Action	Change #	UID	Data (for reference only)
Latest Change	Add	10	2	“Patrik”
Oldest Change	Modify	9	7456	“B” to “Lars”
	Delete	8	34345	“C”
	Mod	7	814	“Ericsson”
	Mod	6	899	“Jörgen”
	Mod	5	789	“E”
	Mod	4	12345	“D”
	Mod	3	34345	“C”
	Mod	2	7456	“B”
	Mod	1	567	“A”

Sync Engine	Obex Command	Obex Response	Device
Get Change Log	GET ‘telecom/pb/luid/7.log’		
		SN: xyz1243 DID:12345678 Total-Records: 7 Max-Records: 20 *	Error! To many changes since ‘7’.
OK, I know you!			
Oh no! Change Log says *. We need to semi slow sync!			
Get current change counter	GET ‘telecom/pb/luid/cc.log’		
		10	Return current change counter.
Temporarily store ‘10’			
Get Entire Database	GET ‘telecom/pb.vcf’		
		“A”, X-IRMC-LUID=567 ”Lars”, X-IRMC-LUID=7456 “D”, X-IRMC-LUID=12345 “E”, X-IRMC-LUID=789 “Jörgen”, X-IRMC-LUID=899 “Ericsson”, X-IRMC-LUID=814 “Patrik”, X-IRMC-LUID=2	Return entire Object Store
Compare all LUIDs with the stored. From this the sync engine will recreate the entire sync log. This is where the LUIDs are the most valuable.			
Store ‘10’ as the latest change number.			

5.7.5 Sync with User Data Entry During Synchronization

One record added to the PC (“John Doe”). One entry (“Tarzan”) added to the IrMC Object Store in the middle of the sync operation. Prior to the sync it looks like this:

PC PIM Object Store	
Record	**UID**
“A”	456
“D”	98567
“E”	10403
“Lars”	55002
“Ericsson”	234
“Jörgen”	1
Patrik	11
John Doe	15

Mobile IrMC Object Store	
Object	**LUID**
“A”	567
“Lars”	7456
“D”	12345
“E”	789
“Jörgen”	899
“Ericsson”	814
“Patrik”	2

Sync Engine Table	
PC UID	**Mobile LUID**
456	567
98567	12345
10403	789
55002	7456
40603	34345
234	814
1	899
11	2

The mobile has the following change log:

	Action	**Change #**	**LUID**	Data (for reference only)
Latest Change	Mod	10	2	“Patrik”
Oldest Change	Mod	9	7456	“B” to “Lars”
	Delete	8	34345	“C”
	Mod	7	814	“Ericsson”
	Mod	6	899	“Jörgen”
	Mod	5	789	“E”
	Mod	4	12345	“D”
	Mod	3	34345	“C”
	Mod	2	7456	“B”
	Mod	1	567	“A”

Sync Engine	Obex Command	Obex Response	Device
Get Change Log	GET ‘telecom/pb/luid/10.log’		
		SN: xyz1243 DID:12345678 Total-Records: 7 Max-Records: 20 (Empty)	Return Empty Change Log
OK. I know you!			
			User adds “Tarzan” to the Phone Book from the user interface.
Add “John Doe” to the device	PUT ‘telecom/pb/luid/.vcf’ “John Doe”		
		OK, LUID=123, CC=12 using AppResponse Header	Add “John Doe”, return response
The difference between the old change counter (10) and the new (12) is bigger than one.			
Get change log for changes after ‘10’	GET ‘telecom/pb/luid/10.log’		
		SN: xyz1243 DID:12345678 Total-Records: 9 Max-Records: 20 M:11:999 M:12:123	
	GET ‘telecom/pb/luid/999.vcf		
Change counter = 11		“Tarzan”, X-IRMC-LUID=999	
Change counter = 12…Complete			

The sync engine detected the new entry added by the user because the change counter increased more than one.

5.7.6 Reset

The device has been reset. All data in the device is lost.

Sync Engine	Obex Command	Obex Response	Device
			Reset
			Create new database, assign new Database ID by generating a unique 32 bit number. Empty Change Log.
Get Change Log	GET ‘telecom/pb/luid/12.log’		
		SN: xyz1243 DID:4564565 Total-Records: 9 Max-Records: 20 (Empty)	Return Empty Change Log with new DID.
I know you!			
Database id does not match last stored. Erase all UID relation tables and do slow sync!			

The sync engine detected the reset because the Database ID did not match.

5.7.7 Restore

The user wants to restore the database. This must be done with a Level 2 PUT by the sync engine.

Sync Engine	Obex Command	Obex Response	Device
Get Change Log	GET ‘telecom/pb/luid/12.log’		
		SN: xyz1243 DID:12345678 Total-Records: 9 Max-Records: 20 (Empty)	Return Change Log
I know you!			
	PUT ‘telecom/pb.vcf’ <data>		
			Create new database, assign new database id by generating a unique 32 bit number. Empty change log.
		OK	Write data and assign new UIDs
Get UID assigned	GET ‘telecom/pb.vcf’		
		<data>	Return Database
Get Information Log	GET ‘telecom/pb/info.log’		
		info.log	Return Info Log
Store Database ID			
GET Change Counter	GET ‘telecom/pb/cc.log’		
		12	Return CC.log
Store Change Counter			

A subsequent synchronization by a different sync engine will recognize that the database has been replaced because of the new Database ID and will do a slow sync.

5.7.8 A Modification

One record modified in the PC “John Doe” to “Homer Simpson”.

PC PIM Object Store	
Record	**UID**
“A”	456
“D”	98567
“E”	10403
“Lars”	55002
“Ericsson”	234
“Jörgen”	1
Patrik	11
Homer Simpson	15
Tarzan	16

Mobile IrMC Object Store	
Object	**LUID**
“A”	567
“Lars”	7456
“D”	12345
“E”	789
“Jörgen”	899
“Ericsson”	814
“Patrik”	2
Tarzan	999
John Doe	123

Sync Engine Table	
PC UID	**Mobile LUID**
456	567
98567	12345
10403	789
55002	7456
234	814
1	899
11	2
15	123
16	999

The mobile has the following change log:

	Action	**Change #**	**LUID**	Data (for reference only)
Latest Change	Add	12	123	“John Doe”
Oldest Change	Add	11	999	“Tarzan”
	Add	10	2	“Patrik”
	Modify	9	7456	“B” to “Lars”
	Delete	8	34345	“C”
	Add	7	814	“Ericsson”
	Add	6	899	“Jörgen”
	Add	5	789	“E”
	Add	4	12345	“D”
	Add	3	34345	“C”
	Add	2	7456	“B”
	Add	1	567	“A”

Sync Engine	Obex Command	Obex Response	Device
Get Change Log	GET ‘telecom/pb/luid/12.log’		
		SN: xyz1243 DID:12345678 Total-Records: 7 Max-Records: 20 (Empty)	Empty Change Log.
I know you!			
Modify “John Doe” to “Homer Simpson” in the device	PUT ‘telecom/pb/luid/123.vcf’ “Homer Simpson” Appl Command Header [Maximum Expected Change Counter=12]		
			No modifications to ‘123’ has been made since change counter 12, request Ok, modify “John Doe” to “Homer Simpson”
		OK, LUID=123, CC=13 using AppResponse Header	Send Response Back
Complete fast Synchronization…			

5.8 PUSH Commands

As described above, when initiated on the PC synchronization is rather straightforward. The PC user requests a sync. The PC, as an IrMC Client, requests a change log from a phone or other IrMC Server.

In some cases, it may desirable for the user of an IrMC Device to request a synchronization. For instance, a mobile phone user may want to push a “Sync Now” button on her phone to instruct a synchronization engine on a PC to begin synchronization.

To make the request, the phone must first act as an IrMC Client and request the PC to begin synchronization. The PC must receive this request as a server, and then turn the connection around and act as the IrMC Client and synchronize with the phone which is the IrMC Server.

To accomplish this the IrMC Phone can use Level 1 Information Exchange to send the “telecom/push.txt” object to the PC. A list of command(s) is contained in the object and is transferred by IrOBEX Put commands to remote device. A list of those commands, and their expected actions, are defined as follows.

5.8.1 Formal Definition of Push Command

<push-instruction-object-name> ::= “**telecom/push.txt**”

<push-instruction-object> ::= *<push-command>**

<push-command>::= *<push-put-command>* | *<push-sync-command>*

<push-put-command> ::= "**PUT:**" **<target-object-name>** *<CRLF>*

<push-sync-command> ::= "**SYNC:**" *<target-object-name>* *<CRLF>*

<target-object-name> ::=

{ "**telecom/pb.vcf**" | "**telecom/cal.vcs**" | “**telecom/note.vnt**” |
"**telecom/inmsg.vmg**" | "**telecom/outmsg.vmg**" | "**telecom/sentmsg.vmg**" |
"**telecom/rtc.txt**" | **default**" | *<any-IrMC-object>*
}

<any-IrMC-object> ::= ‘Any object name’

Any device that implements Level 4 Client Services must also implement PUSH command Server services.

Any device that implements Level 4 Server Services is not required to implement PUSH command Server services.

Any device that implements Level 4 Server Services is not required to implement PUSH Client Services.

For example, consider a PC-Based Sync Engine, i.e., an implementer of Level 4 Client Services; and a mobile phone, i.e., an implementer of Level 4 Server Services. In this example, the PC Sync Engine must also provide an IrMC Server to “listen” for PUSH commands from mobile phones. The Phones do not have to listen for this command, nor do they have to be capable of sending the PUSH command.

If the phone sends a PUSH command to start a synchronization session, the Sync Engine will initiate a synchronization with the phone that sent “telecom/push.txt”, if <*push-sync-command*> is included in push.txt and the Object Store specified in <*target-object-name*> is supported. It will initiate that synchronization session by reversing roles and acting as a client again. It will then connect to the phone (now an IrMC Server).

6. Device Information

The manufacturer-specific information of an IrMC Device is stored as a <*device-info-object*> with the name <*device-info-object-name*>. This object is mandatory for all IrMC Devices which support connection-oriented services 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, to provide information about the model and firmware/software versions of the device. It is used in synchronization to obtain the serial number 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. The format of the field is manufacturer dependent. This property identifier is optional.

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 IrMC-Version

The IrMC version property identifier specifies the version of IrMC supported by this device. This property did not exist in 1.0 implementations, but is mandatory for all future implementations. The only allowable field value at present is “1.1”.

IRMC-VERSION:1.1

6.1.9 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

6.1.10 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

6.1.11 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 was optional in IrMC 1.0 implementations. From IrMC 1.1 and into the future, this property identifier is mandatory.

SN:1218182THD000001-2

6.1.12 Phone Book Type

The Phone Book Type property identifier specifies the data format supported by the Phone Book application. It identifies the format(s) that the Object Store can transmit and the format(s) it can receive. The first listed item in the supported transmit format(s) is the default transmit format. If the Phone Book Object Store supports the vCard 2.1 format as described in this specification, then the field must contain VCARD2.1. Multiple data types can be separated by a ‘;’. If the Phone Book Object Store is not supported, the value must be NONE. This property identifier was added in IrMC 1.1 and is mandatory.

PB-TYPE-TX:VCARD2.1

PB-TYPE-RX:VCARD2.1

6.1.13 Calendar Type

The Calendar Type property identifier specifies the data format supported by the Calendar application. It identifies the format(s) that the Object Store can transmit and the format(s) it can receive. The first listed item in the supported transmit format(s) is the default transmit format. If the Calendar Object Store supports the vCalendar 1.0 format as described in this specification, then the field must contain VCAL1.0. Multiple data types can be separated by a ‘;’. If the Calendar Object Store is not supported, the value must be NONE. This property identifier was added in IrMC 1.1 and is mandatory.

CAL-TYPE-TX:VCAL1.0

CAL-TYPE-RX:VCAL1.0

6.1.14 Message Type

The Message Type property identifier specifies the data format supported by the Message application. It identifies the format(s) that the Object Store can transmit and the format(s) it can receive. The first listed item in the supported transmit format(s) is the default transmit format. If the Message Object Store supports the vMessage format as defined in this specification, then the field must contain VMSG1.1. Multiple data types can be separated by a ';'. If the Message Object Store is not supported, the value must be NONE. This property identifier was added in IrMC 1.1 and is mandatory.

MSG-TYPE-TX:VMSG1.1

MSG-TYPE-RX:VMSG1.1

6.1.15 Note Type

The Note Type property identifier specifies the data format supported by the Note application. It identifies the format(s) that the Object Store can transmit and the format(s) it can receive. The first listed item in the supported transmit format(s) is the default transmit format. If the Note Object Store supports the vNote format as defined in this specification, then the field must contain VNOTE1.1. Multiple data types can be separated by a ';'. If the Note Object Store is not supported, the value must be NONE. This property identifier was added in IrMC 1.1 and is mandatory.

NOTE-TYPE-TX:VNOTE1.1

NOTE-TYPE-RX:VNOTE1.1

6.1.16 Inbox Capability

The Inbox Capability property identifier specifies whether the devices Inbox can receive Obex frames containing multiple objects. If the value is SINGLE then the device can only receive Obex frames containing single objects. If the value is MULTIPLE, then the device can receive frames that contain multiple objects of the same type, e.g., vCard. This property identifier was added in IrMC 1.1 and is mandatory. If the property identifier isn't present, it should be assumed that the device only supports reception of SINGLE objects.

INBOX:MULTIPLE

6.1.17 Sent Box Capability

The Sent Box Capability property identifier specifies whether the device's Message application supports the Sent Box Object Store. If the value is YES then the device's Message application includes an Inbox, Outbox and Sentbox. If the value is NO then the device's Message application includes an Inbox and Outbox, but no Sent box. In addition, a value of NO is present if the device does not support a Message application. This property identifier was added in IrMC 1.1 and is mandatory. If the property identifier isn't present, a client must explicitly request the Sent Box Object to determine if it exists or not.

MSG-SENT-BOX:YES

6.1.18 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 IrMC 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 of devinfo.txt

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

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

<*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-irmc-version-field*> |
<*device-info-hardware-version-field*> |
<*device-info-hardware-date-field*> |
<*device-info-serial-number-field*> |
<*device-info-pb-type-tx*> |
<*device-info-pb-type-rx* > |
<*device-info-calendar-type-tx* > |
<*device-info-calendar-type-rx* > |
<*device-info-message-type-tx* > |
<*device-info-message-type-rx* > |
<*device-info-note-type-tx* > |
<*device-info-note-type-rx* > |
<*device-info-inbox-capability-field*> |
<*device-info-msg-sent-box*> |
<*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-irmc-version-field*> ::= “**IRMC-VERSION:1.1**”

<*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-pb-type-tx*> ::=
“**PB-TYPE-TX:**” {“**NONE**” | {“**VCARD2.1”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-pb-type-rx*> ::=
“**PB-TYPE-RX:**” {“**NONE**” | {“**VCARD2.1”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-cal-type-tx*> ::=
“**CAL-TYPE-TX:**” {“**NONE**” | {“**VCAL1.0”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-cal-type-rx*> ::=
“**CAL-TYPE-RX:**” {“**NONE**” | {“**VCAL1.0”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-msg-type-tx*> ::=
“**MSG-TYPE-TX:**” {“**NONE**” | {“**VMSG1.1”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-msg-type-rx*> ::=
“**MSG-TYPE-RX:**” {“**NONE**” | {“**VMSG1.1”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-note-type-tx*> ::=
“**MSG-NOTE-TX:**” {“**NONE**” | {“**VNOTE1.1”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-note-type-rx*> ::=
“**MSG-NOTE-RX:**” {“**NONE**” | {“**VNOTE1.1”** [**“;”**<*default-char-not-lf*>+]}}

<*device-info-inbox-capability-field*> ::=
“**INBOX:**” {“**SINGLE**” | “**MULTIPLE”**}

<*device-info-msg-sent-box*> ::=
“**MSG-SENT-BOX:**” {“**YES**” | “**NO”**}

<*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 Overview

The Phone Book application provides a means for a user to manage contact records, i.e., business cards, and to exchange those cards with other IrMC Devices. It also provides a means to monitor and maintain information about Call History.

In addition to the four levels of Information Exchange, there are three optional Call History objects which may be implemented regardless of the level of the Information Exchange supported. These three objects are: Incoming Call Objects, Outgoing Call Objects and Missed Call Objects.

The level of Information Exchange supported by the Phone Book application must be identified by the IAS PhoneBookSupport parameter as described in section 13.1.2.1 and must be identified in the Phone Book's Information Log.

The Phone Book application may store its objects in any internal format that is chosen by the implementer. However, the Phone Book application may support more than one format for Information Exchange. The type of objects supported, e.g., vCard2.1, must be identified in the PB-TYPE property of the Device Information Object. At a minimum, Phone Book applications must support Information Exchange using the vCard 2.1 Object format. Furthermore, the examples in this chapter use vCards.

Information about the Phone Book application is stored in the object named <*phone-book-information-log-object-name*>. The format of that object is <*phone-book-information-log-object*>.

7.2 Phone Book Level 1 Information Exchange

The minimum implementation of a phone book application provides a simple business card "push" 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*>.

Devices with an IrMC Phone Book application, regardless as to whether they provide connection-oriented services, are required to provide server support for Level 1 Information Exchange of Phone Book objects.

Level 1 Phone Book Example:

Local Device **PUT** [name - Celia.vcf]
Remote Device
device inbox:
Celia.vcf
optional UI indication "Do you want to save this number into your phone book? "

7.3 Phone Book Level 2 Information Exchange

This Object Store is named <phone-book-stream-object-name> and its format is defined as <phone-book-stream-object>. The first Object in the <phone-book-stream-object> is always the Object of the owner/local device. If no owner information is available, the first Object will be empty. Entries with restricted access are also listed as empty Objects.

Static Indexing Notes

If the Phone Book supports Static Indices support, the organization of the stream in the <*phone-book-stream-object*> must reflect the organization of the single <*phone-book-indexed-object*> Objects. Such Phone Books have all the Objects in the Phone Book Object Store listed in a <*phone-book-stream-object*>.

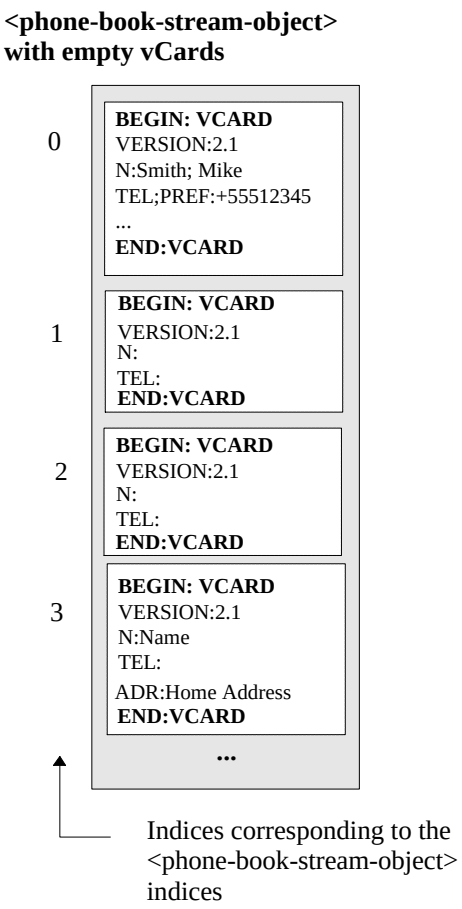


Figure 7-1 A Phone Book Stream With an Empty Object

When reading or writing all the Objects of a Phone Book with Static Indexing, all empty locations must be indicated by an empty Object as shown in the **Figure 7-1 A Phone Book Stream With an Empty Object**. This way it is possible to maintain the original location of the Objects in the Object Store. If empty Objects (as shown in **Figure 7-2 An Empty Phone Book Object**) are indicated these have to be taken into account by the receiver. If no new Phone Book Objects are made, the relative order of the entries remains the same in a Level 2 GET/PUT operation. Index "0", be it static or a LUID, is the Owner's object.

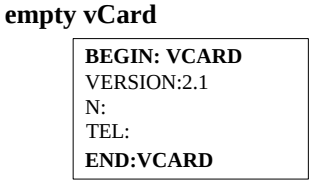


Figure 7-2 An Empty Phone Book Object, in this case an empty vCard

The <phone-book-stream-object> may be deleted by writing an empty entry on top of the original entry, i.e., by applying an OBEX PUT operation without a BODY to the original object.

Level 2 Phone Book Example:

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

7.4 Phone Book Level 3 Information Exchange

Each Phone Book Object is assigned a static index. Each Object is named as <*phone-book-indexed-object-name*>. The format of these objects is defined as <*phone-book-indexed-object*>. Index "0" always stores the Object of the owner/local device. If no owner Object is available, the location "0" will hold an empty Object.

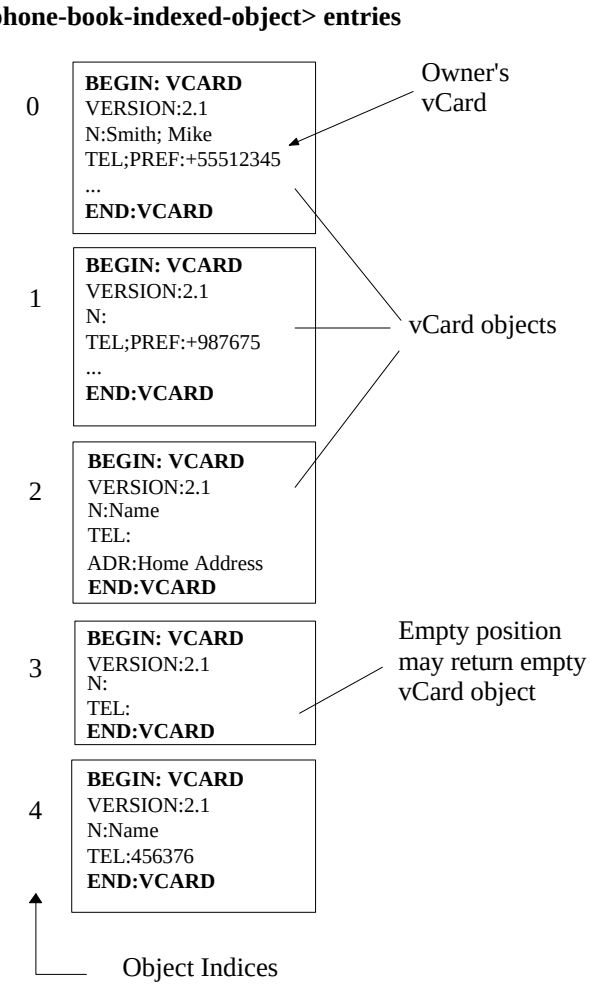


Figure 7-3 vCard entries with indices

These devices have the phone book entries organized as <*phone-book-indexed-object*>. 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]

7.5 Phone Book Level 4 Information Exchange

For synchronizing phone books of different devices, information about any changes or additions in the Phone Book Object Store is maintained in a Change Log <*phone-book-change-log-object-name*>. The format of that log is <*phone-book-change-log-object*>.

Each Phone Book Object is assigned a unique index, or LUID. Each Object is named as <*phone-book-unique-indexed-object-name*>. The format of these objects is defined as <*phone-book-unique-indexed-object*>. LUID "0" always stores the Object of the owner/local device. If no owner Object is available, the location "0" will hold an empty Object.

Furthermore, if a Change Counter Sync Anchor is maintained, the last used change counter value is stored in an object named <*phone-book-change-counter-object-name*>. The format of that Change Counter is <*phone-book-change-counter-object*>.

Detailed examples of Level 4 Information Exchange are given in Chapter 5 Synchronization.

7.6 Call History Objects

In order to allow access to the latest received, made and missed calls, devices that support calling may implement optional call history objects. These objects may be read, written and deleted in the same way as the <*phone-book-stream-object*> by using the OBEX GET and PUT operations. The Call History objects may only be accessed using Level 2 services. There is no Level 3 or 4 Information Exchange with Call History Objects.

7.6.1 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 Phone Book Objects, each with an individual index. This indexing is FIFO 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.

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 must be unambiguously stated by the IAS PhoneBookOptional parameter as described in section 13.1.2.1.

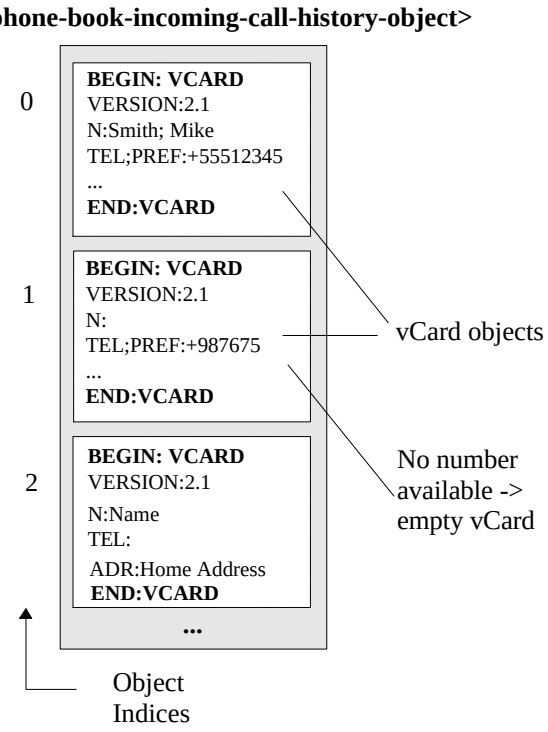


Figure 7-4 Incoming Call History Stream

7.6.2 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 Phone Book Objects, each with an individual index. This indexing is FIFO 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 must be unambiguously stated by the IAS PhoneBookOptional parameter as described in section 13.1.2.1.

7.6.3 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 Phone Book Objects, each with an individual index. This indexing is FIFO 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 must be unambiguously stated by the IAS PhoneBookOptional parameter as described in section 13.1.2.1.

7.7 Formal Definition of Phone Book Objects

7.7.1 Information Log

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

<*phone-book-information-log-object*> ::= {

<*information-log*>

<*incoming-call-log*> ::= "**ICL:**" {**"YES**" | **"NO**"}

<*outgoing-call-log*> ::= "**OCL:**" {**"YES**" | **"NO**"}

<*missed-call-log*> ::= "**MCL:**" {**"YES**" | **"NO**"}

}

7.7.2 Phone Book Minimum Support

<*phone-book-minimum-support-object-name*> ::= <*default-char-not-lf*>$^{+}$ <*extension*>

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

7.7.3 Phone Book Access Support

<*phone-book-stream-object-name*> ::= "**telecom/pb"** <*extension*>

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

7.7.4 Phone Book Index Support

<*phone-book-indexed-object-name*> ::= "**telecom/pb/**" <*digit*>$^{+}$ <*extension*>

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

7.7.5 Phone Book Synchronization Support

<*phone-book-unique-indexed-object-name*> ::= "**telecom/pb/luid/**" <*luid*> <*extension*>

<*phone-book-unique-indexed-object*> ::= <*common-vcard*> | <*other-object-format*>

<*phone-book-change-counter-object-name*> ::= "**telecom/pb/luid/cc.log**"

<*phone-book-change-counter-object*> ::= <*change-counter*>

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

<*phone-book-change-log-object*> ::= <*change-log-object*>

7.7.6 Call History Objects

7.7.6.1 Incoming Calls

<*phone-book-incoming-call-history-object-name*> ::= "**telecom/ich**" <*extension*>

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

7.7.6.2 Outgoing Calls

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

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

7.7.6.3 Missed Calls

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

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

7.8 vCard Object Format Support

All IrMC Phone Book applications, must support Information Exchange using the vCard Object Format as defined in the vCard 2.1 Specification. The vCard Phone Book objects are identified by the VCF extension in the NAME.

7.8.1 Mandatory Fields

All of 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. In particular, the Unique Identifier property, if present, must always be stored if storage space on the IrMC Device is sufficient.

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 [VCARD].

7.8.2 Formal Definitions as required by vCard 2.1

<extension> ::= "**.vcf**"

7.8.3 Default TEL Types Usage

There is a need for applications that provide support of the TEL properties in vCards to clearly identify what can be supported as TEL default properties. Without the clear definition of what TEL property typings are available for default, implementations may not be able to communicate TEL properties with each other.

The standard defaults as specified in [VCARD] indicate that VOICE is the default TEL parameter type. This would indicate that if the syntax within a vCard is either TEL:WORK or TEL:HOME that the default type association for this syntax is that these examples refer to Work Voice Phone Number and Home Voice Phone Number respectively. If any other TEL typing is contained on either a vCard TEL line or an X-IRMC-FIELDS TEL line with either of the above examples of TEL:WORK or TEL:HOME, it is safely assumed that the default VOICE usage is negated and must explicitly be invoked to be used properly.

7.8.4 Use of Pronunciation Capabilities in Information Log

There is a need for applications that support pronunciation of key fields within a record, i.e. Japanese Yomi pronunciations, to announce which fields will have pronunciation attributes. Without the capabilities of indicating which fields are to be interpreted as pronunciation fields there can potentially be a loss of data between applications.

Through the use of the standard vCard delimiters specified in Pronunciation Attribute Extension for vCard 1998-04-21 http://www.imc.org/pdi/vcard-pronunciation.html, it will be possible to identify which fields will have added pronunciations by utilizing the delimiters within the X-IRMC-FIELDS entry within info.log for vCard.

The rules for announcing pronunciation capabilities are as follows:

1. Announcing that a field will have pronunciation capabilities will be done by including angle brackets "<>" within the field definition.
2. A separate string length size must be carried for pronunciation strings which are contained within angle brackets "<>". The length is used to minimize the risk of data truncation and will be presented immediately after the angle brackets "<>" by the use of an equal sign "=" followed by the numeric value. The length takes into account the length of only those strings that exist within the angle brackets "<>".

For example, a name field which has a pronunciation for the 1st field will be represented as follows in X-IRMC-FIELDS:

N[1=20<>=25]; ENCODING=QUOTED-PRINTABLE;CHARSET=ISO-2022-JP

This example indicates that the 1st field has a length of 20, and that the pronunciation contained within the angle brackets has a length of 25.

For example, a name field which has pronunciations for the 1st and 2nd fields will be represented as follows in X-IRMC-FIELDS:

N[1=20<>=25;2=22<>=27]; ENCODING=QUOTED-PRINTABLE;CHARSET=ISO-2022-JP

This example indicates that the 1st field has a length of 20 with the pronunciation contained within the angle brackets having a length of 25, and the 2nd field has a length of 22 with the pronunciation contained within the angle brackets having a length of 27.

For example, an organization "ORG" with string length of 80 may have a pronunciation length of 85 and would be represented as follows:

ORG; ENCODING=QUOTED-PRINTABLE;CHARSET=ISO-2022-JP:=80<>=85

3. Not all applications will be able to store Kanji data and Yomi data separately. According to the vCard pronunciation specification, <Yomi> should be placed prior to Kanji and whole of "<Yomi>Kanji" should be encoded with including "<" and ">". This indicates that there might be an application which stores Yomi and Kanji data as one field data. Provided that <Yomi> is placed prior to Kanji, it's easy to sort data by Yomi for such device. In this case when a length is given and an asterisk is used to indicate the length of the Yomi, the entire length of the Yomi and Kanji is the given length minus 2 for the angle brackets.

On the assumption that Kanji and Yomi are handled as one field data, total maximum length of ORG is indicated as 10 bytes. This type of data storing can be indicated as follows:

ORG:.....:=10<>=*

(In this case, actual total length of Yomi and Kanji is 8 bytes, because the removal of the length of each of the angle brackets"<>".)

When an application sends [Kanji+Yomi] data to another application, the other application will process the contents according to its length as follows:

Case 1: [Total length -2] = length of [Kanji+Yomi]

Whole of [Kanji+Yomi] data will be sent.

Example:

- info.log length from application =10<>=*
- original Yomi data zyx
- original Kanji data ABC

Case 2: [Total length - 2] < length of [Kanji+Yomi]

The end portion of Kanji will be truncated when [Kanji+Yomi] is sent.

Example:

- info.log length from application =10<>=*
- original Yomi data zyxwvu
- original Kanji data ABC
- Stored data in other application <zxywvu>AB

Case3: [Total length - 2] < length of Yomi

The end portion of Yomi and whole of Kanji will be truncated when

[Kanji+Yomi] is sent.

Example:

- info.log length from application =10<>=*
- original Yomi data zyxwvusrqpo
- original Kanji data ABC
- Stored data in other application <zyxwvuts>

Generally, Yomi data is used for data sorting so, the Yomi data will have higher priority than Kanji as represented in Case 3.

7.8.5 TEL Types Usage in Information Log

There is a need for applications that provide support of the TEL properties in vCards to clearly identify TEL support when X-IRMC-FIELDS is used. Some applications may support any number of phone fields with any number of possible parameter combinations for each of the phone fields. Providing the exact number of phone fields in an application as well as stating all the possible types that may be available for those fields is important to successfully announce the TEL capabilities of an application.

Without the capabilities of explicitly announcing the TEL capabilities of an application, there is a possibility of loss of data if the number of supported phone fields and the total number of parameter types that can potentially be supported is not projected to other applications.

Through a combination of adding additional TEL field clarifying elements which includes the addition of a X-TEL-FIELDS to identify the number of phone fields in an application and a clear representation of all the possible types that may be provided by any single vCard record, this problem can be resolved.

To identify the number telephone fields any vCard record will have during a session, the following field must be present within the X-IRMC-FIELDS entry within info.log along with a numeric to represent the number of phones supported by the application on a per record basis:

```
...
X-TEL-FIELDS: 5
...
```

This indicates that the application from which the info.log has been requested, has support for 5 telephone fields. This field should be present for any TEL representations contained in X-IRMC-FIELDS, and should contain at least the value one "1" when present. This solves the problem of being unable to identify how many phones may be supported within an application record.

Through the example above, we know that there will exist 5 phone fields on a per record basis within the application from which we received the info.log. What we do not know at this point is what are all the possible phone types for these fields. TEL types should be identified within the info.log so that for each TEL parameter type that can possibly be supported within a field in any remote application, a corresponding TEL type syntax line will exist within the X-IRMC-FIELDS in the info.log.

A couple of different scenarios can exist for the possible phone types that could be contained in these 5 example fields.

In the simplest case for each of the 5 fields, one and only one telephone type will be available for each of the 5 fields. For demonstration sake, lets identify each of the fields as a different TEL type - so we will have a representation that indicates 5 telephone fields in the application and that there exists only 5 possible types for these 5 telephone fields and that the use of each of the telephone types is mutually exclusive such that if TEL:TYPE=WORK;VOICE is used in the definition of the typing for one field, none of the other 5 fields in the record may have the same type definition. Within X-IRMC-FIELDS within the info.log, the scenario presented above would be represented as follows:

...
X-TEL-FIELDS: 5
TEL:TYPE=WORK;VOICE
TEL:TYPE=WORK;FAX
TEL:TYPE=WORK;PAGER
TEL:TYPE=HOME;VOICE
TEL:TYPE=HOME;FAX
...

This example indicates that within an application, X-IRMC-FIELDS indicates that the application from which the info.log has been received has 5 telephone fields and only one of each of the fields per record may be either a Work Voice Phone Number, Work Fax Phone Number, Work Pager Phone Number, Home Voice Phone Number, or a Home Fax Phone Number.

This works when vCards from a remote application is received into a local application. Each incoming telephone field has only one local telephone field location to map to. The telephone record capabilities have been announced through the info.log so that is known explicitly that only 5 telephone fields exist per vCard and there is only one of each of the 5 distinct types that can exist on a per field basis for each record. This also provides for explicit mapping when vCards are sent from a local application to a remote application. Each outgoing field has only one clear field location to map to.

The second more complicated scenario is where there exist 5 telephone fields where each of the fields can have any number of the previously indicated telephone types on a per record basis. This becomes more complicated because there is a potential in this scenario that not all the telephone fields will be able to be represented or transferred from a target application to a local application and vice versa. A need arises to allow for an indication of what are the multiple possible types that can be supported on a per record basis. For demonstration sake, lets use the 5 fields as well as the 5 field types identified from the simple example above.

However, the usage of the TEL types in this scenario are not mutually exclusive on a per record basis so we will have a representation that indicates 5 telephone fields in the application and that there will no longer exist only 5 possible telephone types for these 5 telephone fields. There is a potential that in this scenario that each of the 5 telephone fields may contain the same typing or any combination of typings such that if TEL:WORK;VOICE is used in the definition of the typing for one field, any of the other 5 fields in the record may also have the same type definition.

To allow for a mapping and in this scenario it is also necessary to indicate the number of times each of the telephone types presented may be used within a record. To allow for the clear representation of the maximum number of types that may occur on a per record basis, the maximum number of potential iterations that could occur for each telephone type may be optionally represented by the inclusion of that number within parens "()"as part of the type syntax for X-IRMC-FIELDS within the info.log. It should be noted that if no parens are present, the default behavior is to interpret the TEL type as a single instance occurrence. The following example indicates the typing of TEL:TYPE=WORK;VOICE can be repeated a maximum of 5 times on a per record basis:

TEL:TYPE=WORK;VOICE:(5)

The following example indicates that the typing of TEL:TYPE=WORK;VOICE can be repeated a maximum of 5 times on a per record basis and that maximum size of the telephone number is 24:

TEL:TYPE=WORK;VOICE:(5)=24

Within X-IRMC-FIELDS within the info.log, the scenario presented where there exists 5 telephone fields with a maximum length of 24 for each field on a per record basis and each of the 5 phone types may be represented in all 5 fields in any given record would be represented as follows:

```
...
X-TEL-FIELDS: 5
TEL:TYPE=WORK;VOICE:(5)=24
TEL:TYPE=WORK;FAX:(5)=24
TEL:TYPE=WORK;PAGER:(5)=24
TEL:TYPE=HOME;VOICE :(5)=24
TEL:TYPE=HOME;FAX:(5)=24
...
```

This example indicates that within an application, X-IRMC-FIELDS indicates that the application from which the info.log has been received has 5 telephone fields, where each field is of a length 24, and the total of the 5 fields per vCard record may contain number up to 5 of any one type or any combination of Work Voice Phone Number, Work Fax Phone Number, Work Pager Phone Number, Home Voice Phone Number, or a Home Fax Phone Number.

When vCards from a remote application is received into a local application, each incoming field has projected the maximum possible mappings that may occur on a per record basis as announced in X-IRMC-FIELDS through the info.log.

A problem occurs when records are to be sent to another application when the question arises, "Which of the potentially large number of telephone fields that the local application contains are important to store in an application that can only store a small number of telephone fields?" Without the info.log clearly identifying which telephone type fields are the potentially important fields to a user in the target application where there exists many possible telephone type combinations, there is a possibility that when a new record is transferred between a local device to the remote device that not all the proper telephone field record data may be transmitted. To fix this problem, Telephone field priority numbers are optionally to be contained on each line of the TEL type description within X-IRMC-FIELDS in the info.log within curly braces "{}". A TEL type that can be used (n) times on a per records basis, should also optionally indicate the priority for each of the TEL type's instance by separating each instance in the curly brace by a comma ",". If the optional field priority numbers are not utilized, then the field priority order will be determined by the order in which the TEL type descriptions are listed in the X-IRMC-FIELDS, TOP to BOTTOM. TEL types that have the possibility of multiple occurrences within a record, and that do not have a field priority order explicitly defined, will be interpreted as having the (n) occurrences as the priority followed by TOP to BOTTOM representation in X-IRMC-FIELDS.

Within X-IRMC-FIELDS within the info.log, a simple scenario of the default use of priorities as stated above would be represented as follows:

```
...
X-TEL-FIELDS: 3
TEL:TYPE=WORK;VOICE
TEL:TYPE=WORK;FAX
TEL:TYPE=WORK;PAGER
TEL:TYPE=HOME;VOICE
TEL:TYPE=HOME;FAX
...
```

This example uses the default priority ordering when writing new records to a remote application. Since there are only 3 fields, the 1st 3 TEL type syntax strings are the indicated order that is important to the remote application: Work Voice Phone Number, Work Fax Phone Number, and then Work Pager Phone Number.

Within X-IRMC-FIELDS within the info.log, a simple scenario of the curly brace use of priorities as stated above would be represented as follows:

```
...
X-TEL-FIELDS: 3
TEL:TYPE=WORK;VOICE:{1}
TEL:TYPE=WORK;FAX:{3}
TEL:TYPE=WORK;PAGER:{5}
TEL:TYPE=HOME;VOICE:{4}
TEL:TYPE=HOME;FAX:{2}
...
```

This example uses the curly brace priority ordering when writing new records to a remote application. Since there are only 3 fields, the fields which have the field priority numbers represented in curly braces indicate the order that is important to the remote application: Work Voice Phone Number, Home Fax Phone Number, Work Fax Phone Number, Home Voice Phone Number and then Work Pager Phone Number.

Within X-IRMC-FIELDS within the info.log, a more complicated scenario of the curly brace use of priorities presented here represents the existence of 3 telephone fields with a maximum length of 24 for each field on a per record basis and any the 5 phone types may be represented in all 3 fields in any given record with a specific field order for newly created records, would be represented as follows:

```
...
X-TEL-FIELDS: 3
TEL:TYPE=WORK;VOICE:(3){1,6,7}=24
TEL:TYPE=WORK;FAX:(3){2,8,9}=24
TEL:TYPE=WORK;PAGER:(3){4,10,11}=24
TEL:TYPE=HOME;VOICE :(3){3,12,13}=24
TEL:TYPE=HOME;FAX:(3){5,14,15}=24
...
```

This example indicates that the remote device has only 3 telephone fields on a per record basis, where any of the 3 fields may have any combination of 5 TEL types presented, the maximum length for each field is 24 and when writing new records to this device, the fields that are most important to the receiving application for a user are based on the priorities within the curly braces:

1. Work Voice Phone Number
2. Work Fax Phone Number
3. Home Voice Phone Number
4. Work Pager Phone Number
5. Home Fax Phone Number
6. Work Voice Phone Number
7. Work Voice Phone Number
8. Work Fax Phone Number
9. Work Fax Phone Number
10. Home Voice Phone Number
11. Home Voice Phone Number
12. Work Pager Phone Number
13. Work Pager Phone Number
14. Home Fax Phone Number
15. Home Fax Phone Number

The rules:

1. To indicate that an application has support for a specified number of telephone fields, the X-IRMC-FIELDS entry within the info.log must contain the following:

   ```
   X-TEL-FIELDS: 5
   ```

Where the number after the colon ":" indicates the number of telephone fields in the remote application. The number after the colon should be present even if there is only one field available in an application.

2. TEL types should be identified within the info.log so that for each TEL parameter type that can possibly be supported within a field in any remote application, a corresponding TEL type syntax line will exist within the X-IRMC-FIELDS in the info.log.

3. To allow for the clear representation of the maximum number of types that may occur on a per record basis, the maximum number of potential iterations that could occur for each telephone type may be optionally represented by the inclusion of that number within parens "()"as part of the type syntax for X-IRMC-FIELDS within the info.log.

   The following example indicates the typing of TEL:TYPE=WORK;VOICE can be repeated a maximum of 5 times on a per record basis:

   ```
   TEL:TYPE=WORK;VOICE:(5)
   ```

4. It should be noted that if no parens with a value are present, the default behavior is to interpret the TEL type as a single instance occurrence.

5. Telephone field priority numbers are optionally to be contained on each line of the TEL type description within X-IRMC-FIELDS in the info.log within curly braces "{}".

6. A TEL type that can used (n) times on a per records basis, should also optionally indicate the priority for each of the TEL type's instance by separating each instance in the curly brace by a comma ",".

7. If the optional field priority numbers are not utilized, then the field priority order will be determined by the order in which the TEL type descriptions are listed in the X-IRMC-FIELDS, TOP to BOTTOM.

8. TEL types that have the possibility of multiple occurrences within a record, and that do not have a field priority order explicitly defined, will be interpreted as having the (n) occurrences as the priority followed by TOP to BOTTOM representation in X-IRMC-FIELDS.

8. Calendar

8.1 Calendar Overview

The Calendar application provides a means for a user to manage appointments and “to-do” items and to exchange those objects with other IrMC Devices.

The level of Information Exchange supported by the Calendar application must be identified by the IAS CalendarSupport parameter as described in section 13.1.2.2 and must be identified in the Calendar’s Information Log.

The Calendar application may store its objects in any internal format that is chosen by the implementer. However, the Calendar application may support more than one format for Information Exchange. The type of objects supported, e.g., vCalendar 1.0, must be identified in the CAL-TYPE property of the Device Information Object. At a minimum, Calendar applications must support Information Exchange using the vCalendar 1.0 Object format. Furthermore, the examples in this chapter use vCalendar objects.

Information about the Calendar application is stored in the object named <*calendar-information-log-object-name*>. The format of that object is <*calendar-information-log-object*>.

8.2 Calendar Level 1 Information Exchange

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

Devices with an IrMC Calendar application, regardless as to whether they provide connection-oriented services, are required to provide server support for Level 1 Information Exchange of Calendar objects.

Level 1 Calendar Example:

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

8.3 Calendar Level 2 Information Exchange

This Object Store is named <*calendar-stream-object-name*> and its format is defined as <*calendar-stream-object*>.

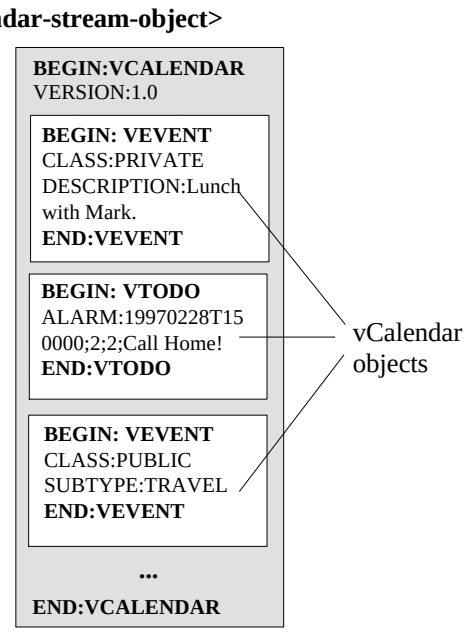


Figure 8-1 A Calendar Object Stream

The entire <*calendar-stream-object*> may be deleted by writing an empty entry on top of the original entry, i.e., by applying an OBEX PUT operation without a BODY to the original object.

Level 2 Calendar Example:

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

8.4 Calendar Level 3 Information Exchange

Each Calendar Object is assigned a static index. Each Object is named as <*calendar-indexed-object-name*>. The format of these objects is defined as <*calendar-indexed-object*>.

The organization of the stream in the <*calendar-stream-object*> must reflect the organization of the single <*calendar -indexed-object*> Objects. All the Objects in the Calendar Object Store are listed in a <*calendar-stream-object*>.

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 Calendar 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. As internal organization of the calendar entries may have significance to the application user, no unauthorized changes in the entry order shall be made.

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.

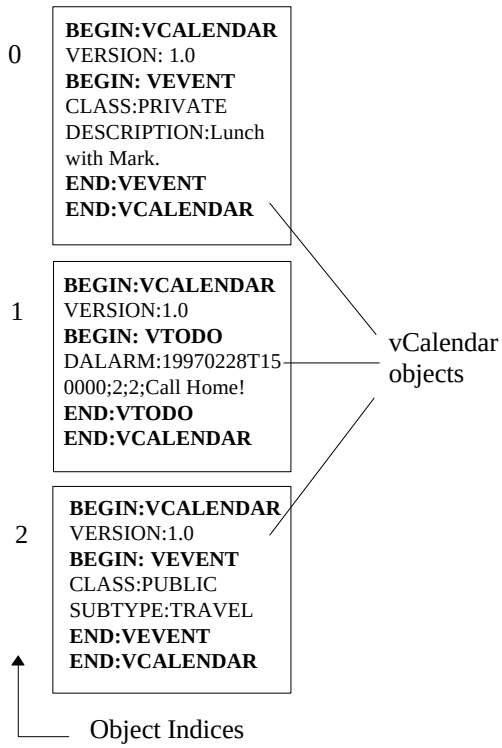


Figure 8-2 Calendar Objects with Indices

Calendar Index Support Example:

Local Device	**GET**	[name - telecom/cal/2.vcs]
Remote Device	**GET response**	[response code - success body - telecom/cal/2.vcs]
Local Device	**EDIT 1.vcs**	
	PUT	**[**name - telecom/cal/1.vcs]

8.5 Calendar Level 4 Information Exchange

For synchronizing calendars of different devices, information about any changes or additions in the Calendar Object Store is maintained in a Change Log <*calendar-change-log-object-name*>. The format of that log is <*calendar-change-log-object*>.

Each Calendar Object is assigned a unique index, or LUID. Each Object is named as <*calendar-unique-indexed-object-name*>. The format of these objects is defined as <*calendar-unique-indexed-object*>.

Furthermore, if a Change Counter Sync Anchor is maintained, the last value used is stored in an object named <*calendar-change-counter-object-name*>. The format of that Change Counter is <*calendar-change-counter-object*>.

Detailed examples of Level 4 Information Exchange are given in Chapter 5 Synchronization.

8.6 Formal Definition of Calendar

8.6.1 Information Log

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

<*calendar-information-log-object*> ::= <*information-log*>

8.6.2 Calendar Minimum Support

<*calendar-minimum-support-object-name*> ::= <*default-char-not-lf*>+ <*extension*>

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

8.6.3 Calendar Access Support

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

<*calendar-stream-object*> ::= <*common-vcalendar*>* | <*other-object-format*>*

8.6.4 Calendar Index Support

<*calendar-indexed-object-name*> ::= "**telecom/cal/**" <*digit*>* <*extension*>

<*calendar-indexed-object*> ::= <*common-vcalendar*>* | <*other-object-format*>*

8.6.5 Calendar Synchronization Support

<*calendar-unique-indexed-object-name*> ::= "**telecom/cal/luid/**" <*luid*> <*extension*>

<*calendar-unique-indexed-object* > ::= <*common-vcalendar*>* | <*other-object-format*>

<*calendar-change-counter-object-name*> ::= "**telecom/cal/luid/cc.log**"

<*calendar-change-counter-object* > ::= <*change-counter*>

<*calendar-change-log-object-name*> ::= "**telecom/cal/luid/**" <*change-counter*> "**.log**"

<*calendar-change-log-object*> ::= <*change-log-object*>

8.7 vCalendar Object Format Support

All IrMC Calendar applications must support Information Exchange using the vCalendar Object Format as defined in the vCalendar 1.0 specification. The vCalendar Objects are identified by VCS extension in the NAME.

Received vCalendar objects may include fields which are not supported by the device. It is recommended that no supplied fields be removed in order to maintain the original content. In particular, the Unique Identifier property, if present, shall always be stored if storage space on the IrMC Device is sufficient.

IrMC applications must include the *Version* number in addition to the other mandatory fields in all vCalendar objects transmitted.

8.7.1 Mandatory vCalendar Event Fields

The following fields are mandatory for vCalendar Event objects. Other fields are recommended including DTEND, PRIORITY, STATUS and SUMMARY. However, this specification can not mandate inclusion of these fields because to do so would require that at times they be transmitted as empty fields. However, some non-IrMC products on the market will reject vCalendar records with empty fields such as DTEND. For this reason, the only mandatory vCalendar Event fields are:

vCalendar field	Property Name	Description
Version	VERSION	Version of the vCalendar Specification used in the Implementation
Description	DESCRIPTION	Provides details of the vCalendar Entity.
Start Date/Time	DTSTART	Defines the date and the Event will start.

8.7.2 Mandatory vCalendar ToDo Fields

vCalendar field	Property Name	Description
Version	VERSION	Version of the vCalendar Specification used in the Implementation
Categories	CATEGORIES	Categories of the vCalendar Entity.
Date/Time Completed	COMPLETED	Defines the Date and Time the Todo was actually completed.
Description	DESCRIPTION	Provides details of the vCalendar Entity.
	Due Date/Time	DUE Date and time the Todo is to be Competed.
Priority	PRIORITY	Priority of the vCalendar Entity.
Status	STATUS	Status associated with the vCalendar entity.
Summary	SUMMARY	Short summary or subject of the vCalendar entity.

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

8.7.3 Formal Definitions as Required by vCalendar 1.0

<*extension*> ::= **".vcs"**

8.7.4 vCalendar Stream Characteristics

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.

8.7.5 Usage of Telephony URLs in Calendar Events

This section describes how Telephony URLs can be used in calendar procedure alarms. This usage enables automatic initiation of phone/fax/modem call to indicated phone number.

vCalendar PALARM Property Name Handling

When using the PALARM property it is possible to include an URL which will be "executed" when the alarm time. This URL needs only to fulfill the requirements on URL format set forth in [RFC1738]. This section describes how special telephony URLs can be used to initiate phone, fax, or modem calls.

The URLs for telephony [URL_TELEPHONY] may be used in the place of the URL in the procedure alarm. It is recommended that the telecom device prompts the end user prior to executing a procedure alarm that causes any kind of call to be made.

Informative Examples

Initiating a phone call:

```
PALARM:19980209T120000;;;phone:+358-55-123-12345
```

Initiating a fax call:

```
PALARM:19980311T130000;;;fax:+358-55-123-12345
```

Initiating a modem (data) call:

```
PALARM:19980203T141200;;;modem:+3585512341234;type=v32b?7e1
```

[RFC1738] Uniform Resource Locators (URL). December 1994. T.

Berners-Lee et al. <URL:ftp://ds/internic.net/rfc/rfc1738.txt>

[URL_TELEPHONY] URLs for Telephony. November 1997. A. Vähä-Sipilä, Nokia.

<URL:http://ftp.ds.internic.net/internet-drafts/draft-antti-telephony-url-04.txt>

8.7.6 Object Deletion for vCalendar

This section describes a consistent mechanism for the deletion of vCalendar objects in a Level 2, 3 or Level 4 Object.

NOTE: Within the examples below, any blank lines between BEGIN:VCALENDAR and END:VCALENDAR are only provided to make this section easier to read. They should not be included in the actual calendar stream. In order to provide for deletion of specific vCalendar entities, it is necessary to write back an empty object. In reference to the vCalendar stream object example in figure 8.2 in the IrMC specification, we are presented with the following data stream

<calendar-stream-object>

BEGIN:VCALENDAR
VERSION:1.0

BEGIN:VEVENT
CLASS:PRIVATE
DESCRIPTION: Lunch with Mark.
DTSTART:19980102T120000
END:VEVENT

BEGIN:VTODO
DALARM:19970228T150000;PT5M;2CallHome!
CATEGORIES:
COMPLETED:
DESCRIPTION:
DUE:
PRIORITY:
STATUS:
SUMMARY:
END:VTODO

BEGIN:VEVENT
CLASS:PUBLIC
X-SUBTYPE: TRAVEL
DESCRIPTION: Fly To London
DTSTART:19980301T090000
END:VEVENT
...
END:VCALENDAR

To delete the first VEVENT object within the VCALENDAR for the datastore presented above, one would write the object stream as follows:

<*calendar-stream-object*>

BEGIN:VCALENDAR
VERSION:1.0

BEGIN:VEVENT
END:VEVENT

BEGIN:VTODO
DALARM:19970228T150000;PT5M;2CallHome!
END:VTODO

BEGIN:VEVENT
CLASS:PUBLIC
X-SUBTYPE: TRAVEL
END:VEVENT
...
END:VCALENDAR

The writing of the stream record in this format, would delete the 1st VEVENT object in the target datastore.

Similarly for the deletion of a VCALENDAR Index Object, we are presented the following index object entries as presented in Figure: 8.2 of the IrMC specification:

<calendar-indexed-object> entries

BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VEVENT
CLASS:PRIVATE
DESCRIPTION: Lunch with Mark.
END:VEVENT
END:VCALENDAR

BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VTODO
DALARM:19970228T150000;PT5M;2CallHome!
END:VTODO
END:VCALENDAR

BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VEVENT
CLASS:PUBLIC
X-SUBTYPE: TRAVEL
END:VEVENT
END:VCALENDAR

To delete the first VEVENT object within the VCALENDAR for the datastore presented above, one would write the 1st object index item as follows:

<calendar-indexed-object> entries

BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VEVENT
END:VEVENT
END:VCALENDAR

The writing of the 1st record presented above would delete the 1st VEVENT object in the target datastore.

9. Messaging

9.1 Message Overview

The Message application provides a means for a user to manage messages and to exchange those messages with other IrMC Devices. It supports three primary objects:

- Inbox – a collection of messages that have been received
- Outbox – a collection of messages that may be transmitted
- Sent – a collection of messages that have been transmitted (NOTE: support of the Sent Object Store is optional and is indicated in the Device Information Object.)

In addition there is an optional Missed Message History Object which may be implemented regardless of the level of Information Exchange that is supported.

The level of Information Exchange supported by the Message application must be identified by the IAS MessageSupport parameter as described in section 13.1.2.3 and must be identified in the Message application's Information Log.

The Message application may store its objects in any internal format that is chosen by the implementer. However, the Message application may support more than one format for Information Exchange. The type of objects supported, e.g., vMessage, must be identified in the MSG-TYPE property of the Device Information Object. At a minimum, Message applications must support Information Exchange using the vMessage 1.1 Object format as defined later in this chapter. Furthermore, the examples in this chapter use vMessage objects.

Information about the Message application is stored in the object named <*message-information-log-object-name*>. The format of that object is <*message-information-log-object*>.

When moving vMsg objects from the Inbox to other Object Stores, care should be taken to make sure that there aren't multiple

9.2 Message Level 1 Information Exchange

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

Devices with an IrMC Message application, regardless as to whether they provide connection-oriented services, are required to provide server support for Level 1 Information Exchange of Message objects.

Level 1 Message Example:

Local Device **PUT** [name – fish.vmg]
Remote Device
device inbox
fish.vmg
optional UI indication "Do you want to save this entry in your mailbox? "

9.3 Message Level 2 Information Exchange

Message applications with Level 2 Information Exchange implement three Message Object Stores listing all the Incoming, Outgoing and Sent Message Objects (note that none of the three message Object Stores includes Inbox message entries.) These object stores are called <*message-incoming-stream-object-name*>, <*message-outgoing-stream-object-name*> and <*message-sent-stream-object-name*>, and are defined as <*message-incoming-stream-object*>, <*message-outgoing-stream-object*> and <*message-sent-stream-object*>.

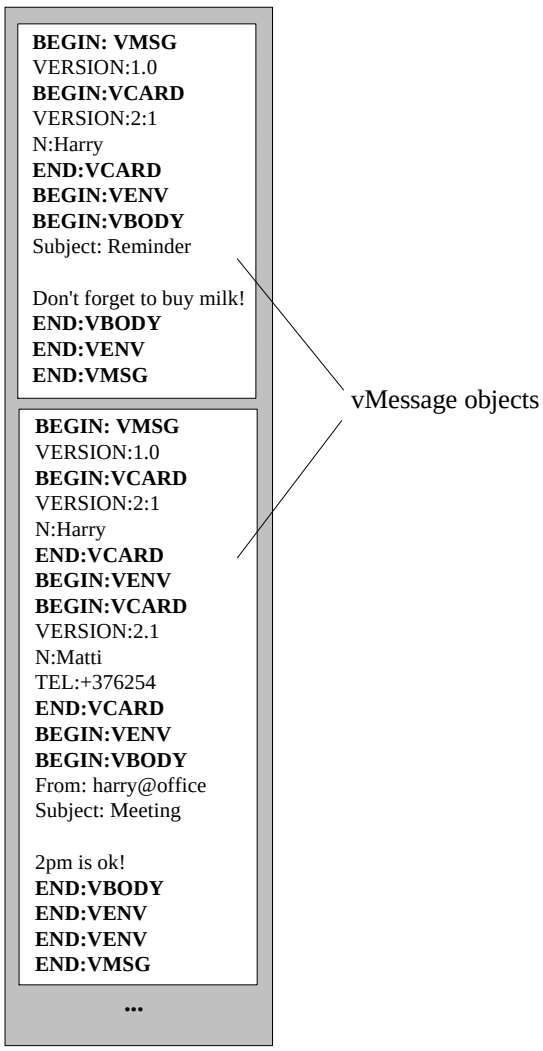


Figure 9-1 A stream of outgoing messages

The entire <*message-stream-object*> may be deleted by writing an empty entry on top of the original entry, i.e., by applying an OBEX PUT operation without a BODY to the original object. Entries with restricted access are also listed as empty Objects.

Level 2 Message Example:

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

9.4 Message Level 3 Information Exchange

Each Message Object is assigned a static index or unique index. 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*>. Messages that have been sent are written and stored as <*message-sent-indexed-object*> with the name <*message-sent-indexed-object-name*>.

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.

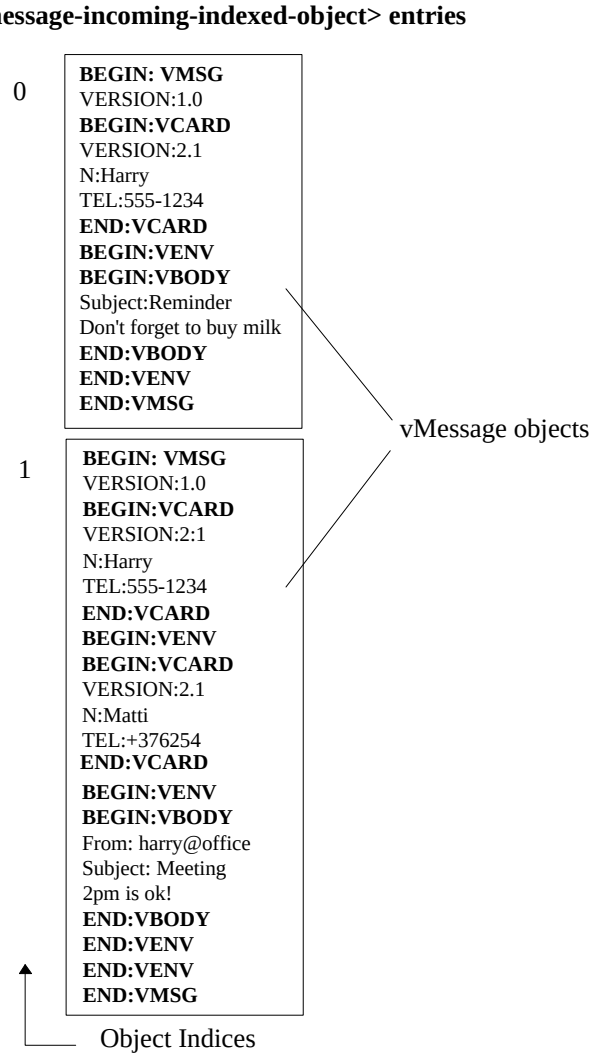


Figure 9-2 Received messages with indices

Message Index Example - Incoming Messages: Three messages in the incoming list. A new message is received and stored in the index 0.

telecom/msg/in/**0**.vmg [shiny]
telecom/msg/in/1.vmg [happy]
telecom/msg/in/**2**.vmg [people]

-> new message [hello] received

telecom/msg/in/0.vmg [hello]
telecom/msg/in/1.vmg [shiny]
telecom/msg/in/2.vmg [happy]
telecom/msg/in/3.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/0.vmg [shiny]
telecom/msg/out/1.vmg [happy]
telecom/msg/out/2.vmg [people]

-> new to be sent message [hello] received

telecom/msg/out/0.vmg [shiny]
telecom/msg/out/1.vmg [happy]
telecom/msg/out/2.vmg [people]
telecom/msg/out/3.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/**0**.vmg [happy]
telecom/msg/out/**1**.vmg [people]
telecom/msg/out/2.vmg [hello]

-> two messages are sent

telecom/msg/out/0.vmg [hello]

Message Index Support Example:

Local Device	**GET**	[name - telecom/msg/in/2.vmg]
Remote Device	**GET response**	[response code - success body - telecom/msg/in/2.vmg]
Local Device	**EDIT 1.vmg** **PUT**	**[**name - telecom/msg/in/1.vmg]

9.5 Message Level 4 Information Exchange

For synchronizing the three Message Object Stores of different devices, information about any changes or additions in the Object Stores is maintained in Change Logs <*message-incoming-change-log-object-name*>, <*message-outgoing-change-log-object-name*> and <*message-sent-change-log-object-name*>. The format of the logs is <*message-incoming-change-log-object*>, <*message-outgoing-change-log-object*> and <*message-sent-change-log-object*>.

Each Message Object is assigned a unique index, or LUID. Messages received from an external source are stored as <*message-incoming-unique-indexed-object*> with the name <*message-incoming- unique-indexed-object-name*>. Correspondingly, messages to be sent to an external end point are written and stored as <*message-outgoing-unique-indexed-object*> with the name <*message-outgoing-unique-indexed-object-name*>. Messages that have been sent are written and stored as <*message-sent-unique-indexed-object*> with the name <*message-sent- unique-indexed-object-name*>.

Furthermore, if a Change Counter Sync Anchor is maintained, then the last used Change Counter value is stored in objects named <*message-incoming-change-counter-object-name*>, <*message-outgoing-change-counter-object-name*> and <*message-sent-change-counter-object-name*>. The format of that Change Counter is <*message-incoming-change-counter-object*>, <*message-outgoing-change-counter-object*> and <*message-sent-change-counter-object* >.

The LUIDs should be unique across all three Object Stores, since it is possible that an item may move from one store to another and there needs to be a way to keep track of such movement.

Detailed examples of Level 4 Information Exchange are given in Chapter 5 Synchronization.

When a message is created in the Outbox, regardless if it is to be transmitted or not, an entry is created in the Outbox Change Log to indicate its creation. When the message is successfully sent by the device itself, a new entry is created in the Call Message History Object Store, if such a store is present. If the Sent Box is supported, a

copy of the message with the same LUID is saved in the Sent Box and an entry is written into the Sent Box Change Log. When a message is successfully sent

SPECIAL HANDLING OF OUTBOX MESSAGES DURING SYNC

Outboxes contain in-process message items. Full synchronization of messages that exist in an Outbox would mean that potentially 2 or more applications may end up with the same messages. Once this has occurred there exists the possibility that the intended recipient of the message will receive two copies of the same message.

With regard to outgoing message objects, synchronization will be implemented as a transfer and delete operation. The user will then have a single outgoing item that has been transferred to a new application and the originally existing vMessage on the originating application will be deleted. The Sync Client is responsible for carrying out the delete and transfer operation.

9.6 Message Missed 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 waiting to be 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.

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.

The support of this object is optional. If the object is supported, it must be unambiguously stated by the IAS MessageOptional parameter as described in section 13.1.2.3 and in the Information Log.

9.7 Message Sent Box Support

Implementation of the Sent Object Store is optional. Because of this there is a need to indicate whether is has been implemented. If the object is supported, it must be unambiguously stated by the IAS MessageOptional parameter as described in section 13.1.2.3 and in the Device Information Object.

9.8 Formal Definition of Message

9.8.1 Information Log

<message-incoming-information-log-object-name> ::= “**telecom/msg/in/info.log**”

<message-incoming-information-log-object> ::= {

<information-log>

<missed-message-history-call-log> ::= “**MMHL:**” **“YES**” | **“NO**”

}

<message-outgoing-information-log-object-name> ::= “**telecom/msg/out/info.log**”

<message-outgoing-information-log-object> ::= {

<information-log>

<missed-message-history-call-log> ::= “**MMHL:**” **“YES**” | **“NO**”

}

<message-sent-information-log-object-name> ::= “**telecom/msg/sent/info.log**”

<message-sent-information-log-object> ::= {

<information-log>

<missed-message-history-call-log> ::= “**MMHL:**” **“YES**” | **“NO**”

}

9.8.2 Minimum Support

<*message-minimum-support-object-name*> ::= <*default-char-not-lf*>+ <*extension*>
<*message-minimum-support-object*> ::= <*common-vmessage*> | <*other-object-format*>

9.8.3 Message Access Support

<*message-incoming-stream-object-name*> ::= "**telecom/msg/in"** <*extension*>
<*message-incoming-stream-object*> ::= <*common-vmessage*>* | <*other-object-format*>
<*message-outgoing-stream-object-name*> ::= "**telecom/msg/out"** <*extension*>
<*message-outgoing-stream-object*> ::= <*common-vmessage*>* | <*other-object-format*>
<*message-sent-stream-object-name*> ::= "**telecom/msg/sent"** <*extension*>
<*message-sent-stream-object*> ::= <*common-vmessage*>* | <*other-object-format*>

9.8.4 Message Index Support

<*message-incoming-indexed-object-name*> ::= "**telecom/msg/in/**" <*digit*>+ <*extension*>
<*message-incoming-indexed-object*> ::= <*common-vmessage*> | <*other-object-format*>
<*message-outgoing-indexed-object-name*> ::= "**telecom/msg/out/**" <*digit*>+ <*extension*>
<*message-outgoing-indexed-object*> ::= <*common-vmessage*> | <*other-object-format*>
<*message-sent-indexed-object-name*> ::= "**telecom/msg/sent/**" <*digit*>+ <*extension*>
<*message-sent-indexed-object*> ::= <*common-vmessage*> | <*other-object-format*>

9.8.5 Missed Messages History Object

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

9.8.6 Message Synchronization Support

<*message-incoming-unique-indexed-object-name*> ::= "**telecom/msg/in/luid/**" <*luid*> <*extension*>
<*message-incoming-unique-indexed-object*> ::= <*common-vmessage*> | <*other-object-format*>
<*message-outgoing-unique-indexed-object-name*> ::= "**telecom/msg/out/luid/**" <*luid*> <*extension*>
<*message-outgoing-unique-indexed-object*> ::= <*common-vmessage*> | <*other-object-format*>
<*message-sent-unique-indexed-object-name*> ::= "**telecom/msg/sent/luid/**" <*luid*> <*extension*>
<*message-sent-unique-indexed-object*> ::= <*common-vmessage*> | <*other-object-format*>

<*message-incoming-change-counter-object-name*> ::= "**telecom/msg/in/luid/cc.log**"
<*message--incoming-change-counter-object*> ::= <*change-counter-object*>
<*message-outgoing-change-counter-object-name*> ::= "**telecom/msg/out/luid/cc.log**"
<*message- outgoing-change-counter-object*> ::= <*change-counter-object*>
<*message-sent-change-counter-object-name*> ::= "**telecom/msg/sent/luid/cc.log**"
<*message-sent-change-counter-object*> ::= <*change-counter-object*>

<*message-incoming-change-log-object-name*> ::= "**telecom/msg/in/luid/**" <*change-counter*> "**.log**"
<*message--incoming-change-log-object*> ::= <*change-log-object*>
<*message-outgoing-change-log-object-name*> ::= "**telecom/msg/out/luid/**" <*change-counter*> "**.log**"
<*message- outgoing-change-log-object*> ::= <*change-log-object*>
<*message-sent-change-log-object-name*> ::= "**telecom/msg/sent/luid/**" <*change-counter*> "**.log**"
<*message-sent-change-log-object*> ::= <*change-log-object*>

9.9 Message Object Definition

The IrMC 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 9.9.1. The property values are expressed as property strings. The unique identifier property, if present, must always be stored.

The formal definition of the vMessage is introduced in section 9.9.2.

9.9.1 Property Identifiers

9.9.1.1 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 1.1 to correspond to this specification.

VERSION:1.1

9.9.1.2 Encoding, Character Set and Language

Encoding, character set and language parameters, as defined in the vCard 2.1 Specification 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).

9.9.2 Formal Definition of vMessage 1.1

<*vmessage-object*> ::= {
"**BEGIN:VMSG**" <CRLF>
<*vmessage-property*>*
[<*vmessage-originator*>]*
<*vmessage-envelope*>
"**END:VMSG**" <*CRLF*>
}

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

<*vmessage-envelope*> ::= {
"**BEGIN:VENV**" <*CRLF*>
{
[<*vmessage-recipient*>]*
<*vmessage-envelope*> * | <*vmessage-content*>
}
"**END:VENV**" <*CRLF*>
}
<*vmessage-recipient*> ::= <*vcard*>

<*vmessage-content*>::= {
"**BEGIN:VBODY**"<*CRLF*>
<*vmessage-body-property*>*
<*vmessage-body-content*>
"**END:VBODY**"<*CRLF*>

}

<*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-body-content*>::='message as specified in RFC822 and MIME RFC2045/RFC2047'

<*vmessage-property*>::=<*vmessage-version-property*> * X-IRMC-STATUS * X-IRMC-TYPE

<*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 must be provided in the following way:

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

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

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

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

vMessage Example: Figure 9-3 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.

<vmessage-object>

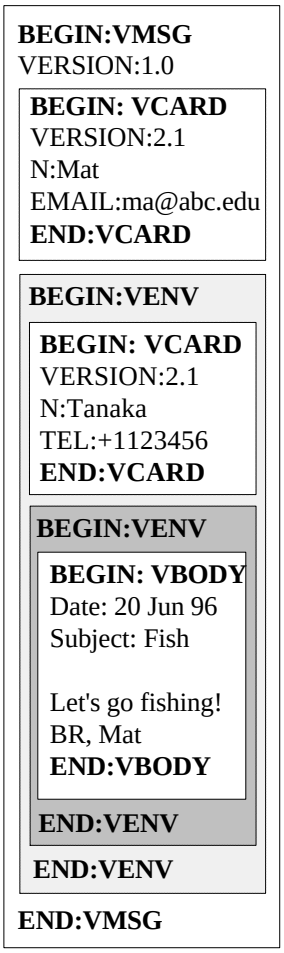


Figure 9-3 A vMessage with one recipient

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

```
BEGIN:VMSG                                  // open message
    VERSION:1.1                             // 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
```

9.10 vMessage Support with IrMC

9.10.1 Mandatory Field Support

A vMessage message object must support Version and vBody..

VMessage field	Property Name	Description
Version	VERSION	Version of the vMessage specification used in the implementation.
Message Body	BEGIN:VBODY END:VBODY	Message as specified in RFC822 or RFC 2045/2047

9.10.2 Formal Definitions as required by vMessage

<*extension*> ::= "**.vmg**"

9.10.3 Read Indication for vMessages

In order to mark messages as having been read, a status extension is defined for vMessage. The status extension property is X-IRMC-STATUS and will contain one of the following values:

Property Value	Description
UNREAD	Indicates the message has not been read
READ	Indicates the message has been read

9.10.4 Additional Information Log property

The maximum size of the <vmessage-body-content> is specified as an additional property in the Information Log (defined in section 2.9.16) for incoming, outgoing and sent messages.

The property VBODY is used to indicate the maximum message size. For example:

VBODY:=200

indicates that the maximum message size is 200 bytes.

If the receiving device can dynamically store very large messages, then the wildcard can be used.

VBODY:=*

9.10.5 Object Deletion for vMessage

In order to provide for deletion of specific vMessage objects, it is necessary to write back an empty object. In reference to the vMessage stream object example in figure 9.2, we are presented with the following data stream:

<message-outgoing-stream-object>
BEGIN:VMSG
VERSION:1.1
BEGIN:VCARD
VERSION:2.1
N:Harry
TEL:555-1234
END:VCARD
BEGIN:VENV
BEGIN:VBODY
Subject: Reminder

Don’t forget to buy milk!
END:VBODY
END:VENV
END:VMSG

BEGIN:VMSG
VERSION:1.1
BEGIN:VCARD
VERSION:2.1
N:Harry
TEL:555-1234
END:VCARD
BEGIN:VENV
BEGIN:VCARD
VERSION:2.1
N:Matti
TEL:+376254
END:VCARD
BEGIN:VENV
BEGIN:VBODY
From:Harry@office
Subject: Meeting

2PM is ok!
END:VBODY
END:VENV
END:VENV
END:VMSG

To delete the first VMSG object within the datastore presented above, one would write the object stream as follows:

<message-outgoing-stream-object>
BEGIN:VMSG
VERSION:1.1
END:VMSG

BEGIN:VMSG
VERSION:1.1
BEGIN:VCARD
VERSION:2.1
N:Harry
TEL:555-1234
END:VCARD
BEGIN:VCARD
VERSION:2.1
N:Matti
TEL:+376254
END:VCARD
BEGIN:VENV
BEGIN:VBODY
From:Harry@office
Subject: Meeting

2PM is ok!
END:VBODY
END:VENV
END:VENV
END:VMSG

The writing of the stream record in this format, would delete the 1st VMSG object in the target datastore.

Similarly for the deletion of a VMSG Index Object, we are presented the following index object entries as presented in Figure: 9.2:

<message-incoming-index-object> entries
BEGIN:VMSG
VERSION:1.1
BEGIN:VCARD
VERSION:2.1
N:Harry
TEL:555-1234
END:VCARD
BEGIN:VENV
BEGIN:VBODY
Subject: Reminder

Don’t forget to buy milk!
END:VBODY
END:VENV
END:VMSG

BEGIN:VMSG
VERSION:1.1
BEGIN:VCARD
VERSION:2.1
N:Harry
TEL:555-1234

END:VCARD
BEGIN:VENV
BEGIN:VCARD
VERSION:2.1
N:Matti
TEL:+376254
END:VCARD
BEGIN:VENV
BEGIN:VBODY
From:Harry@office
Subject: Meeting

2PM is ok!
END:VBODY
END:VENV
END:VENV
END:VMSG

To delete the first VMSG object within the datastore presented above, one would write the 1st object index as follows:

<message-incoming-index-object> entries
BEGIN:VMSG
VERSION:1.1
END:VMSG

The writing of the 1st record presented above would delete the 1st VMSG object in the target datastore.

9.10.6 Message Type Indication for vMessage

The message in vBody is specified to conform to RFC822. In the RFC822 specification, there are several mandatory fields: dates, source and destination (e.g. “Date”, “from” and “to”) in the header. In particular, the destination field must specify at least one e-mail type address. But, in some mobile communication services like paging services and short message services, some of this information (e.g. e-mail address, date) can’t be handled. Therefore, the values of the RFC822 mandatory header fields can’t be set in vBody on this type communication.

To solve this, a type extension is defined for vMessage. The type extension property is X-IRMC-TYPE and will contain one of the following values:

Property Value	**Description**
INET	Indicates the message is an e-mail type, and the vBody will completely conform to the RFC822 or MIME Type
MSG	Indicates the message is a kind of message service type, and the vBody won’t completely conform to the RFC822 and will omit some or none of the RFC822 header fields.

Both vMsg INET and MSG types may coexist within the existing data stores of inbox, outbox, and sentbox, as represented by the type extension property of X-IRMC-TYPE, in the same way that VEVENT and VTODO types coexists within the vCAL data store." This allows an application to store both internet email messages as well as non-email format messages.

INET means RFC822/RFC2045.

MSG can mean any format of data. There are no specifications relating to the format of the message or the header, apart from the body text begins after the blank line (i.e. just a CRLF).

10. Notes

10.1 Notes Overview

The Notes application provides a means for a user to manage small notes or messages, and to exchange those cards with other IrMC Devices.

The level of Information Exchange supported by the Notes application must be identified by the IAS NotesSupport parameter as described in section 13.1.3.1 and must be identified in the Notes Information Log.

The Notes application may store its objects in any internal format that is chosen by the implementer. However, the Notes application may support more than one format for Information Exchange. The type of objects supported, e.g., vNote 1.1, must be identified in the NOTE-TYPE property of the Device Information Object. At a minimum, Notes applications must support Information Exchange using the vNote 1.1 Object format. Furthermore, the examples in this chapter use vNotes.

Information about the Notes application is stored in the object named <*notes-information-log-object-name*>. The format of that object is <*notes-information-log-object*>.

10.2 Notes Level 1 Information Exchange

The minimum implementation of a Notes application provides a simple note “push” functionality. The name of the pushed object is specified as <*notes-minimum-support-object-name*>, and the object as <*notes-minimum-support-object*>.

Devices with an IrMC Notes application, regardless as to whether they provide connection-oriented services, are required to provide server support for Level 1 Information Exchange of Notes objects.

Level 1 Notes Example:

Local Device **PUT** [name – mynote.vnt]
Remote Device

device inbox:

mynote.vnt

optional UI indication "Do you want to save this note in your Notepad.? "

10.3 Notes Level 2 Information Exchange

This Object Store is named <*notes-stream-object-name*> and its format is defined as <*notes-stream-object*>. Entries with restricted access are also listed as empty Objects.

Static Indexing Notes
If the Notes Application supports Static Indices, the organization of the stream in the <*notes-stream-object*> must reflect the organization of the single <*notes-indexed-object*> Objects.

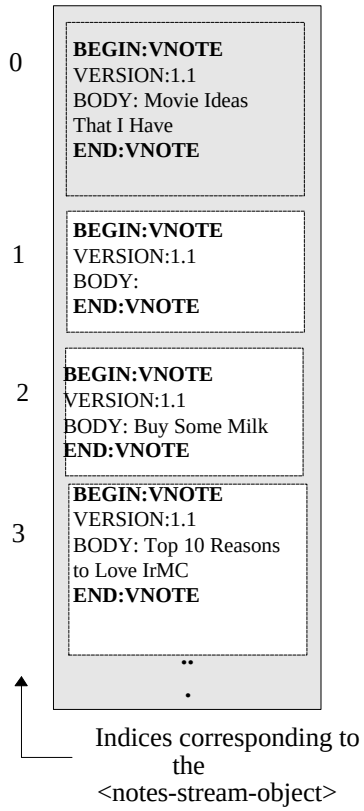


Figure 10-1 A Notes Stream With an Empty Object

When reading or writing all the Objects of a Notes Application with Static Indexing, all empty locations must be indicated by an empty Object as shown in the **Figure 10-1 A Notes Stream with An Empty Object**. This way it is possible to maintain the original location of the Objects in the Object Store. If empty Objects (as shown in **Figure 10-2 An Empty Notes Object**) are indicated these have to be taken into account by the receiver. If no new Notes Objects are made, the relative order of the entries remains the same in a Level 2 GET/PUT operation.

empty vNote

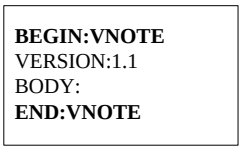


Figure 10-2 An Empty Notes Object

The <*notes-stream-object*> may be deleted by writing an empty entry on top of the original entry, i.e., by applying an OBEX PUT operation without a BODY to the original object.

Level 2 Notes Example:

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

10.4 Notes Level 3 Information Exchange

Each Notes Object is assigned a static index. Each Object is named as <*notes-indexed-object-name*>. The format of these objects is defined as <*notes-indexed-object*>.

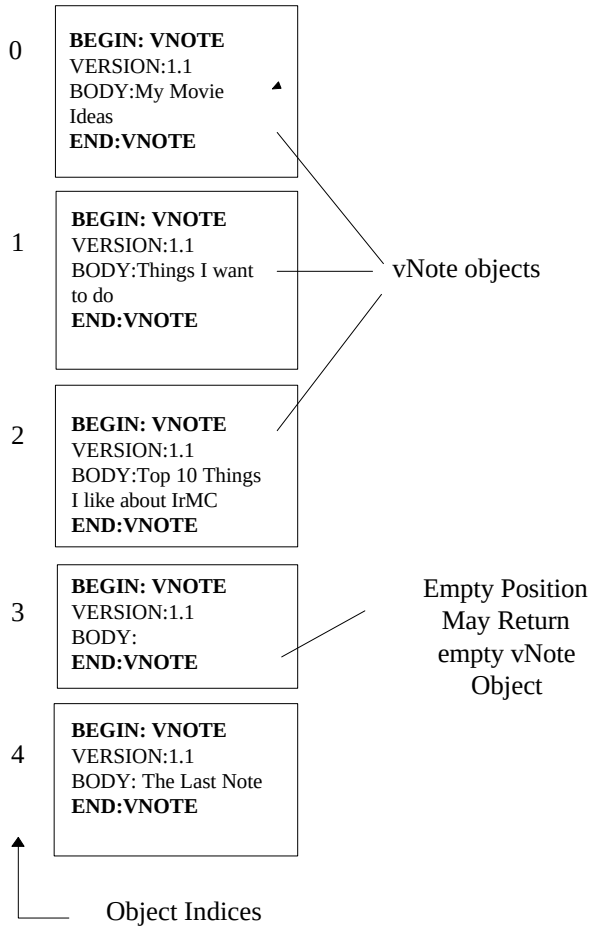


Figure 10-3 vNote entries with indices

These devices have the Notes entries organized as <*notes-indexed-object*>. 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.

Notes Index Support Example:

Local Device	**GET**	[name - telecom/nt/5.vnt]
Remote Device	**GET response**	[response code - unauthorized]

Local Device	**GET**	[name - telecom/nt/4.vnt]
Remote Device	**GET response**	[response code - success body - telecom/nt/4.vnt]
Local Device	**EDIT 4.vnt**	
	PUT	**[**name - telecom/nt/4.vnt]

10.5 Notes Level 4 Information Exchange

For synchronizing Notes of different devices, information about any changes or additions in the Notes Object Store is maintained in a Change Log <*notes-change-log-object-name*>. The format of that log is <*notes-change-log-object*>.

Each Notes Object is assigned a unique index, or LUID. Each Object is named as <*notes-unique-indexed-object-name*>. The format of these objects is defined as <*notes-unique-indexed-object*>.

Furthermore, if a Change Counter Sync Anchor is maintained, then the last used value of the Change Counter is stored in an object named <*notes-change-counter-object-name*>. The format of that Change Counter is <*notes-change-counter-object*>.

Detailed examples of Level 4 Information Exchange are given in Chapter 5 Synchronization.

10.6 Formal Definition of Notes Objects

10.6.1 Information Log

<*notes-information-log-object-name*> ::= "**telecom/nt/info.log**"

<*notes-information-log-object*> ::= <*information-log*>

10.6.2 Notes Minimum Support

<*notes-minimum-support-object-name*> ::= <*default-char-not-lf*>$^{+}$ <*extension*>

<*notes-minimum-support-object*> ::= <*common-vnote*> | <*other-object-format*>

10.6.3 Notes Access Support

<*notes-stream-object-name*> ::= "**telecom/nt"** <*extension*>

<*notes-stream-object*> ::= <*common-vnote*>* | <*other-object-format*>*

10.6.4 Notes Index Support

<*notes-indexed-object-name*> ::= "**telecom/nt/**" <*digit*>$^{+}$ <*extension*>

<*notes-indexed-object*> ::= <*common-vnote*> | <*other-object-format*>

10.6.5 Notes Synchronization Support

<*notes-unique-indexed-object-name*> ::= "**telecom/nt/luid/**" <*luidt*> <*extension*>

<*notes-unique-indexed-object*> ::= <*common-vnote*> | <*other-object-format*>

<*notes-change-counter-object-name*> ::= "**telecom/nt/luid/cc.log**"

<*notes-change-counter-object*> ::= <*change-counter*>

<*notes-change-log-object-name*> ::= "**telecom/nt/luid/**" <*change-counter*> "**.log**"

<*notes-change-log-object*> ::= <*change-log-object*>

10.7 vNote Object Format Support

All IrMC Notes applications, must support Information Exchange using the vNote Object Format as defined in this chapter, section 10.7.3. The vNote objects are identified by the VNT extension in the NAME.

10.7.1 Mandatory Fields

All of the mandatory property fields of vNote, i.e. *Body* and *Version*, must be present. It should be noted that even though the Body identifier is always required in all vNote objects, the Body field value may be left empty.

Received vNotes 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.

vNote field	Property Name	Description
Body	BODY	The text of the Note
Version	VERSION	Version number of the vNote specification used in the implementation.

10.7.2 Formal Definitions as required by vNote 1.1

<*extension*> *::=* "**.vnt**"

10.7.3 Formal Definition of vNote 1.1

<*vnote-object*> ::= {

"**BEGIN:VNOTE**" <*CRLF*>

"**VERSION:**" <*vnote-version*> <*CRLF*>

["**X-IRMC-LUID:**" <*luid*> <*CRLF*>]

["**DCREATED:**" <*common-date*> <*CRLF*>]

["**LAST-MODIFIED:**" <*common-date*> <*CRLF*>]

["**SUMMARY**" <*vcard-properties*>* "**:**" *default-char-not-lf*>* <*CRLF*>]

"**BODY**" <*vcard-properties*>* "**:**" <*default-char-not-lf*>* <*CRLF*>

["**CATEGORIES:**" <*default-char-not-lf*>* <*CRLF*>]

["**CLASS:**" <*private-flag*> <*CRLF*>]

"**END:VNOTE**" <*CRLF*>

}

<*vnote-version*> ::= "**1.1**" ; only currently supported version is 1.1

<*private-flag*> ::= "**PRIVATE**" | "**PUBLIC**" | "**CONFIDENTIAL**"

<*vcard-properties*> ::= "**;**" <*vcard-property-identifier*>

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

11. 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.

11.1 Command Syntax and Character Set

11.1.1 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 must be done on the basis of the recommended default setting of the subparameter.

11.1.2 Command Syntax

(1) COMMAND LINE

Figure 11-1 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.

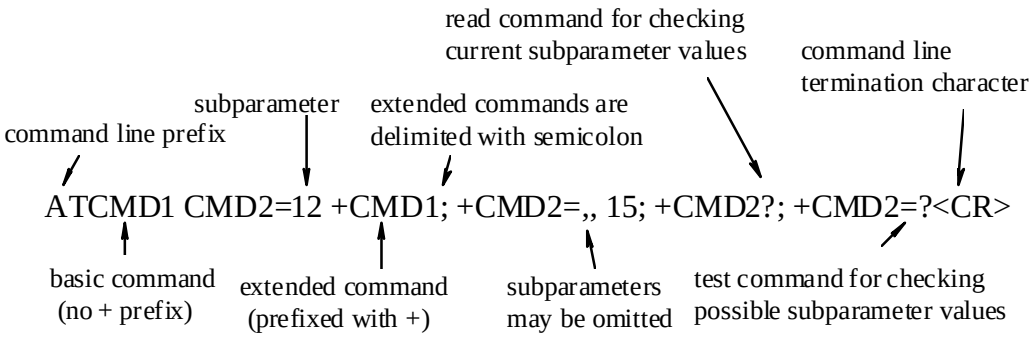


Figure 11-1 Basic structure for a command line

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 11-1 could be as shown in Figure 11-2. 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>.

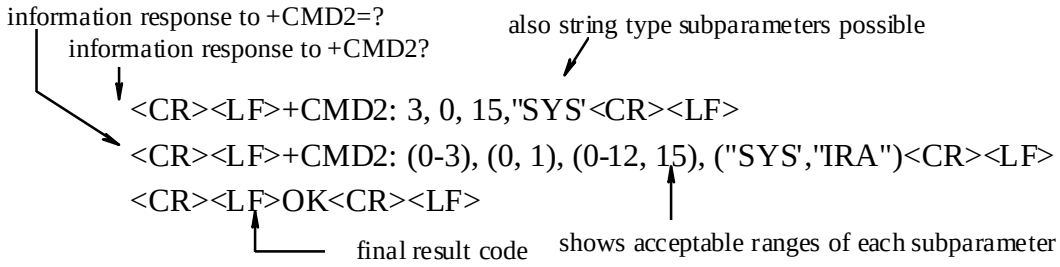


Figure 11-2 Response to a command line

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)

11.1.3 Character Sets

International reference alphabet (ITU-T T.50 IRA:7bit code) is used for control commands and responses. When 7bit code is used, the most significant bit of each word in IrLAP frame shall be set to zero. For the entries in phone book the character set selected by the command + CSCS is used.

(1) Select TE character set
(i) Command:+ CSCS

+CSCS parameter command syntax

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

(ii) Description
Set command informs ME which character set <chset> is used by the TE. Read command shows current setting and test command displays the character sets supported by ME.

(iii) Defined values
<chset> (conversion schemes not listed here can be defined by manufactures):
"IRA" international reference alphabet (ITU-T T.50)
(mandatory, default setting for all IrMC devices)
"GSM" GMS default alphabet (GSM 03.38 subclause 6.2.1)
"2022-JP" ISO-2022-JP character set

"SJIS" Shift-JIS
"PCCPxxx" PC character set code page xxx
"PCDN" PC Danish/Norwegian character set
"UCS2" 16-bit universal multiple-octet coded character set (ISO/IEC 10646);UCS2 character strings are converted to hexadecimal numbers from 0000 to FFFF; e.g., "004100620063"equals three 16-bit characters with decimal values 65, 98, 99, $(AT R97)$
"8859-n" ISO 8859 Latin n(1-6) character set
"8859-C" ISO 8859 Latin/Cyrillic character set
"8859-A" ISO 8859 Latin/Arabic character set
"8859-G" ISO 8859 Latin/Greek character set
"8859-H" ISO 8859 Latin/Hebrew character set
"HEX" character strings consist only of hexadecimal numbers from 00 to FF;
e.g."032FE6" equals three characters with decimal values 3,47 and 230

NOTE: If ME is seeing GSM default alphabet , its characters shall be padded with 8th bit (zero) before converting them to hexadecimal numbers (i.e., no SMS-style packing of 7-bit alphabet)

(vi) Implementation
Mandatory when a command using the setting of this command is implemented.

11.2 Common Command Set

11.2.1 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 IrMC 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 IrMC Specification.

(iv) Implementation
Mandatory.

11.2.2 Generic Commands

Table 11.2.2-1 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 11.2.2-1 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

11.2.3 TE-ME Interface Commands

Table 11.2.3-1 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 11.2.3-1 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

11.2.4 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 ’voic

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 7.6.1 7.7.6.1)

"DC" ME dialed calls (outgoing call, refer to section 7.6.2 7.7.6.2)

"MC" ME missed calls (refer to section 7.6.3 7.7.6.3)

"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. Character set should be selected using +CSCS.

<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 11.2.4-1 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 11.2.4-1 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)

11.2.5 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 selected with +CSCS.

<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 selected with +CSCS.

<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.

11.2.6 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, IRDA OBEX 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. Voice data transmission is prohibited after ME sends the response +CTALK.

(iii) Defined value
<n> 0 disable

1 enable (Prohibited)

(iv) Implementation
Optional.

11.2.7 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 < 0.3%
	1	0.3…1%
	2	1...3%
	3	3% < BER
	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" bar incoming calls from numbers Not stored to ME memory

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

"NA" bar 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. Each 4 bits in Device Nickname defined in IrLMP is hexadecimal coded into IRA characters (e.g., Device Nickname (Character Set is ASCII, and Name is “Abc”) is

(iv) Implementation
Optional.

(18) CHANGING THE AUDIO PATH AND THE RTCON STANDBY-TALK MODE FROM TE
(i) Command: +CMPC

+CMPC parameter command syntax

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

(ii) Description

Set command changes the audio path and the RTCON mode (STANDBY / TALK) whenever TE wants the audio transmission. There are two audio paths, which called internal audio path and external audio path. Internal audio path means both of microphone and speaker built in ME’s body. During the audio transmission using this audio path, no audio data is transmitted through RTCON. On the other hand, during the data transmission using external audio path, the audio data is transmitted though RTCON and microphone and speaker are not used. The usage case of this command is described below.

(1) While ME is talking (connecting to NW), RTCON connection gets established between ME and TE. Then, TE issues this command. After that, audio path is opened to TE, and RTCON changes to TALK. For example, when ME is put on a car dock while ME is talking, a 'handsfree' can be activated without terminating ME’s call. Or while ME is talking, a user can transmit conversation data to PC and store it there. This command is issued by TE in both cases.

(2) While ME is talking (connecting to NW) and the audio is transmitted by RTCON between ME and TE, TE issues this command. Then, audio path is closed, and RTCON changes to STANDBY. After that, ME gets stand alone with keeping connection to NW. For example, when ME is taken away from a car dock during talking by 'handsfree', ME can continue to talk by itself. This command is issued by TE.

(3) Even if ME is not talking, RTCON connection can be established between ME and TE. Then, TE issues this command. After that, audio path is opened to TE, and RTCON changes to TALK. For example, ME can act as just a speaker to play back audio data stored in PC. Or ME can act as just a microphone to transmit audio data to PC and store it there. This command is issued by TE in both cases.

(4) When ME receives “RING” that shows a call is coming, ME connects RTCON to TE (PC), and transmits “RING” there. Then, such an application as storing voice messages can issue this command and then issues answer command. After that, ME can change talking (connecting to NW), and audio data can be transmitted to the PC, there audio data (voice messages) can be stored.

Read command returns currently selected audio path and RTCON mode.

Test command returns supported audio path and RTCON mode.

(iii) Defined value

<n>: integer type value

0	Audio path is internal, and RTCON is on STANDBY mode.
1	Audio path is external, and RTCON is on TALK mode.
2	Audio path is both of internal and external, and RTCON is on TALK mode.

(iv) Implementation
Optional

(19) REQUEST HANDS-FREE FROM TE

(i) Command: +CRHFR

+CRHFR parameter command syntax

Command	Possible response(s)
+CRHFR=<p>,<m>	
+CRHFR?	+CRHFR:<p>,<m>
+CRHFR=?	+CRHFR: (list of supported <p>s), (list of supported <m>s)

(ii) Description

Set command changes the audio path and ME’s internal mode (Handheld / Car mount) whenever TE wants to do that. When ME is put on a car dock (TE), the car dock can issue this command. Then, ME can not only be changed the audio path, but also changed its internal mode between handheld and car mount. Once, ME is taken away from the cat dock, audio path returns to an initial setting and its internal mode is changed to handheld.

Read command returns currently voice path and mode.

Test command returns supported voice path and mode.

(iii) Defined value

<p>: integer type value indicates voice path

0:	Audio path is internal
1:	Audio path is external
2:	Audio path is both internal and external

<m>: integer types value indicates voice mode

0:	Handheld
1:	Car mount

(iv) Implementation
Optional

(20) REQUEST TO RECORD AUDIO DATA

(i) Command: +CRRCD

+CRRCD parameter command syntax

Command	Possible response(s)
+CRRCD=<a>,<m>	
+CRRCD?	+CRRCD: <a>,<m>
+CRRCD=?	+CRRCD: (list of supported <a>s), (list of supported <m>s)

(ii) Description

Set command make MT record audio data transmitted by RTCON. When TE issues this command, audio path is changed to external and RTCON is changed to TALK. Then, TE can transmit sound or voice data by a real-time audio transmission. ME can store such audio data after completing its transmission.

Read command returns currently record mode.

Test command returns supported record mode.

(iii) Defined value action, mode

<a>: integer type value indicates start / stop action

1: Start

0: End

<m>: integer type value indicates record mode

0: Ringer sound

1: Sound during holding a call

2: Voice message responds to incoming call

3: Voice recognition

4: Voice memo

others: reserved

(iv) Implementation

Optional

11.2.8 Mobile Equipment Error

11.2.8.1 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.

11.2.8.2 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.

11.2.9 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 RTCON 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 waiting tone (start timing of audio transmission) (for PDC)
14-20	Reserved

21 dial “1”
22 dial “2”
23: dial “3”
24: dial “4”
25: dial “5”
26: dial “6”
27: dial “7”
28: dial “8”
29: dial “9”
30: dial “0”
31: dial “*”
32: dial “#”
33: others - ***Note:*** *Others mean any keys except “0” ~ “9”, “*” and “#” on ME.*
34-80 Optional (operator/manufacturer specified tone.)

(iv) Implementation
Mandatory. Items 21-33 are optional.

(3) Indicate Changed Audio Path and RTCON STANDBY-TALK Mode
(i) Result code: +CMPCIND:<n>
(ii) Description
This code indicates changing audio path and RTCON mode (STANDBY / TALK) whenever ME wants audio transmission. There are two audio paths, which called internal audio path and external audio path. Internal audio path means both of microphone and speaker built in ME’s body. During the audio transmission using this audio path, no audio data is transmitted through RTCON. On the other hand, during the data transmission using external audio path, the audio data is transmitted though RTCON and microphone and speaker are not used. The usage case of this command is described below.

(1) While ME is talking (connecting to NW), RTCON connection gets established between ME and TE. Then, ME issues this code. After that, audio path is opened to TE, and RTCON changes to TALK. For example, when ME is put on a car dock while ME is talking, a 'handsfree' can be activated without terminating ME’s call. Or while ME is talking, a user can transmit conversation data to PC and store it there. This command is issued by ME in both cases.

(2) While ME is talking (connecting to NW) and the audio is transmitted by RTCON between ME and TE, ME issues this code. Then, audio path is closed, and RTCON changes to STANDBY. After that, ME gets stand alone with keeping connection to NW. For example, when ME is taken away from a car dock just before talking by 'handsfree', ME can continue to talk by itself. This command is issued by ME.

(3) Even if ME is not talking, RTCON connection can be established between ME and TE. Then, ME issues this code. After that, audio path is opened to TE, and RTCON changes to TALK. For example, ME can act as just a speaker to play back audio data stored in PC. Or ME can act as just a microphone to transmit audio data to PC and store it there. This command is issued by ME in both cases.

(iii) Defined value
<n>:integer value

0: Audio path is internal, and RTCON is on STANDBY mode.
1: Audio path is external, and RTCON is on TALK mode.
2: Audio path is both of internal and external, and RTCON is on TALK mode.

(iv) Implementation
Optional

(4) Indicate Prohibit Voice Transmission
(i) Result code: +CPVTIND:<n>
(ii) Description
This code indicates prohibit / resume audio transmission whenever ME wants to do that. When audio data is transmitted by RTCON between ME and TE while ME is talking (connecting to NW), ME issues this code if it wants to prohibit audio transmission. Then, RTCON changes to STANDBY with keeping the connection to NW. At this time, other LSAP connection (e.g. IrCOMM, IrOBEX) can be established. For example, ME can switch to non-audio transmission by IrCOMM from audio transmission during talking. Or ME can pause the audio

transmission if user wants, then vCard/vCalendar/vMessage data could be transmitted to the TE by IrOBEX, and the TE can read phone number/schedule/messages. After resuming operation, the audio transmission can continue. This command is issued by ME in both cases.

(iii) Defined value

<n>:integer value

0: enable
1: disable (prohibit)

(iv) Implementation
Optional

(5) Indicate Pre-report Reconnection
(i) Result code: +CPRRCN
(ii) Description
This code pre-reports the reconnection of IrLAP when ME wants to do that. When ME wants to reconnect IrLAP connection while RTCON connection is established, ME can issue this code before it disconnect. TE can know such an action before it is done. ME reconnects IrLAP, and RTCON connection just after it disconnects IrLAP.

(iii) Usage Example
Optional

(6) INDICATE PROHIBIT VOICE TRANSMISSION
(i) Result code: +CPVTIND:<n>
(ii) Description
This code indicates prohibit / resume audio transmission whenever ME wants to do that. When audio data is transmitted by RTCON between ME and TE while ME is talking (connecting to NW), ME issues this code if it wants to prohibit audio transmission. Then, RTCON changes to STANDBY with keeping the connection to NW. At this time, other LSAP connection (e.g. IrCOMM, IrOBEX) can be established. For example, ME can switch to non-audio transmission by IrCOMM from audio transmission during talking. Or ME can pause the audio transmission if user wants, then vCard/vCalendar/vMessage data could be transmitted to the TE by IrOBEX, and the TE can read phone number/schedule/messages. After resuming operation, the audio transmission can continue. This command is issued by ME in both cases.

(iii) Defined value

<n>:integer value

0: enable
1: disable (prohibit)

(iv) Implementation
Optional

(7) Indicate the Phone status Talk-Standby
This code is added to section 10.2.10, Responses.

(i) Result code: +CTALKIND:<n>
(ii) Description

This code indicates changing ME's status between standby and talk. When ME makes a call or receives a call, ME changes its status to talking from standby, which means ME is connecting to NW. At this time, ME issues this code. Then, TE can get this code, and reflect this information to a display. Likewise, when ME terminates a call or is disconnected a call, ME changes the status to standby from talking, which means ME is not connecting to NW. At this time, ME issues this code. Then, TE can reflect it to the display.

(iii) Defined value

<n>:integer value

1: talk
0: standby

(iv) Implementation
Optional

11.3 System Oriented Command Set

11.3.1 Control Commands for GSM

The following GSM 07.07 commands (and result codes related to them) from the IrMC 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.

11.3.2 Control Commands for PDC

11.3.2.1 Control Command

11.3.2.1.1 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

11.3.2.1.2 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 status 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 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.

(7) Request Non-Voice Communication Service (Corresponding serial signal: Request Non-Voice Communication Service)

(i) Command: +CRNV

+CRNV parameter command syntax

Command	Possible response(s)
+CRNV=<m>,<n>[,<csys>[,<cdis>[,<cser1>,<cser2>[,<chlc>,<cbc1>,<cbc2>[,<cta>]]]]	
+CRNV=?	

(ii) Description
This command requests the non-voice communication service and service type from TE (including modem) to ME.

Test command returns supported non-voice communication service.

(iii) Defined value
<m>: integer type value
- 0: ON
- 1: OFF

<n>: integer type value
- 0: FAX
- 1: NMP
- 3: TEL
- 4: Attribute of originating / answering call and type of ADP
- 5: System information
- 6: Display information
- 7: Service information

<csys>: integer type value indicates continuous data for System
- 1: V.24 type
- others: reserved

Note: *This value will be set only when <n>=5.*

<cdis>: string type value indicates continuous data for Service

Possible values:

Status of connection	Value
Before receiving XID (Connect)	“ASYNC”
MNP	“MNP 4” “MNP 5” “MNP 10”
V.42	“V.42” “V.42 bis”
Normal mode	“NORMAL”
PDC High Speed Data	“ASYNC D-D Comp” “ASYNC D-D”
D/V mode	“Voice & Data Comp” “Voice & Data”
Others	“CONNECT”

Note: This value will be set only when <n>=7.

<cser1>: integer type value indicates continuous data for Service

0: AT command, escape sequence is permitted.
1: AT command, escape sequence is prohibited.
others: reserved.

<cser2>: integer type value indicates continuous data for Service

0: Custom
1: MNP4
2: MNP5
3: MNP10
4: V.42
5: V.42bis
6: Normal Mode

<chlc>: integer type value indicates continuous data for High Layer Compatibility

0: Telephone
4: G2 / G3 FAX
33: Profile of document applications for G4 FAX
others: reserved.

Note: This value will be set only when <n>=4.

<cbc1>: integer type value indicates continuous data for Bearer Capability

0: 11.2 k
1: 5.6 k

***Note:** This value will be set only when <n>=4.*

<cbc2>: integer type value indicates continuous data for Bearer Capability

0: Voice
8: Unrestricted Digital
23: Voice + Data
others: reserved.

***Note:** This value will be set only when <n>=4.*

<cta>: integer type value, which consist of 6 digits: $X_5X_4X_3X_2X_1X_0$, indicates continuous data for Type of ADP

X_0= (0,1): 9600 bps functionality
X_1= (0,1): MNP functionality
X_2= (0,1): FAX functionality
X_3= (0,1): Service ON Request at originating a call
X_4= (0,1): Output request for attribute of answering a call
X_5= (0,1): System indication request after FAX / MNP-ON

***Note:** This value will be set only when <n>=4. Each digit should be set as “1” when <cta> has meaning of itself.*

(iv) Implementation

Optional

11.3.2.1.3 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 not designated)

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

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

9 roaming mode 2 (Network identity in group is designated)

10 roaming mode 3 (Network identity in group is 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)

11.3.2.2 Unsolicited Result Code

11.3.2.2.1 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.

(7) Indicate Incoming Number
(i) Result code: +CINCNIND:<number>
(ii) Description
This code indicates the Incoming Number. When ME gets an incoming call which includes an answering device's telephone number (incoming number), ME can issue this code. At this time, if TE is assigned identifiable number and it is same as the incoming number, TE can know the call should be terminated on itself. Then, TE can automatically answer to this call. This is useful if TE is a fax machine.

(iii) Defined value
<number>: string type phone number
(iv) Implementation
Optional

(8) Indicate Originating Number
(i) Result code: +CRONIND:<number>
(ii) Description
This code indicates the Originating Number. When ME makes a call, ME can issue this code. With the receipt of this command, TE can know the telephone number of the originator.

(iii) Defined value
<number>: string type phone number
(iv) Usage Example
Optional

(9) Indicate Non-Voice Communication Service (Corresponding serial signal: Confirm Non-Voice Communication Service)
(i) Result code:
+CSNV:<m>,<n>[,<csys1>,<csys2>[,<cdis>[,<cser1>,<cser2>[,<chlc>,<cbc1>,<cbc2>[,<ctad>]]]]]
(ii) Description
This code responds to the "Request Non-Voice Communication Service" command. The contents of this response are the class of service, system indication and the other indications.

(iii) Defined value
<m>: same as parameter in command: +CRNV
<n>: same as parameter in command: +CRNV
<csys1>: integer type value indicates continuous data for System
1: High-speed data
other: reserved
<csys2>: integer type value indicates continuous data for System

1: Communication mode
2: Standby mode
3: Other mode
other: reserved

<cdis>: same as parameter in command: +CRNV
<cser1>: same as parameter in command: +CRNV
<cser2>: same as parameter in command: +CRNV
<chlc>: same as parameter in command: +CRNV
<cbc1>: same as parameter in command: +CRNV
<cbc2>: same as parameter in command: +CRNV
<ctad>: integer type value, which consist of 6 digits: $X_5X_4X_3X_2X_1X_0$, indicates continuous data for Type of ADP

X_0= (0,1): 9600 bps functionality
X_1= (0,1): MNP functionality
X_2= (0,1): FAX functionality
X_3= (0,1): Service ON Request at originating a call
X_4= (0,1): Output request for attribute of answering a call
X_5= (0,1): System indication request after FAX / MNP-ON

Note: *This value will be set only when <n>=4. Each digit should be set as "1" when <cta> has meaning of itself.*

(iv) Implementation
Optional

(10) Indicate Control of Non-Voice Communication Service (Corresponding serial signal: Non-Voice control signal)

(i) Result code: +CCNV:<s>
(ii) Description
This result code indicates the existence of adapter (including modem) to TE. This command is issued when ME detects the existence of adapter.

(iii) Defined value
<s>: integer type value
0: ME has no information for ADP.
1: ME has information for ADP.
(iv) Implementation

Optional

11.3.2.2.2 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.

12. Audio

12.1 Audio Transmission Overview

The scope of audio transmission architecture is shown below.

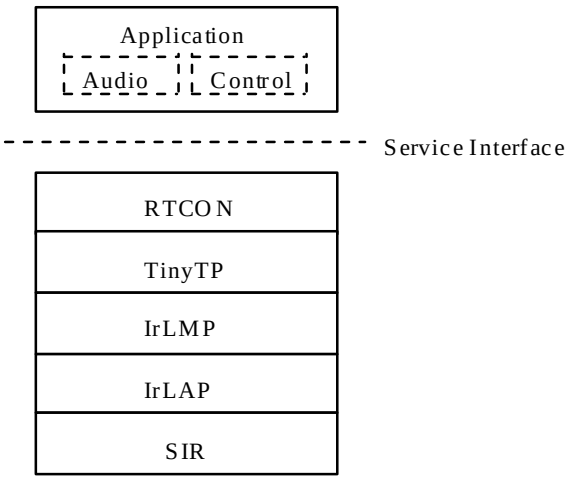


Figure 12-1 Audio Transmission Architecture

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.

12.2 Transmission Operation

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

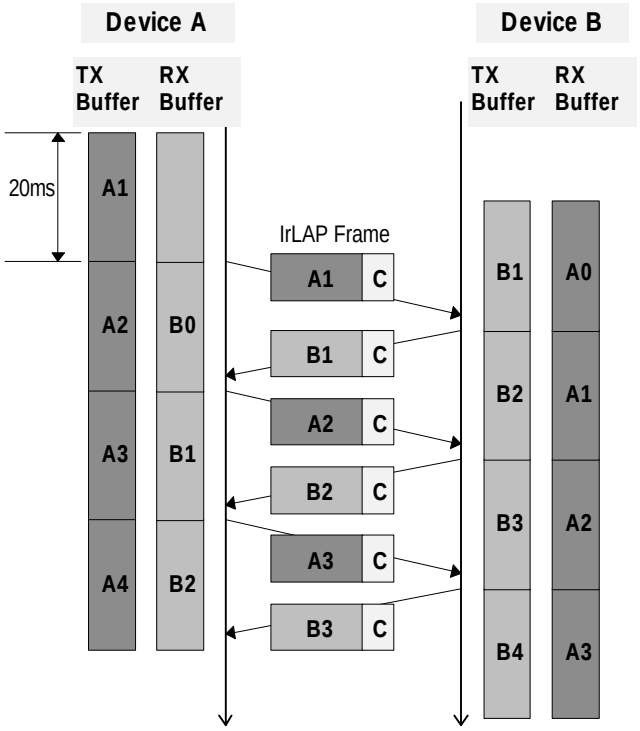


Figure 12-2 Normal Transmission Operation

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

12.2.1 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.

12.2.2 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.

12.2.3 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 12.6.1, in order to correct the IrLAP role.

12.2.4 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.

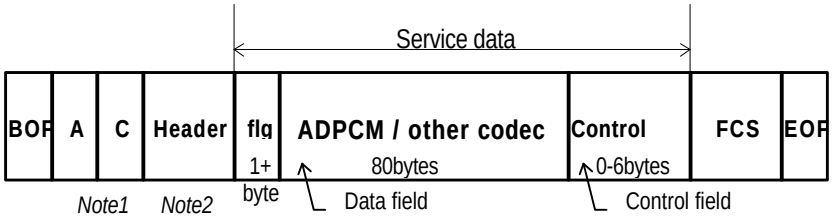
12.3 Frame Format

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 12-3

Flag (Flg) field

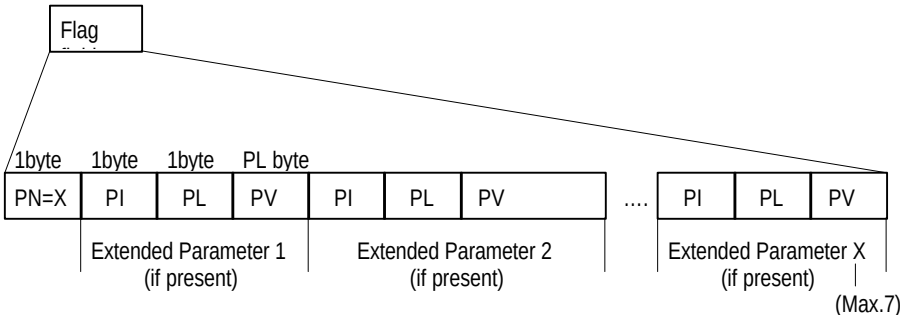


Figure 12-4

- 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

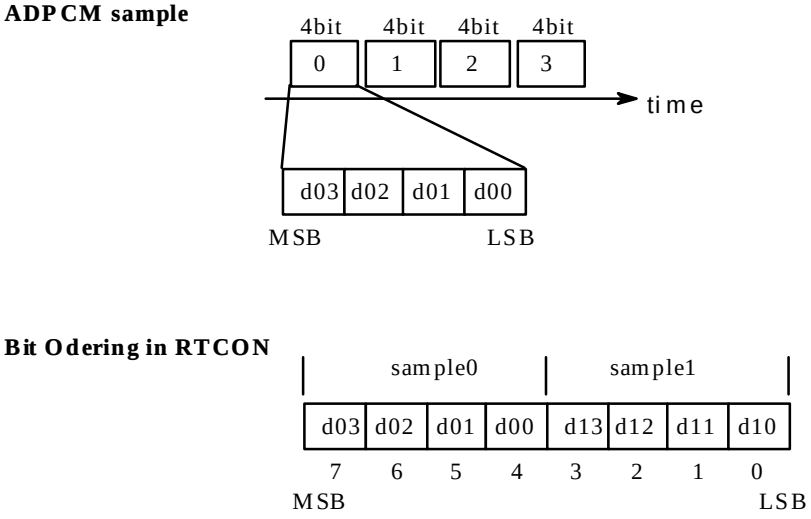
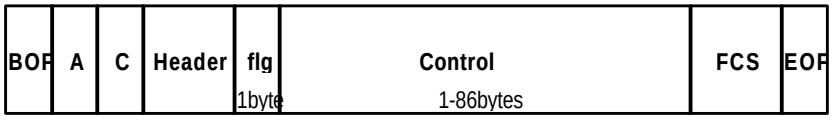


Figure 12-5

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

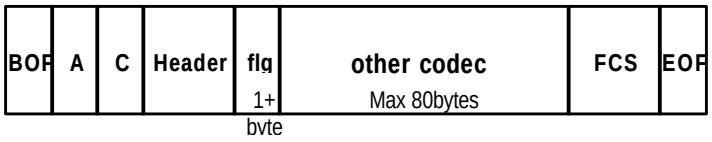


- Control Field: 1 ~ 86 bytes (variable)

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

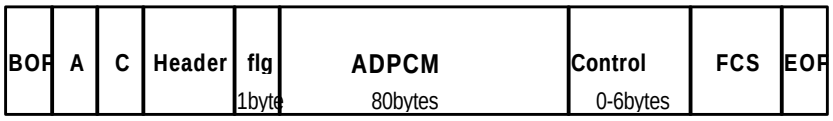
- DATA Field: 80 bytes (Fixed), 32 kbps ADPCM

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



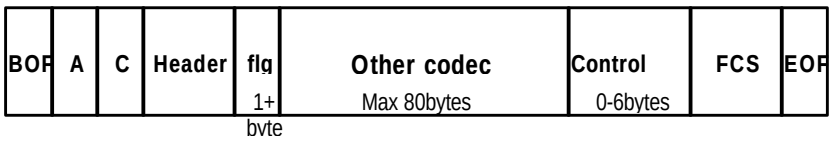
- DATA Field: maximum 80 bytes (dependent on codec or implementation)

(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)

12.4 Service Interface Definition

12.4.1 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

12.4.2 Disconnect Service

RTCON_Disconnect.request()
RTCON_Disconnect.indication()

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

12.4.3 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).

12.4.4 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).

12.4.5 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).

12.4.6 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.

12.4.7 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

12.5 State Chart

12.5.1 Primary State

12.5.1.1 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.

12.5.1.2 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.

12.5.2 Secondary State

12.5.2.1 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 */

12.5.2.2 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.

12.6 Service Sequence Example

12.6.1 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.

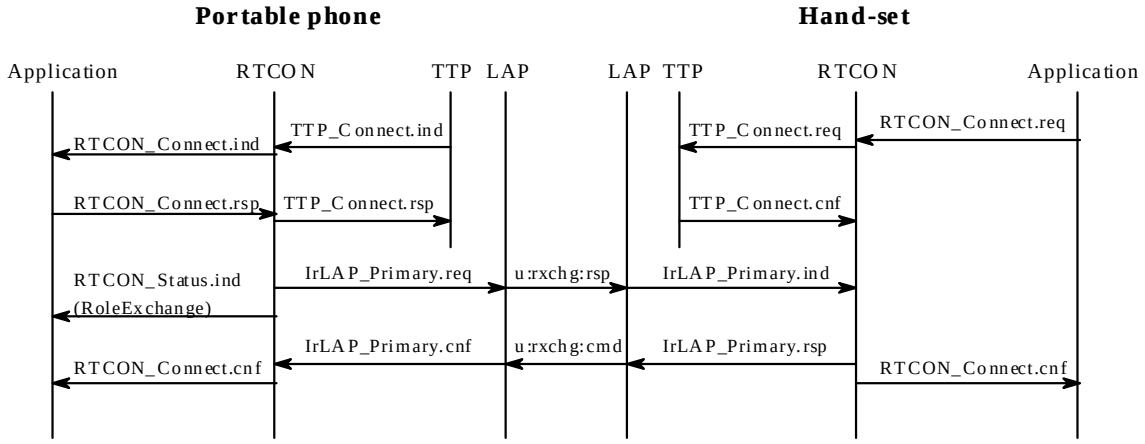


Figure 12-6

Case2: Hand-set does not support IrLAP RoleExchange service.

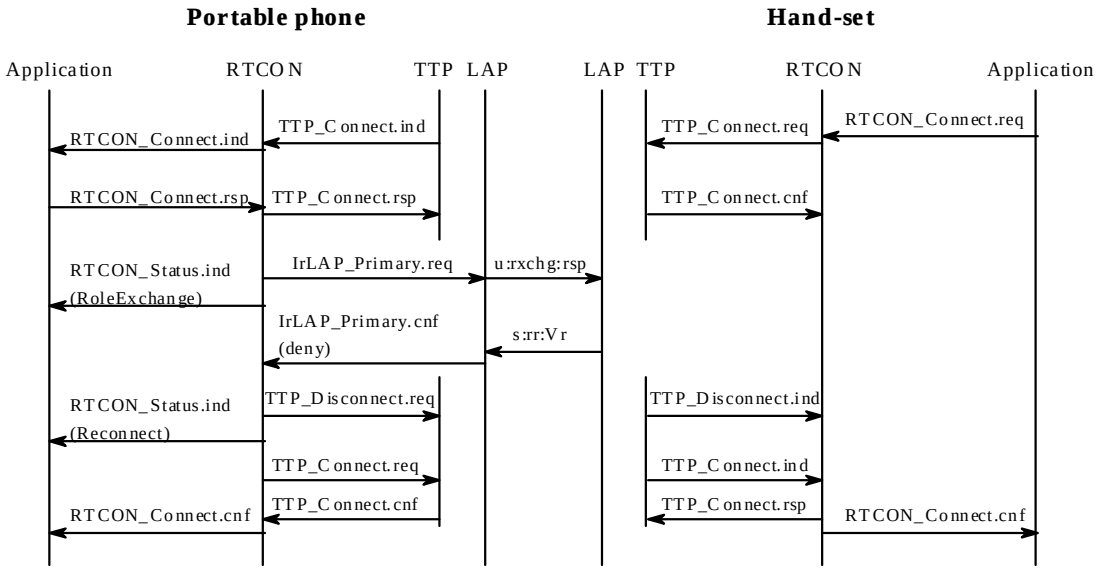


Figure 12-7

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**).

12.6.2 State Change from Standby to Talk

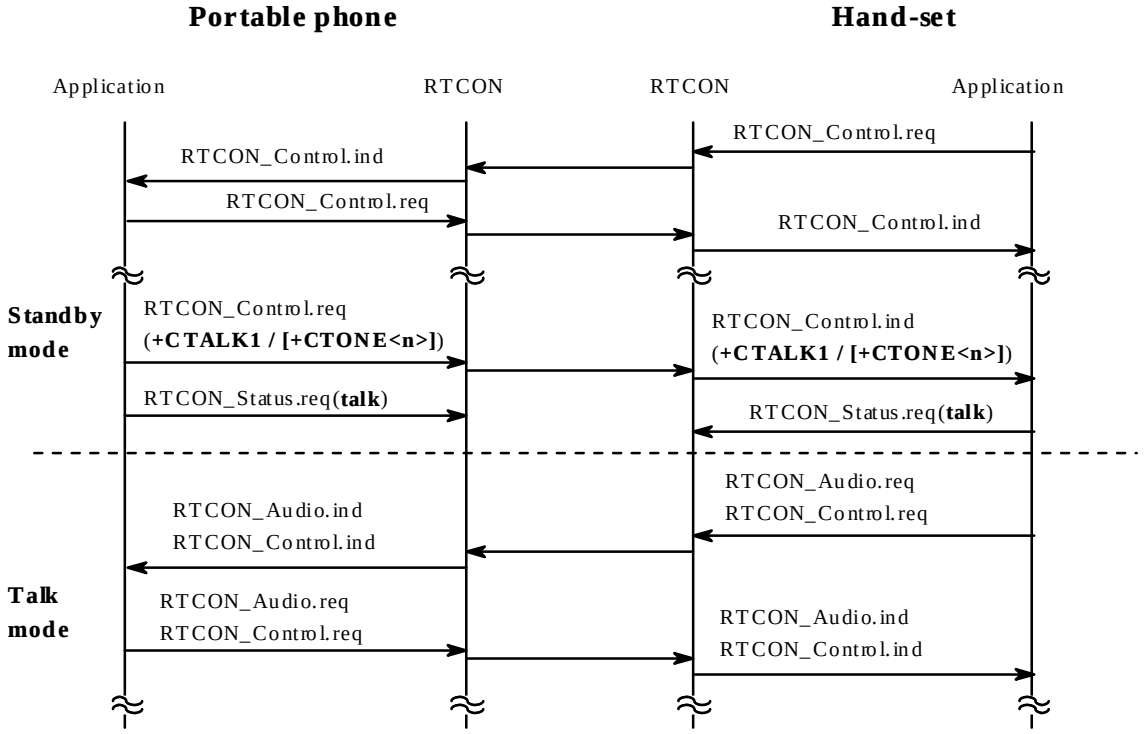


Figure 12-8

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

12.6.3 State Change from Talk to Standby

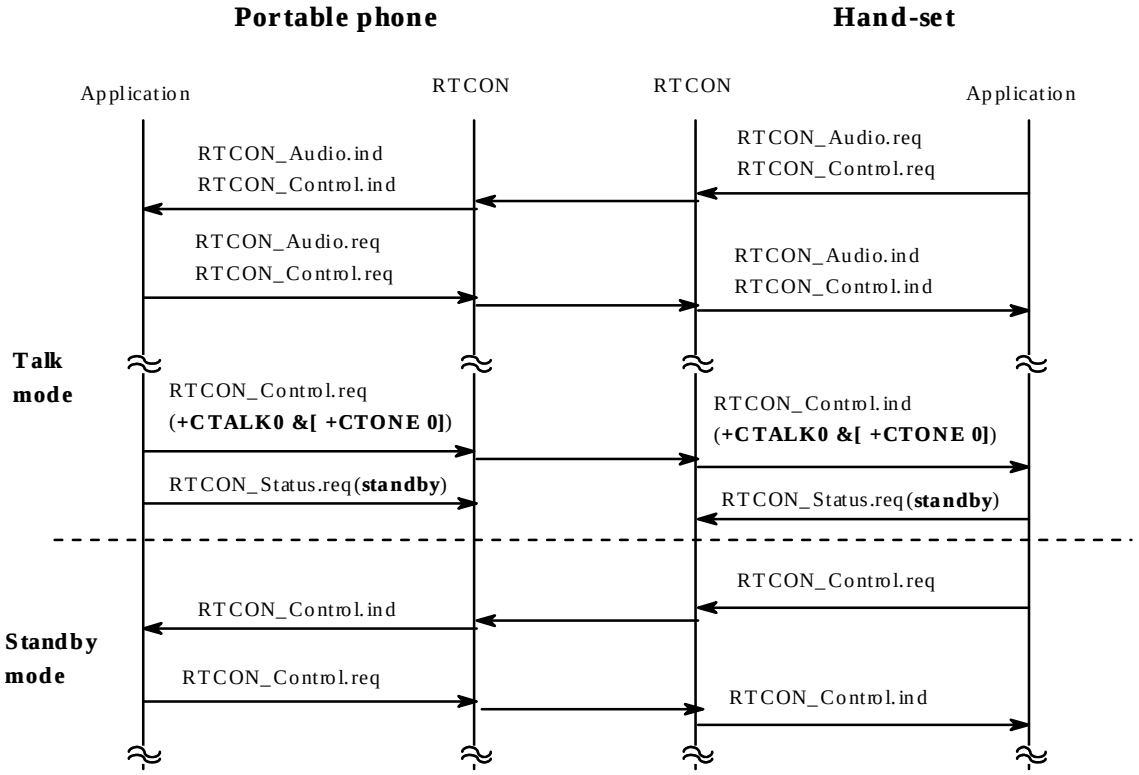


Figure 12-9

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.

12.6.4 Codec Mode Change

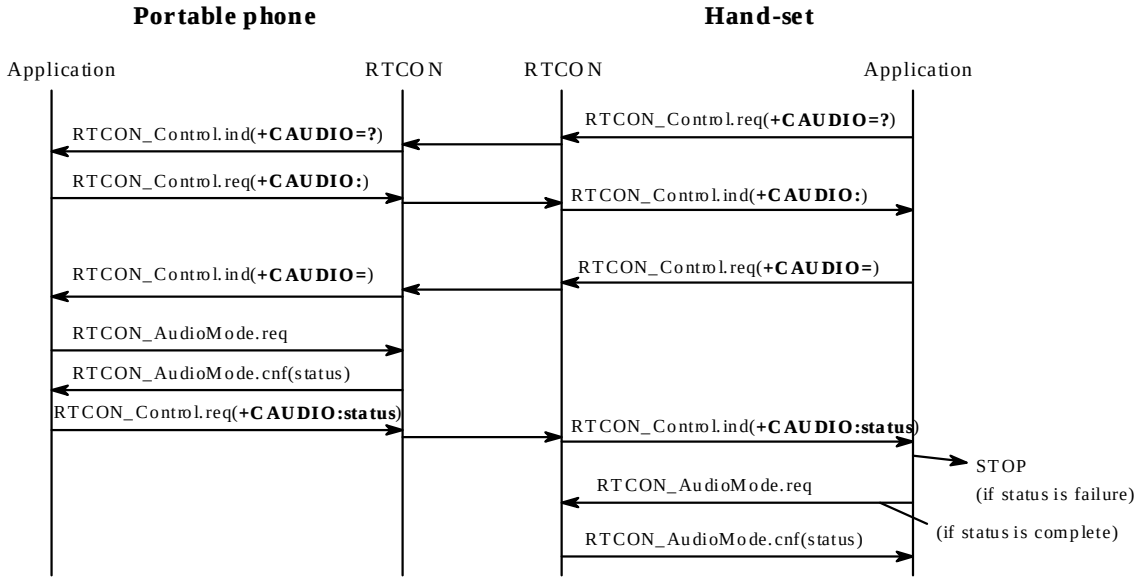


Figure 12-10

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

12.7 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.

12.7.1 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.

12.7.2 Recommendation for Implementation Type

If RTCON cannot be implemented as both the primary and the secondary procedures within an IrMC 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 12.6.1 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.

12.7.3 Simple Implementation

12.7.3.1 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)

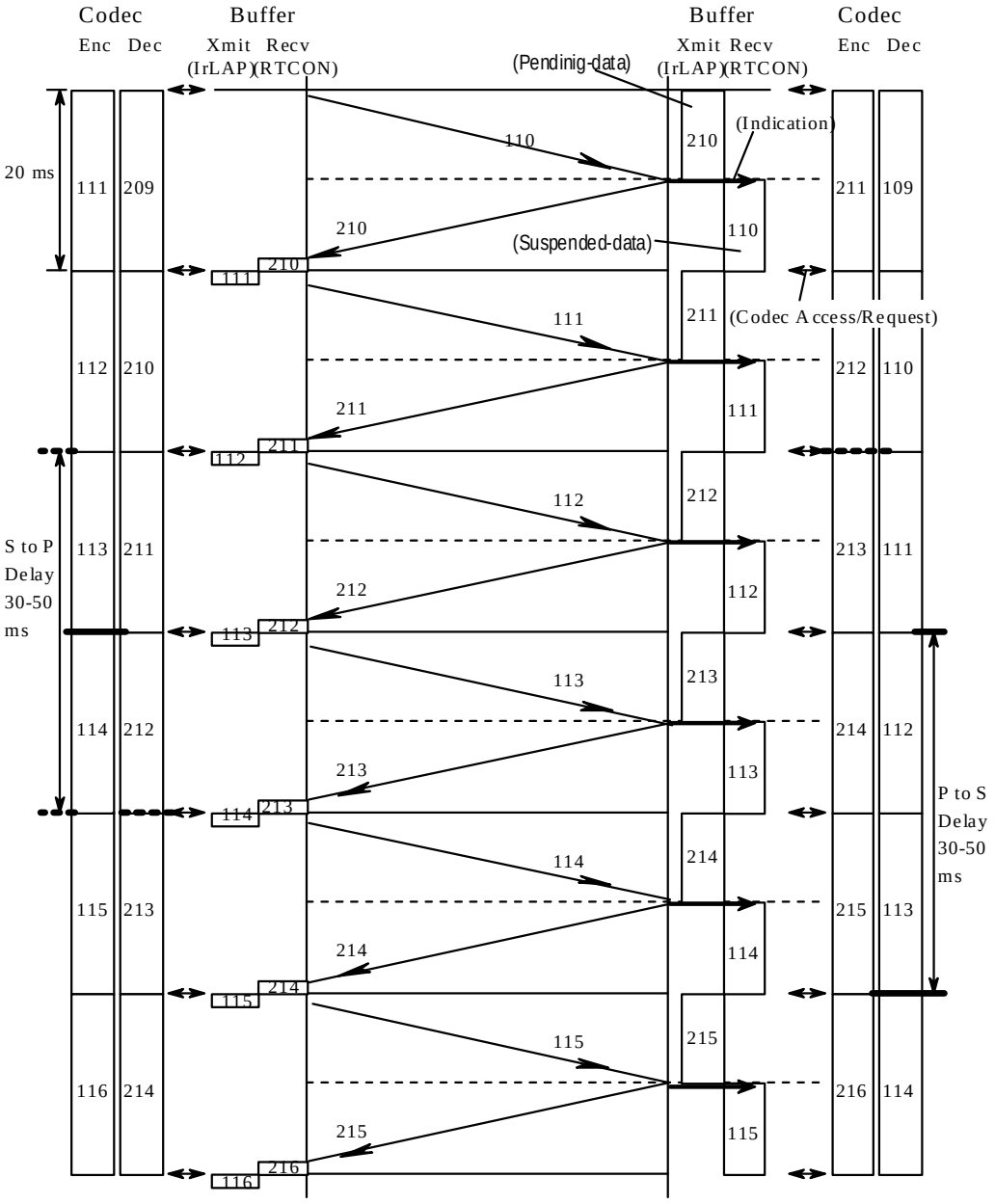


Figure 12-11 Normal Sequence

12.7.3.2 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 12.6.1.

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.

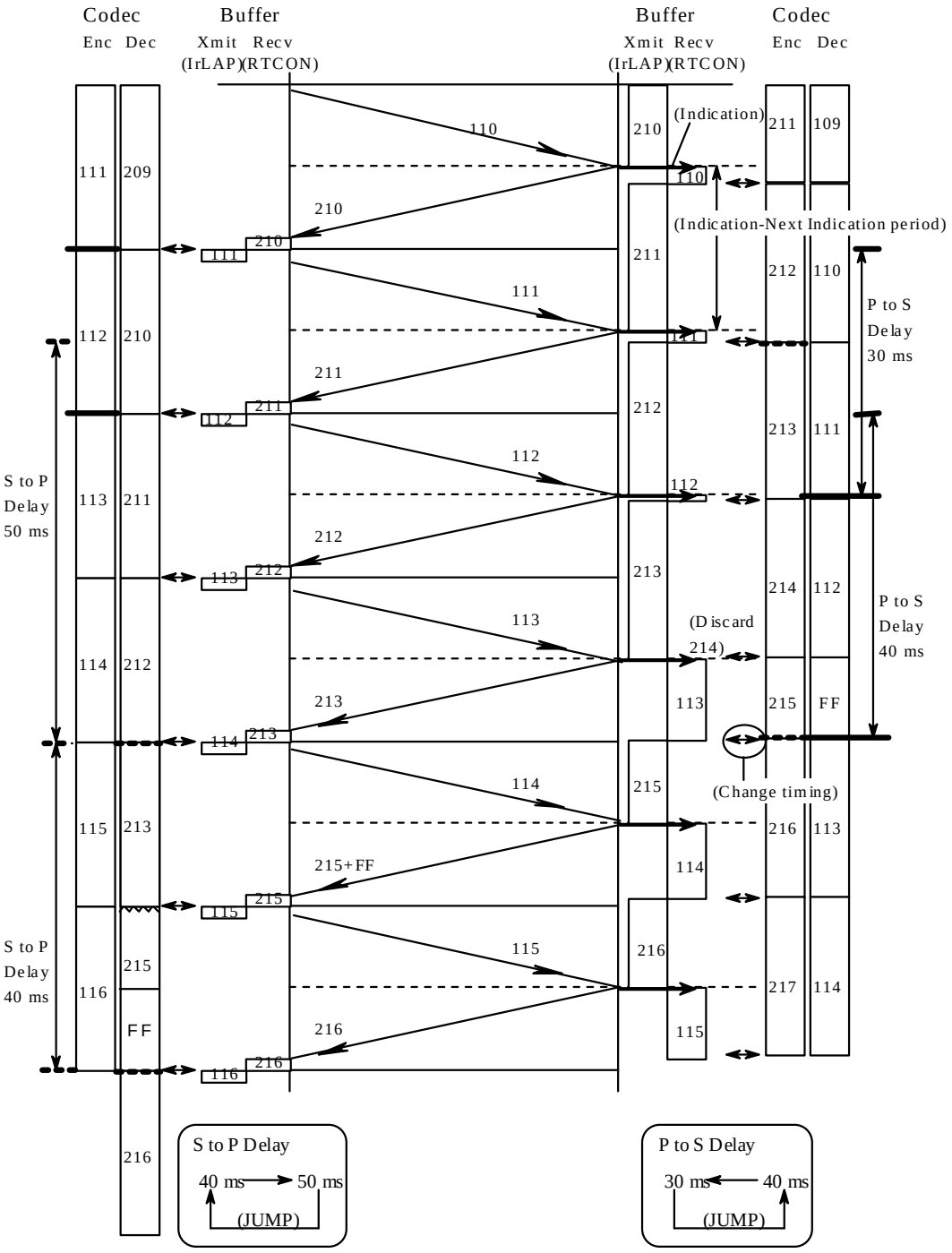


Figure 12-12 Secondary Clock Fast Sequence

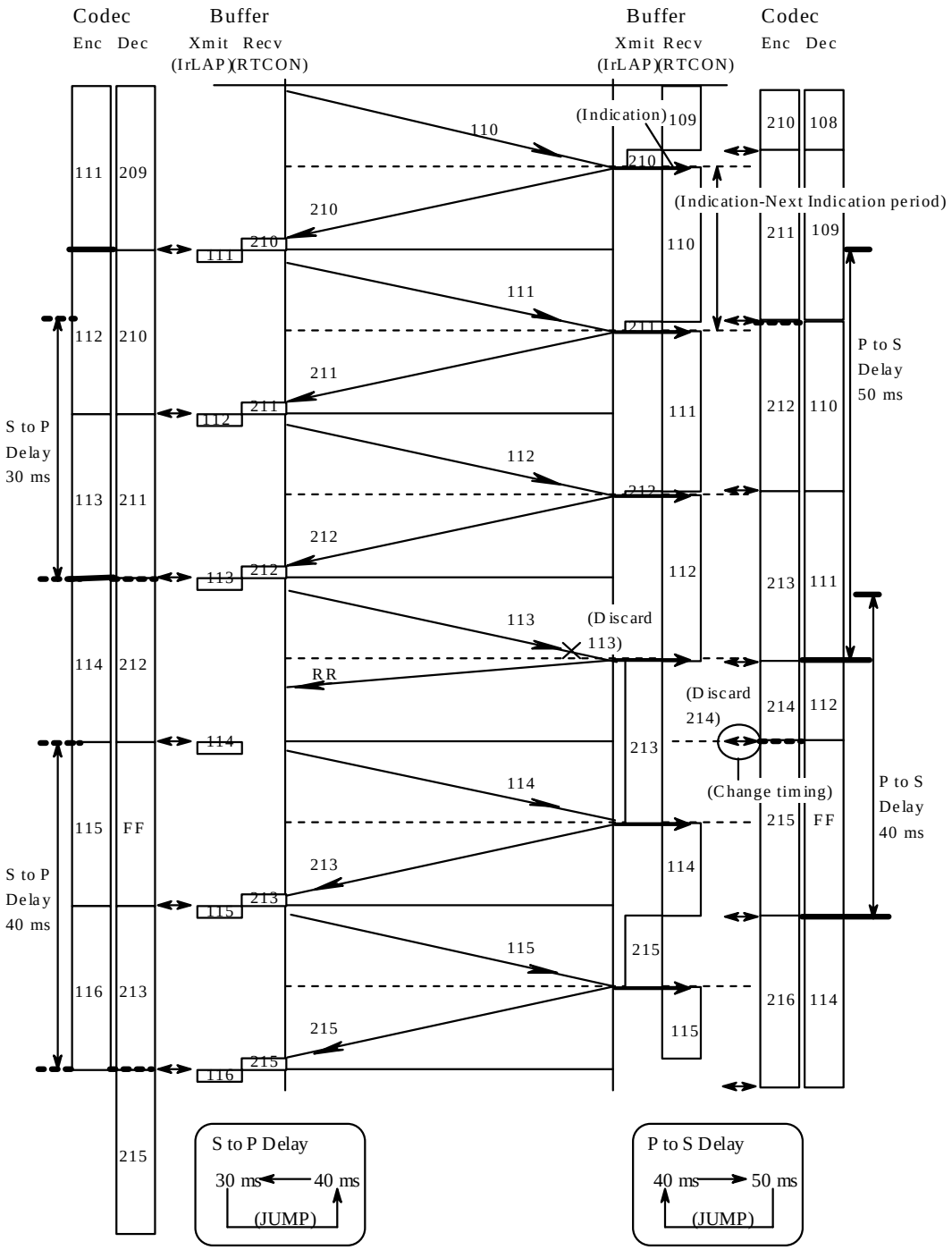


Figure 12-13 Secondary Clock Slow Sequence

12.7.3.3 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	Store 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 /* 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 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 /* fill with 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	Store 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 /* fill with 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	Store 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)

12.7.4 Delay Reduced Implementation for Secondary

12.7.4.1 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 -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).

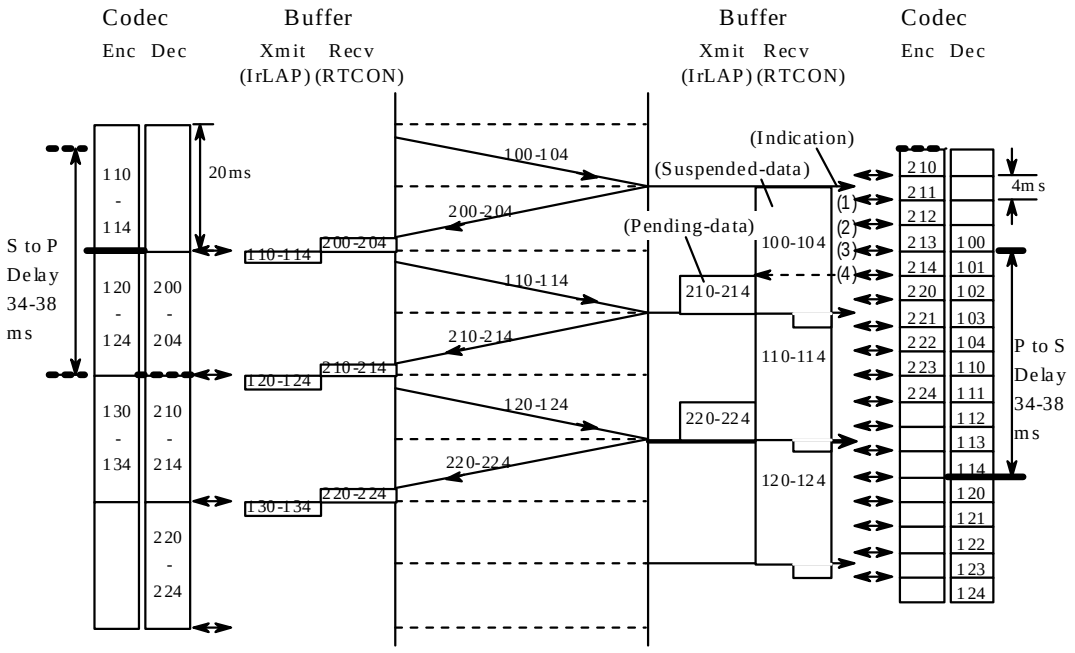


Figure 12-14 Normal Sequence

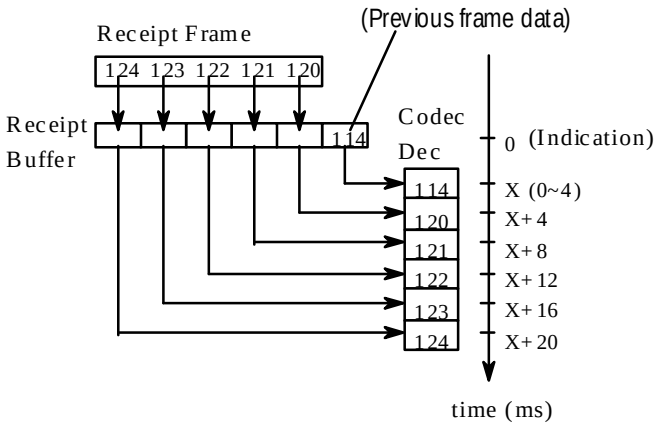


Figure 12-15 Suspended Period of One Frame Data in Receipt Buffer

12.7.4.2 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 12.6.1.

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.

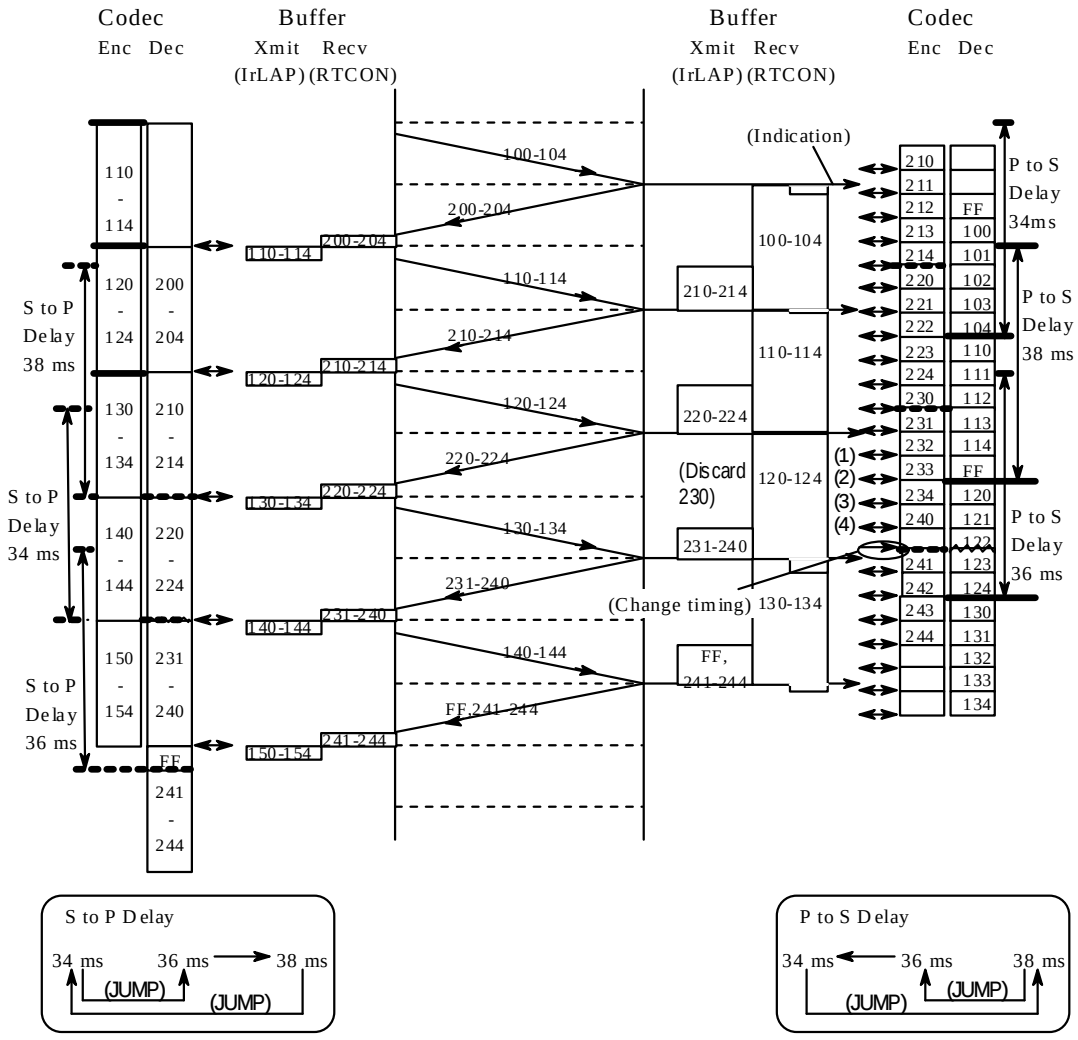


Figure 12-16 Secondary Clock Fast Sequence

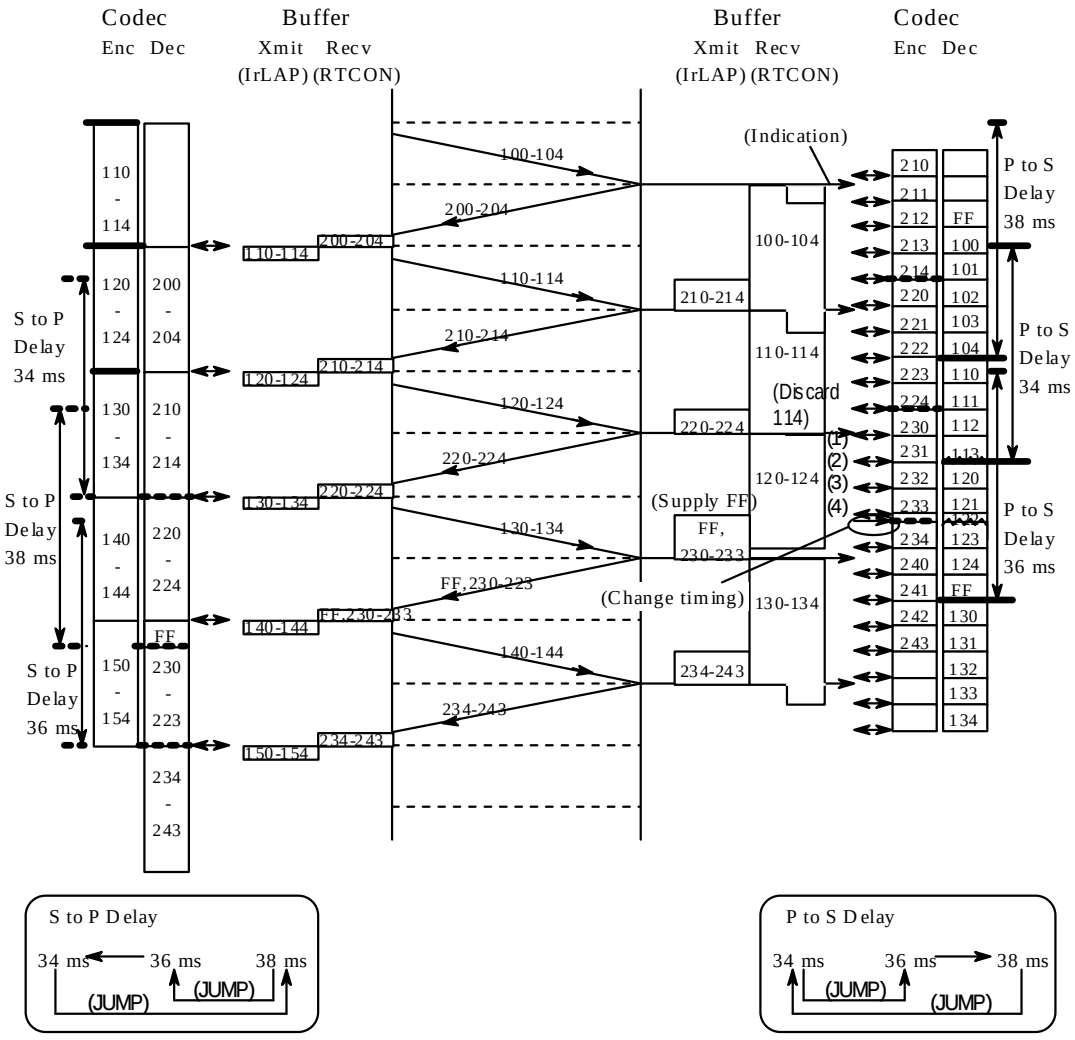


Figure 12-17 Secondary Clock Slow Sequence

12.7.4.3 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 RTCON_Control.Indication (SecodaryControlRxBuf) SecondaryControlRxBuf = Empty	STAND BY
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	Increase SecondaryAudioTxFIFOCount by AudioUnitDataSize Push AudioUnitData into SecondaryAudioTxFIFOData AudioUnitData=	ADPCM _RECV

	< SecondaryAudioTxFIFOSize ^ ShortResetFlag=0 ^ DropResetFlag=0	Pop from SecondaryAudioRxFIFOData by AudioUnitDataSize RTCON_Audio.Indication(AudioUnitData)	
	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 + AudioUnitDataSize < SecondaryAudioTxFIFOSize ^ DropResetFlag=1	Increase SecondaryAudioTxFIFOCount by AudioUnitDataSize Push AudioUnitData into SecondaryAudioTxFIFOData If (SecondaryAudioTxFIFODataCount exceeds over a half size of SecondaryAudioTxFIFOSize){ SecondaryAudioTxFIFOData = Empty DropResetFlag=0}	ADPCM _RECV
	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 delay of Data Indication from Primary */*	SecondaryAudioRxBuf = AudioPart of RTCONData SecondaryControlRxBuf = ControlPart of RTCONData RTCON_Control.Indication (SecondaryControlRxBuf) SecondaryControlRxBuf = null n=1 ShortFlag=0 DropFlag=1	ADPCM _XMIT
	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

13. IrMC applications IAS entry and service hint bit

13.1 IAS Entries

The information needed for in order to access the IrDA IrMC applications is included in the IAS class TELECOM. The Phonebook, Calendar, Message and Note applications are accessed through the OBEX Class. The following table defines the attributes associated with this class. For further information about IAS entries and access, please refer to the IrDA LMP Specification.

Note that IAS should be the primary means of identifying the services supported by a device.

Class IrDA:TELECOM's parameters must be listed in ascending PI order, which means that PhoneBookVersion is to come first, then PhoneBookSupport, etc. No assumptions should be made about the order of the 3-tuples in the Parameters or Parameters2 attributes.

Class IrDA:TELECOM

Attributes	
IrDA:TinyTP:LsapSel	Integer (0x01)
Parameters	Octet Sequence (0x02)
Parameters2	Octet Sequence (0x03)

13.1.1 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.

13.1.2 Parameters

The Parameters attribute uniquely identifies the IrMC services provided by a device. All IrMC service information, except for information about the Notes application, is packed into this one attribute to allow a single IAS GetValueByClass query. No assumptions should be made about the order of the 3-tuples in the Parameters attribute.

The Parameters attribute is an octet sequence, which consists of one or more 3-tuples with the following format:

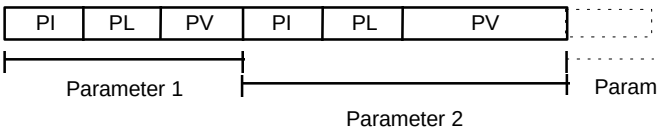


Figure 13-1 The structure of the Parameters attribute

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 13.1.2.1 - 13.1.2.5. 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.

13.1.2.1 Phone Book

PI	PI name	PL	PV data type	PV Description	Default Value
0x00	PhoneBookSupport	1	Byte (bit mask) 0x01 0x02 0x04 0x08	Levels Supported Minimum & Access Minimum, Access & Index Minimum and Sync Minimum only	0
0x01	PhoneBookOptional	1	byte (bit mask) bit 0 bit 1 bit 2 bit 4-3 bit 5-7	 <phone-book-incoming-call-history-object> supported <phone-book-outgoing-call-history-object> supported <phone-book-missed-call-history-object> supported Support for Index Type: BIT4-BIT3 00 = static 01 = dynamic* 10 = unique 11 = reserved Reserved	0
0x02	PhoneBookVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,1)

***** *Note: Dynamic Indexing was present in IrMC 1.0 but was removed in IrMC 1.1. This value is provided for backwards compatability only.*

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.

The parameter value for Minimum Support should be set if the device supports Connection Oriented Minimum Support. There is no need to set the parameter value for the minimum phone book service support if it is only supported as a connectionless service since no IAS queries are to be expected during a connectionless data exchange.

PHONEBOOKOPTIONAL
The PhoneBookOptional parameter is used to indicate the support of the incoming, outgoing and missed call history objects. The parameter also indicates the type of index method supported, if any. This parameter is optional and the default value is 0 indicating no support of these objects and static indexing.

PHONEBOOKVERSION
The PhoneBookVersion parameter is used to indicate the version number of the IrMC 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.

13.1.2.2 Calendar

PI	PI name	PL	PV data type	PV Description	Default Value
0x10	CalendarSupport	1	byte (bit mask) 0x01 0x02 0x04 0x08	Levels Support Minimum & Access Minimum, Access and Index Minimum and Sync Minimum only	0
0x11	CalendarVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,1)
0x12	CalendarOptional	1	byte (bit mask) bit 1- 0 bit 2-7	 Support for Index Type: BIT1-BIT0 00 = static 01 = dynamic* 10 = unique 11 = reserved Reserved	

***** *Note: Dynamic Indexing was present in IrMC 1.0 but was removed in IrMC 1.1. This value is provided for backwards compatability only.*

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.

The parameter value for Minimum Support should be set if the device supports Connection Oriented Minimum Support. There is no need to set the parameter value for the minimum calendar service support if it is only supported as a connectionless service since no IAS queries are to be expected during a connectionless data exchange.

CALENDARVERSION
The CalendarVersion parameter is used to indicate the version number of the IrMC 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.

CALENDAROPTIONAL
The CalendarOptional parameter is used to indicate the type of index method supported, if any. This parameter is optional and the default value is 0 indicating static indexing.

13.1.2.3 Messaging

PI	PI name	PL	PV data type	PV Description	Default Value
0x20	MessageSupport	1	Byte (bit Mask) 0x01 0x02 0x04 0x08	Levels Supported Minimum & Access Minimum, Access and Index Minimum and Sync Minimum only	0
0x21	MessageOptional	1	byte (bit mask) bit 0 bit 1-2 bit3 bits 4-7	 <message-missed-history-object> supported 1 = supported 0 = not supported Index Level Supported BIT1-BIT2 00 = static 10 = dynamic* 01 = unique 11 = reserved Sent Box supported 0 = no** 1 = yes Reserved	0
0x22	MessageVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,1)

***** *Note: Dynamic Indexing was present in IrMC 1.0 but was removed in IrMC 1.1. This value is provided for backwards compatability only.*

****** *Note: In IrMC 1.0, this bit was not defined. Therefore, it is possible that some implementations may exist that contain an Sent Box Object Store, but have not turned on bit 3. Therefore, a value of 0 should probably be treated as a "maybe" in 1.0 implementations. Attempting to retrieve the Sent Object Store is the most accurate means of determining the existence of the object. IN 1.1 implementations, this bit3 means "NO".*

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.

The parameter value for Minimum Support should be set if the device supports Connection Oriented Minimum Support. There is no need to set the parameter value for the minimum Message service support if it is only supported as a connectionless service since no IAS queries are to be expected during a connectionless data exchange.

MESSAGEOPTIONAL
The MessageOptional parameter is used to indicate the support of the missed message history object. The parameter also indicates the type of index method supported, if any. Finally, the parameter indicates support of the Sent Box. This parameter is optional and the default value is 0 indicating no support of the Missed Message History Object, no support of the Sent Box and static indexing.

MESSAGEVERSION
The MessageVersion parameter is used to indicate the version number of the IrMC 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.

13.1.2.4 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.	(1,1)
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 IrMC 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 IrMC 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.

13.1.2.5 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.	(1,1)
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 IrMC 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.

13.1.3 Parameters2

The Parameters2 attribute uniquely identifies information about the IrMC Notes application services provided by a device. Because of packet size limitations, this information can not be stored in the Parameters attribute. No assumptions should be made about the order of the 3-tuples in the Parameters2 attribute.

The Parameters2 attribute is an octet sequence, which consists of one or more 3-tuples with a format as described in Figure 13-1. 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 Parameters2 3-tuples for the Notes service of the IrDA:TELECOM class is listed in the following section. If the Notes service is not supported Parameters2 may be omitted.

Parameter identifier values in the range of 0x20-0x7F and 0x83 -0xFF are reserved for future use.

13.1.3.1 Notes

PI	PI name	PL	PV data type	PV Description	Default Value
0x00	NotesSupport	1	Byte (bit mask) 0x01 0x02 0x04 0x08	Levels Supported Minimum & Access Minimum, Access and Index Minimum & Sync Minimum Support	0
0x01	NotesOptional	1	byte (bit mask) bit 1-0 bit 2-7	 Support for Index Type: BIT1-BIT0 00 = static 01 = reserved 10 = unique 11 = reserved Reserved	0
0x02	NotesVersion	2	two octets	The first octet = major version number The second octet = the minor version number.	(1,1)

NOTESUPPORT
The NoteSupport parameter is used to indicate the level of the service supported. Default value is 0 for no Note Object Store available. If a connection oriented Note service is implemented, this parameter is mandatory.

The parameter value for Minimum Support should be set if the device supports Connection Oriented Minimum Support. There is no need to set the parameter value for the minimum Note service support if it is only supported as a connectionless service since no IAS queries are to be expected during a connectionless data exchange.

NOTEOPTIONAL
The NoteOptional parameter is used to indicate the type of index method supported, if any. This parameter is optional and the default value is 0 indicating static indexing.

NOTEVERSION
The NoteVersion parameter is used to indicate the version number of the IrMC 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.

13.1.3.2 Device Serial Number

PI	PI name	PL	PV data type	PV Description	Default Value
0x10	Serial Number	1+	Bytes	The Serial Number of the Device	*

SERIAL NUMBER
The Serial Number parameter is used to indicate the device serial number. This should be identical to the value stored in the Device Information Object. If there is no Serial Number, or this field can't be supported, then the

13.2 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 IrMC application capabilities of the device. For more information, please refer to the IrDA LMP Specification.

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 IrMC services, for example calendar, call control or audio. Consequently, all PCs, pagers and other non-phone devices are also expected to indicate their IrMC capability with the Telephony bit.

IrOBEX is used to access the Phonebook, Calendar, Message and Note objects. Devices which support one or more of these objects, should set the OBEX IrLMP service hint bit (value of 0x20 in the second hint byte.)