Technical Document

GSM PROTOCOL STACK

G23

UDP

MESSAGE SEQUENCE CHARTS

Document Number:	8444.200.00.001
Version:	0.4
Status:	Draft
Approval Authority:	Udo Sprute
Creation Date:	2000-May-18
Last changed:	2015-Mar-08 by Jacek Kwasnik
File Name:	udp.doc

Copyright © 2003 Texas Instruments
Texas Instruments Proprietary Information – Internal Data
technical_document_20030404.dot

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

No part of this document may be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying and recording, for any purpose without the express written permission of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
2000-May-18	NI et al.		0.1		1
2000-May-31	SG et al.		0.2		2
2003-Jun-12	XINTEGRA		0.3	Draft	
2003-Jun-12	JK		0.4	Draft	3

Notes:

1. Initial version
2. MSC added
3. Nomenclature of DTI corrected

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000]	International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1 Introduction

The User Datagram Protocol (UDP) is a simple datagram-oriented transport protocol based on IP. It basically makes the services of IP available to applications. Consequently, the service offered by UDP is an unreliable, best-effort delivery of datagrams. Its main features over IP are checksumming of the data payload and multiplexing datagrams by port number. There are no mechanisms for end-to-end reliablility, flow control, or sequencing in UDP. Therefore it is suited for applications that do not need these mechanisms or provide their own and can profit from the low overhead of UDP in terms of both processing and bandwidth.

The most common application-layer protocols using UDP are the Domain Name System (DNS), Trivial File Transfer Protocol (TFTP), and Network File System (NFS).

The Wireless Datagram Protocol, upon which WAP is built, is a good candidate for using UDP for three reasons:

1. Because it deals with very small amounts of information (a WML deck has a maximum size of 1400 bytes), there is no need for flow control.
2. Because it also uses other bearer capabilities without end-to-end reliability, WDP has to provide for reliability on its own anyway.
3. Due to the very limited bandwidth of GSM data channels, bandwidth efficiency is of great importance.

1.1 Standards and Supplementary Documents

Internet standards such as UDP are described in the “Request for Comments” (RFC) series of documents. The RFCs are available online from http://www.rfc-editor.org/ (inside Condat from http://chuck/rfcs/). The following RFCs are relevant for UDP:

[RFC 768]
Jon Postel: *User Datagram Protocol*. RFC 768, August 1980.
http://chuck/rfcs/rfc0768.txt or ftp://ftp.isi.edu/in-notes/rfc768.txt
The UDP specification.

[RFC 1071]
R.T. Braden, D.A. Borman, C. Partridge: *Computing the Internet checksum*. RFC 1071, September 1988.
http://chuck/rfcs/rfc1071.txt or ftp://ftp.isi.edu/in-notes/rfc1071.txt
This RFC gives several examples for implementation of the Internet checksum algorithm. RFC 1141 adds a method for incremental update of the checksum, e. g. on TTL decrement.

[RFC 1122]
Robert. T. Braden (ed.): *Requirements for Internet Hosts – Communication Layers*. RFC 1122, October 1989.
http://chuck/rfcs/rfc1122.txt or ftp://ftp.isi.edu/in-notes/rfc1122.txt
Among other topics, RFC 1122 contains clarifications and additions to the initial UDP specification.

[RFC 1700]
Joyce K. Reynolds, Jon Postel: *Assigned Numbers*. RFC 1700, October 1994.
http://chuck/rfcs/rfc1700.txt or ftp://ftp.isi.edu/in-notes/rfc1700.txt
RFC 1700 contains numeric values and other definitions in use with the TCP/IP protocol suite.

In addition to the RFCs, the following books are valuable sources of information:

[MacKusick et al. 1996]
Marshal Kirk McKusick, Keith Bostic, Michael J. Karels, John S. Quarterman: *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley 1996.
Chapter 13 contains a very concise overview of UDP.

[Stevens 1994a]
W. Richard Stevens: *TCP/IP Illustrated Volume 1 – The Protocols*. Addison-Wesley, 1994.
This book contains an excellent introduction to the TCP/IP protocol suite. Relevant to UDP is chapter 11.

[Stevens 1994b]
W. Richard Stevens: *TCP/IP Illustrated Volume 2 – The Implementation*. Addison-Wesley, 1994.
In this book. Stevens annotates the complete 4.4 BSD TCP/IP source code. Relevant to UDP is chapter 23.

These RFCs and books were the main sources used in preparation of this chapter.

The 4.4 BSD TCP/IP software documented in [Stevens 1994a] is an excellent example of a full-featured and very mature implementation of the Internet protocol suite. While it is for several reasons not suited for direct integration into a mobile phone, it is a good reference for any TCP/IP implementer. The TCP/IP code of FreeBSD, a direct successor of 4.4 BSD, can been found at ftp://ftp.freebsd.org/pub/FreeBSD/FreeBSD-stable/src/sys/netinet/ (current version of the "stable" development branch) or, inside Condat, at http://chuck/freebsd/src/sys/netinet/ (from FreeBSD release 3.4 of December 1999).

1.2 UDP Basics

The data portions sent by the UDP layer are usually called UDP datagrams, UDP messages, or UDP packets. To distinguish them from IP datagrams and IP packets (which can be datagrams or fragments), we will call them UDP *messages* in this document.

1.2.1 Features

UDP is one of the transport layer protocol of the TCP/IP protocol suite. UDP messages are encapsulated in IP datagrams.

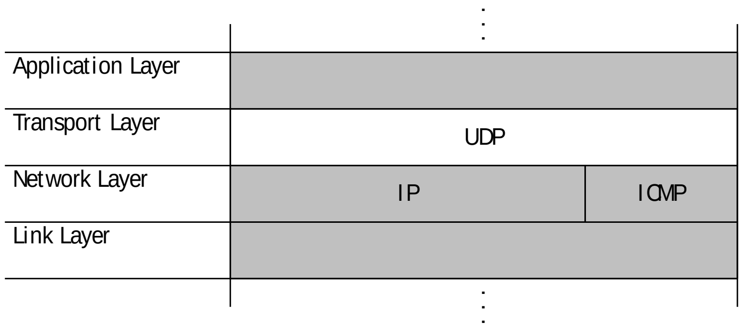

Figure 1-1: ISO/OSI model layer of UDP

UDP is based on IP and provides no mechanisms of end-to-end reliability, flow-control, or sequencing. The service offered by UDP is therefore, like that of IP, an unreliable, connectionless datagram delivery service. Each data portion sent by the application through UDP is sent as a single UDP message, which in turn is sent as a single IP datagram. (The IP datagram may be fragmented in the IP layer of the sender or an intermediate router.) If the size of the data portion including UDP and IP headers is too large for an IP datagram, the message will not be sent and an error will be signaled. The maximum size of an IP datagram depends on the local IP implementation.

The features of UDP are:

Checksumming of data

The UDP header includes a 16-bit checksum of the UDP header, parts of the IP header, and the data payload. Although setting the UDP checksum is optional for the sender, in the past there have been bad experiences with omitting the checksum, so the checksum should *always* be used. For the receiver it is mandatory to recalculate the checksum, if it is set in a packet, and to silently discard UDP messages with checksum errors.

Multiplexing by port numbery

The UDP header contains a *port number* for both source and destination. The port number is used to relate the UDP message to the receiving application. In a simple client-server application, the client sends a UDP message to a *well-known* destination port to address a certain service, using a source port number that identifies the client application on the local host. The server then sends the reply back to the client using the well-known port number as its own source port number and the source port number of the client's request message as the destination port number of the reply message. On both sides, it is the responsibility of the UDP layer to dispatch the message to the correct application or process based on the destination port number in the UDP header.

1.2.2 UDP Packet header

All UDP header fields are in *network order*, i. e. big endian (most significant bit/byte first). C library implementations usually supply the functions or macros ntohs() and htons() to convert 16-bit values between network order and host order.

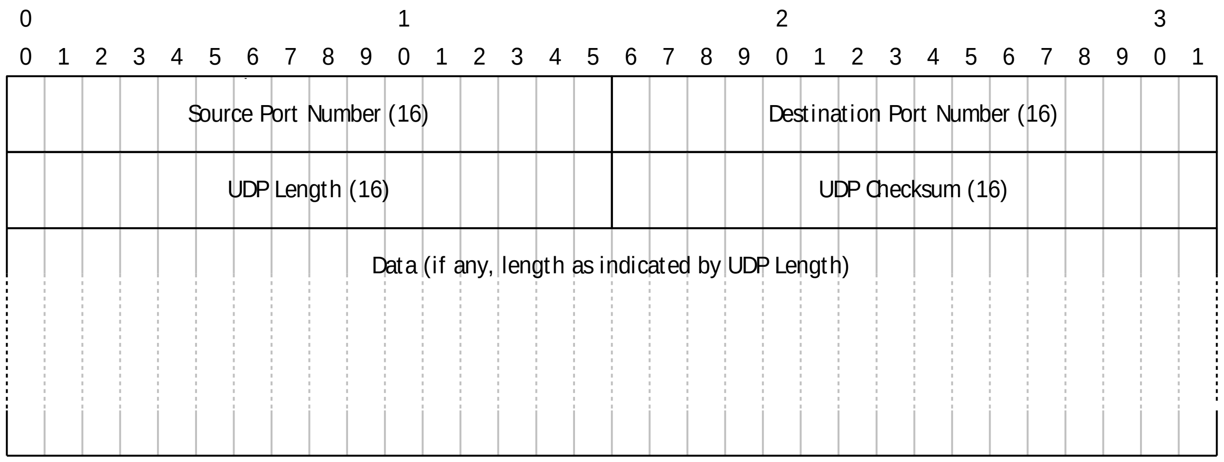

Figure 1-2: UDP message Header

The lengths in the diagram above are given in bits.

Source Port Number,
Destination Port Number
Numbers of the UDP source and destination ports.

UDP Length
Length of the UDP message, including header and data. The minimum length is therefore 8.

UDP Checksum
Checksum over the UDP header, an IP pseudo header, and the Data. The IP pseudo header is conceptually prepended to the UDP header:

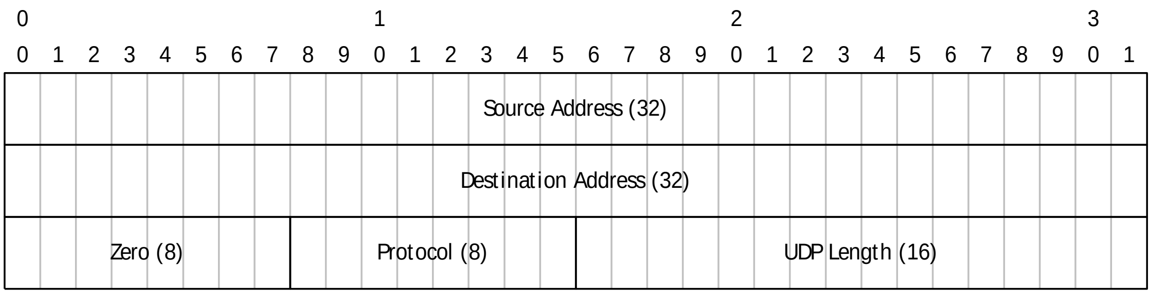

Figure 1-3: IP pseudo header for UDP checksum

The Zero field has the value zero; all other fields have the same value as in the IP header or, for the UDP length, as in the UDP header, respectively. Note that UDP must know the source IP address in advance to calculate the UDP checksum!
The algorithm to calculate the UDP checksum is the same as for the IP checksum. A C implementation of the checksum algorithm can be found in [RFC 1071]. Unfortunately this implementation is incorrect. A corrected version looks like this:

```
{
         /* Compute Internet Checksum for "count" bytes
          *         beginning at location "addr", which is a
          *         pointer to unsigned short (16 bits).
          */
     register long sum = 0;

     while( count > 1 )  {
         /*  This is the inner loop */
             sum += *addr++;
             count -= 2;
     }

         /*  Add left-over byte, if any */
     if( count > 0 )
             sum += * (unsigned char *) addr;

         /*  Fold 32-bit sum to 16 bits */
     while (sum>>16)
         sum = (sum & 0xffff) + (sum >> 16);

     checksum = ~sum;
}
```

The Data may actually have zero length.

1.3 UDP Algorithm

This section is not intended as a comprehensive description of the UDP algorithms, but rather as a general outline.

1.3.1 UDP attach

The application attaches to the UDP layer. The application specify a port number on which it wants to receive UDP messages. If it does not specify a port number, an ephemeral port number will be assigned by the UDP layer. It may also specify an IP address on which it want so receive UDP messages. If no IP address is specified, UDP messages arriving for all IP addresses of the host and the chosen port number will be delivered to the application. (The Berkeley socket interface models this with the socket( ) and bind( ) functions.)

1.3.2 UDP output

The UDP layer is called by the application to send a UDP message to a specified IP address and a specified port.

1. UDP asks the IP layer for the source IP address the packet will be sent with. It is an error if the application has specified a different source IP address.
2. UDP constructs the UDP header and the IP header and calculates the UDP checksum over the fields of the pseudo IP header, the UPD header, and the message payload.
3. The packet is sent to IP.

1.3.3 UDP input

The UDP layer is called by the IP layer to deliver a UDP message.

1. The UDP layer checks the UDP checksum and silently discards the packet if the checksum is in error.
2. The UDP layer sends the UDP message along with the IP header to the application associated with the destination port number of the message. If there is no application associated with this port number, an ICMP message with type Destination Unreachable and code Port Unreachable is sent to sender of the message.

ICMP error messages of the type Destination Unreachable or Time Exceeded is passed to the application as an appropriate error indication.

1.4 Requirement Levels

[RFC 1122] lists a couple of features that a UDP implementation *must*, *should*, *may*, *should not*, and *must not* implement. The meaning of these terms is defined in RFC 2119 ("Key words for use in RFCs to Indicate Requirement Levels", Scott Bradner, March 1997):

1. MUST This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
2. MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
4. SHOULD NOT This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or

even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

The standard RFC 1122 is targeted at general-purpose Internet host. The situation with the mobile phone is vastly different, so the standard is, in this context, not applicable to all issues mentioned in RFC 1122. The restrictions are:

- Sending IP options is not supported; incoming options are ignored.
- Setting of TOS, DF, TTL by the application is not supported.

The following table contains the list of requirement levels for UDP from [RFC 1122].

Req. Level	Feature	We do	Comment
SHOULD	UDP send Port Unreachable	•	
	IP Options in UDP		
MUST	- Pass rcv'd IP options to applic layer	–	We do not really support IP options
MUST	- Applic layer can specify IP options in Send	–	
MUST	- UDP passes IP options down to IP layer	–	
MUST	Pass ICMP msgs up to applic layer	•	As error notifications
	UDP checksums:		
MUST	- Able to generate/check checksum	•	Will always generate checksum
MUST	- Silently discard bad checksum	•	
MAY	- Sender Option to not generate checksum	–	
MUST	- Default is to checksum	•	
MAY	- Receiver Option to require checksum	–	
	UDP Multihoming		
MUST	- Pass spec-dest addr to application	•	
MUST	- Applic layer can specify Local IP addr	•	
MUST	- Applic layer specify wild Local IP addr	•	i. e. 0.0.0.0
SHOULD	- Applic layer notified of Local IP addr used	–	
MUST	Bad IP src addr silently discarded by UDP/IP	•	
MUST	Only send valid IP source address	•	
	UDP Application Interface Services		
MUST	Full IP interface of 3.4 for application	–	
MUST	- Able to spec TTL, TOS, IP opts when send dg	–	may be supported in a later version
MAY	- Pass received TOS up to applic layer	–	

Table 1-1: Summary of the Requirements for Internet Hosts (UDP)

2 Protocol

2.1 Deactivated State

2.1.1 Activation of UDP

```
ACI                          UDP
  |                            |
  |      UDPA_ACTIVATE_REQ     |
  *==========================>*
  |                        (UDP 1)
  |                            |
  |                            |
  |      UDPA_ACTIVATE_CNF     |
  *<===========================*
(ACI 1)                        |
  |                            |
```

(UDP 1)
The UDP entity receives the UDPA_ACTIVATE_REQ and is switched to activated state.

(ACI 1)
UDP informs ACI that the entity is activated.

2.1.2 Reception of DTI2_DATA_REQ in Deactivated State

```
 WAP                        UDP
  |                           |
  |      DTI2_DATA_REQ        |
  *=========================>*
  |                        (UDP 1)
  |                           |
```

(UDP 1)
The UDP entity receives the DTI2_DATA_REQ from an application (e.g. WAP) and ignores it.

2.1.3 Reception of DTI2_READY_IND in Deactivated State

```
 UDP                        IP
  |                           |
  |      DTI2_READY_IND       |
  *<=========================*
(UDP 1)                       |
  |                           |
```

(UDP 1)
The IP entity receives the DTI2_READY_IND from an interface layer and ignores it.

2.1.4 Reception of DTI2_DATA_IND in Deactivated State

```
 UDP                        IP
  |                          |
  |      DTI2_DATA_IND       |
  *<=========================*
(UDP 1)                      |
  |                          |
```

(UDP 1)
The UDP entity receives the DTI2_DATA_IND from an interface layer and ignores it.

2.1.5 Reception of UDP_BIND_REQ in Deactivated State

```
WAP                        UDP
  |                          |
  |      UDP_BIND_REQ        |
  *=========================>*
  |                        (UDP 1)
  |                          |
  |      UDP_BIND_CNF        |
  |   (0, UDP_BIND_UDPDOWN)  |
  *<=========================*
(WAP 1)                      |
  |                          |
```

(UDP 1)
The UDP entity receives the UDP_BIND_REQ from an application (e.g. WAP).

(WAP 1)
UDP sends a UDP_BIND_CNF with the err field of the primitive set to UDP_BIND_UDPDOWN.

2.1.6 Reception of UDP_CLOSEPORT_REQ in Deactivated State

```
WAP                        UDP
  |                          |
  |     UDP_CLOSEPORT_REQ    |
  *=========================>*
  |                        (UDP 1)
  |                          |
  |     UDP_CLOSEPORT_CNF    |
  *<=========================*
(WAP 1)                      |
  |                          |
```

(UDP 1)
The UDP entity receives the UDP_CLOSEPORT_REQ from an application (e.g. WAP).

(WAP 1)
UDP confirms the request.

2.2 Activated and not-configured State

2.2.1 Configuration of UDP

```
ACI                          UDP                         IP
  |                            |                          |
  |    UDPA_CONFIG_REQ         |                          |
  *==========================>*                          |
  |                         (UDP 1)                       |
  |                            |                          |
  |                            |      DTI2_GETDATA_REQ    |
  |                            *=========================>*
  |                            |                        (IP 1)
  |                            |                          |
  |                            |                          |
  |    UDPA_CONFIG_CNF         |                          |
  *<==========================*                          |
(ACI 1)                        |                          |
  |                            |                          |
```

(UDP 1)
ACI configures UDP.

(IP 1)
UDP indicates that the entity is ready to get the next DTI2_DATA_IND primitive from the network layer.

(ACI 2)

UDP confirms the configuration and is now in configured and activated state.

2.2.2 Reception of DTI2_READY_IND in not-configured State

```
 UDP                         IP
  |                           |
  |      DTI2_READY_IND       |
  *<=========================*
(UDP 1)                       |
  |                           |
```

(UDP 1)
The DUP entity receives the DTI2_READY_IND from the network layer and stores it.

2.2.3 Reception of DTI2_DATA_IND in not-configured State

```
 UDP                         IP
  |                           |
  |      DTI2_DATA_IND        |
  *<=========================*
(UDP 1)                       |
  |                           |
```

(UDP 1)
The UDP entity receives the DTI2_DATA_IND from the network layer and ignores it.

2.2.4 Reception of DTI2_GETDATA_REQ in not-configured State

```
 WAP                         UDP
  |                           |
  |      DTI2_GETDATA_REQ     |
  *==========================>*
  |                         (UDP 1)
  |                           |
```

(UDP 1)
The UDP entity receives the DTI2_GETDATA_REQ from an application (e.g. WAP) and ignores it.

2.2.5 Reception of DTI2_DATA_REQ in not-configured State

```
 WAP                         UDP
  |                           |
  |       DTI2_GETDATA_REQ    |
  *==========================>*
  |                         (UDP 1)
  |                           |
```

(UDP 1)
The UDP entity receives the DTI2_DATA_REQ from an application (e.g. WAP) and ignores it.

2.2.6 Reception of UDP_BIND_REQ in not-configured State

```
WAP                         UDP
  |                           |
  |       UDP_BIND_REQ        |
  *==========================>*
  |                         (UDP 1)
  |                           |
  |       UDP_BIND_CNF        |
  |    (0, UDP_BIND_UDPDOWN)  |
  *<==========================*
(WAP 1)                       |
  |                           |
```

(UDP 1)
The UDP entity receives the UDP_BIND_REQ from an application (e.g. WAP).

(WAP 1)
UDP sends an UDP_BIND_CNF with the err field of the primitive set to UDP_BIND_UDPDOWN.

2.2.7 Reception of UDP_CLOSEPORT_REQ in not-configured State

```
WAP                          UDP
  |                            |
  |      UDP_CLOSEPORT_REQ     |
  *===========================>*
  |                          (UDP 1)
  |                            |
  |      UDP_CLOSEPORT_CNF     |
  *<===========================*
(WAP 1)                        |
  |                            |
```

(UDP 1)
The UDP entity receives the UDP_CLOSEPORT_REQ from an application (e.g. WAP).

(WAP 1)
UDP confirms the request.

2.2.8 Reception of UDPA_DEACTIVATE_REQ in not-configured State

```
 ACI                          UDP
  |                            |
  |    UDPA_DEACTIVET_REQ      |
  *===========================>*
  |                          (UDP 1)
  |                            |
  |    UDPA_DEACTIVAE_CNF      |
  *<===========================*
(ACI 1)                        |
  |                            |
```

(UDP 1)
The UDP entity receives the UDPA_DEACTIVATE_REQ and is switched to the deactivated state.

(WAP 1)
UDP informs ACI that the entity is deactivated.

2.3 Configured State

2.3.1 Deactivation of UDP

```
ACI                          WAP                        UDP
  |                           |                          |
  |                UDPA_DECTIVATE_REQ                    |
  *=====================================================>*
  |                           |                        (UDP 1)
  |                           |                          |
  |                           |      UDP_SHUTDOWN_IND    |
  |                           *<=========================*
  |                        (WAP 1)                       |
  |                           |                          |
  |                           |      UDP_SHUTDOWN_RES    |
  |                           *=========================>*
  |                           |                       (UDP 2)
  |                           |                          |
  |                           |                          |
  |                UDPA_DEACTIVATE_CNF                   |
  *<=====================================================*
(ACI 1)                       |                          |
  |                           |                          |
```

(UDP 1)
The UDP entity receives the UDPA_DEACTIVATE_REQ.

(WAP 1)
UDP indicates its deactivation to each bound application.

(UDP 2)
WAP confirms the UDP_SHUTDOWN_IND.

(ACI 1)
UDP confirms its deactivation and is switched to the deactivated state.

2.3.2 Reception of UDP_BIND_REQ in configured State (any port)

```
WAP                           UDP
  |                            |
  |       UDP_BIND_REQ         |
  |           (0)              |
  *==========================>*
  |                          (UDP 1)
  |                            |
  |       UDP_BIND_CNF         |
  | (port, UDP_BIND_NOERROR)   |
  *<==========================*
(WAP 1)                        |
  |                            |
  |      DTI2_GETDATA_REQ      |
  *==========================>*
  |                          (UDP 2
  |                            |
  |      DTI2_READY_IND        |
  *<==========================*
(WAP 2)                        |
  |                            |
```

(UDP 1)
The UDP entity receives the UDP_BIND_REQ from an application (e.g. WAP).

(WAP 1)
UDP sends an UDP_BIND_CNF with the port field set to a free port number (e.g. > 1024).

(UDP 2)
The application (WAP) indicates that the entity is ready to get the next DTI2_DATA_IND primitive from the transport layer. (if there are no other open ports for this specific application)

(WAP 2)
UDP indicates that the entity is ready to get the next DTI2_DATA_REQ primitive from the application. (if there are no other open ports for this specific application)

2.3.3 Reception of UDP_BIND_REQ in configured State (special port, failure)

```
WAP                           UDP
  |                            |
  |       UDP_BIND_REQ         |
  |    (fixed port number P)   |
  *==========================>*
  |                          (UDP 1)
  |                            |
  |       UDP_BIND_CNF         |
  | (P, UDP_BIND_PORTINUSE)    |
  *<==========================*
(WAP 1)                        |
  |                            |
```

(UDP 1)
The UDP entity receives the UDP_BIND_REQ from an application (e.g. WAP). The requested port number is already used by (maybe another) application.

(WAP 1)
UDP sends an UDP_BIND_CNF with the err field of the primitive set to UDP_BIND_PORTINUSE.

2.3.4 Reception of UDP_BIND_REQ in configured State (special port, success)

```
WAP                           UDP
  |                             |
  |       UDP_BIND_REQ          |
  |    (fixed port number P)    |
  *===========================>*
  |                          (UDP 1)
  |                             |
  |       UDP_BIND_CNF          |
  |   (P, UDP_BIND_NOERROR)     |
  *<===========================*
(WAP 1)                         |
  |                             |
  |      DTI2_GETDATA_REQ       |
  *===========================>*
  |                          (UDP 2)
  |                             |
  |      DTI2_READY_IND         |
  *<===========================*
(WAP 2)                         |
  |                             |
```

(UDP 1)
The UDP entity receives the UDP_BIND_REQ from an application (e.g. WAP).

(WAP 1)
UDP sends an UDP_BIND_CNF with the port field set to the requested port.

(UDP 2)
The application (WAP) indicates that the entity is ready to get the next DTI2_DATA_IND primitive from the transport layer. (if there are no other open ports for this specific application)

(WAP 2)
UDP indicates that the entity is ready to get the next DTI2_DATA_REQ primitive from the application. (if there are no other open ports for this specific application)

2.3.5 Reception of UDP_CLOSEPORT_REQ in configured State

```
WAP                           UDP
  |                             |
  |      UDP_CLOSEPORT_REQ      |
  |      (port P)               |
  *===========================>*
  |                          (UDP 1)
  |                             |
  |      UDP_CLOSEPORT_CNF      |
  *<===========================*
(WAP 1)                         |
  |                             |
```

(UDP 1)
The UDP entity receives the UDP_CLOSEPORT_REQ from an application (e.g. WAP). UDP closes port *P*.

(WAP 1)
UDP confirms the request.

2.3.6 Reception of DTI2_DATA_IND (UDP message) in configured State (Port Unreachable)

```
UDP                            IP
  |                             |
  |       DTI2_DATA_IND         |
  |       (UDP message)         |
  *<===========================*
(UDP 1)                         |
  |                             |
  |       DTI2_DATA_REQ         |
  |          (ICMP)             |
  *===========================>*
  |                          (IP 1)
  |                             |
  |       DTI2_GETDATA_REQ      |
  *===========================>*
  |                           (IP 2)
  |                             |
```

(UDP 1)
The UDP entity receives an UDP message but is unable to demultiplex the message.

(IP 1)
UDP sends an ICMP message (Port Unreachable) via DTI2_DATA_REQ.

(IP 2)
UDP indicates that the entity is ready to get the next DTI2_DATA_IND primitive from the network layer.

2.3.7 Reception of DTI2_DATA_IND (UDP message) in configured State (Checksum Error)

```
UDP                            IP
  |                             |
  |       DTI2_DATA_IND         |
  |       (UDP message)         |
  *<===========================*
(UDP 1)                         |
  |                             |
  |       DTI2_GETDATA_REQ      |
  *===========================>*
  |                           (IP 2)
  |                             |
```

(UDP 1)
The UDP entity receives an UDP message with a checksum error. The message is silently dis-carded.

(IP 2)
UDP indicates that the entity is ready to get the next DTI2_DATA_IND primitive from the network layer.

2.3.8 Reception of DTI2_DATA_IND (UDP message) in configured State

```
 WAP                         UDP                        IP
  |                           |                          |
  |                           |                          |
  |                           |      DTI2_DATA_IND       |
  |                           |      (UDP message)       |
  |                           *<=========================*
  |                         (UDP 1)                      |
  |                           |                          |
  |     DTI2_DATA_IND         |                          |
  *<==========================*                          |
(WAP 1)                       |                          |
  |                           |                          |
  |                           |      DTI2_GETDATA_REQ    |
  |                           *=========================>*
  |                           |                       (IP 1)
  |                           |                          |
```

(UDP 1)
The UDP entity receives a UDP message.

(WAP 1)
UDP demultipexes the message and forwards it to the relevant application.

(IP 1)
UDP indicates that the entity is ready to get the next DTI2_DATA_IND primitive from the network layer.

2.3.9 Reception of DTI2_DATA_IND (ICMP message) in configured State

```
 WAP                         UDP                        IP
  |                           |                          |
  |                           |                          |
  |                           |      DTI2_DATA_IND       |
  |                           |     (ICMP message)       |
  |                           *<=========================*
  |                         (UDP 1)                      |
  |                           |                          |
  |     UDP_ERROR_IND         |                          |
  *<==========================*                          |
(WAP 1)                       |                          |
  |                           |                          |
  |                           |      DTI2_GETDATA_REQ    |
  |                           *=========================>*
  |                           |                       (IP 1)
  |                           |                          |
  |     UDP_ERROR_CNF         |                          |
  *==========================>*                          |
  |                         (UDP 2)                      |
  |                           |                          |
```

(UDP 1)
The UDP entity receives an ICMP message.

(WAP 1)
UDP demultipexes the message and generates an UDP_ERROR_IND.

(IP 1)
UDP indicates that the entity is ready to get the next DTI2_DATA_IND primitive from the network layer.

(UDP 2)
The application (e.g. WAP) confirms the UDP_ERROR_IND.

2.3.10 Reception of DTI2_DATA_REQ in configured State (success)

```
 WAP                          UDP                          IP
  |                            |                            |
  |      DTI2_DATA_REQ         |                            |
  *===========================>*                            |
  |                          (UDP 1)                        |
  |                            |                            |
  |                            |     IP_ADDR_REQ            |
  |                            |   (destination address)    |
  |                            *===========================>*
  |                            |                          (IP 1)
  |                            |                            |
  |                            |     IP_ADDR_CNF            |
  |                            |    (source address,        |
  |                            |     IP_ADDR_NOERROR)       |
  |                            *<===========================*
  |                          (UDP 2)                        |
  |                            |                            |
  |                            |     DTI2_DATA_REQ          |
  |                            *===========================>*
  |                            |                           (IP 2)
  |                            |                            |
  |      DTI2_READY_IND        |                            |
  *<===========================*                            |
(WAP 1)                        |                            |
  |                            |                            |
```

(UDP 1)
UDP receives a DTI2_DATA_REQ from an application (UDP message to send).

(IP 1)
UDP requests the source IP address for the specific destination encoded in the DTI2_DATA_REQ from WAP.

(UDP 2)
IP sends UDP the requested source address.

(IP 2)
UDP compiles a UDP message and sends it via DTI2_DATA_REQ to IP

(WAP 1)
UDP indicates that the entity is ready to get the next DTI2_DATA_REQ primitive from the application.

2.3.11 Reception of DTI2_DATA_REQ in configured State (no route to destination)

```
 WAP                         UDP                        IP
  |                           |                          |
  |     DTI2_DATA_REQ         |                          |
  *==========================>*                          |
  |                        (UDP 1)                       |
  |                           |                          |
  |                           |    IP_ADDR_REQ           |
  |                           |  (destination address)   |
  |                           *=========================>*
  |                           |                        (IP 1)
  |                           |                          |
  |                           |    IP_ADDR_CNF           |
  |                           |   (source address,       |
  |                           |    IP_ADDR_NOROUTE)      |
  |                           *<=========================*
  |                        (UDP 2)                       |
  |                           |                          |
  |     UDP_ERROR_IND         |                          |
  *<==========================*                          |
(WAP 1)                       |                          |
  |                           |                          |
  |     UDP_ERROR_CNF         |                          |
  *==========================>*                          |
  |                        (UDP 3)                       |
  |                           |                          |
  |                           |                          |
  |     DTI2_READY_IND        |                          |
  *<==========================*                          |
(WAP 2)                       |                          |
  |                           |                          |
```

(UDP 1)

UDP receives a DTI2_DATA_REQ from an application (UDP message to send).

(IP 1)

UDP requests the source IP address for the specific destination encoded in the DTI2_DATA_REQ from WAP.

(UDP 2)

IP can not determine the correct source address sends a IP_ADDR_CNF with the err field set to IP_ADDR_NOROUTE.

(WAP 1)

UDP indicates that the entity is unable to send the UDP message.

(UDP 3)

The application (WAP) confirms the UDP_ERROR_IND.

(WAP 2)

UDP indicates that the entity is ready to get the next DTI2_DATA_REQ primitive from the application.

2.3.12 Reception of DTI2_DATA_REQ in configured State (source address conflict)

```
 WAP                          UDP                          IP
  |                            |                            |
  |      DTI2_DATA_REQ         |                            |
  *==========================>*                             |
  |                        (UDP 1)                          |
  |                            |                            |
  |                            |     IP_ADDR_REQ            |
  |                            |   (destination address)    |
  |                            *==========================>*
  |                            |                         (IP 1)
  |                            |                            |
  |                            |     IP_ADDR_CNF            |
  |                            |    (source address,        |
  |                            |     IP_ADDR_NOERROR)       |
  |                            *<==========================*
  |                        (UDP 2)                          |
  |                            |                            |
  |      UDP_ERROR_IND         |                            |
  *<==========================*                             |
(WAP 1)                        |                            |
  |                            |                            |
  |      UDP_ERROR_CNF         |                            |
  *==========================>*                             |
  |                        (UDP 3)                          |
  |                            |                            |
  |                            |                            |
  |      DTI2_READY_IND        |                            |
  *<==========================*                             |
(WAP 2)                        |                            |
  |                            |                            |
```

(UDP 1)

UDP receives a DTI2_DATA_REQ from an application (UDP message to send).

(IP 1)

UDP requests the source IP address for the specific destination encoded in the DTI2_DATA_REQ from WAP.

(UDP 2)

IP sends UDP the requested source address.

(WAP 1)

UDP indicates a source address conflict. (source address from IP_ADDR_CNF ≠ source address in DTI2_DATA_REQ).

(UDP 3)

The application (WAP) confirms the UDP_ERROR_IND.

(WAP 2)

UDP indicates that the entity is ready to get the next DTI2_DATA_REQ primitive from the application.

Appendices

A. Acronyms

DS-WCDMA	Direct Sequence/Spread Wideband Code Division Multiple Access

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000)	Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>