UpLink Programming Reference

Version 3.0

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision D
March 2001

Copyright and publication information

This manual reflects version 2.5 of DAVID.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

March 2001

Table of Contents

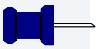
Chapter 1: Introduction to UpLink

UpLink is a client API interface that allows applications to request and send data and video to and from video servers.

UpLink is used by applications running on a DAVID device.

Introduction

UpLink is used by applications running on a DAVID® device such as a Set Top Box (STB). The library described here is used to request data from a server and to control server data flow.



Note

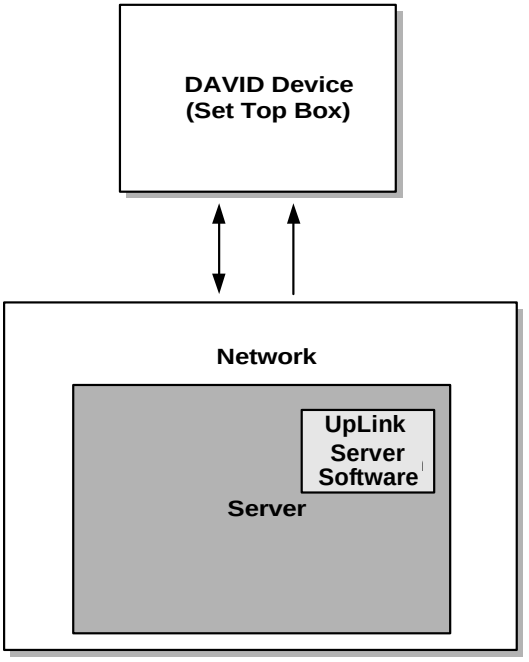
The UpLink protocol resides in the application layer of the OSI model. This document does not cover issues relating to the transport, link, or physical layers that support any protocol at this level. It assumes that the lower layer control channel facilities exist and are reliable.

UpLink is a command protocol Application Programming Interface (API) that resides between DAVID applications and the network. UpLink is used to establish network communications, request data from a server, control server data flow, and manage communications. UpLink provides a common interface layer that hides server-specific details from the application. In this way, applications can be written that are independent of the various servers or networks supporting them. Applications can communicate predictably and transparently with any network or server structure.

Hardware Structure

The UpLink library handles communications between the DAVID device and the network by mapping a standard set of library calls onto the unique services provided by the various networks.

Figure 1-1 Hardware Structure



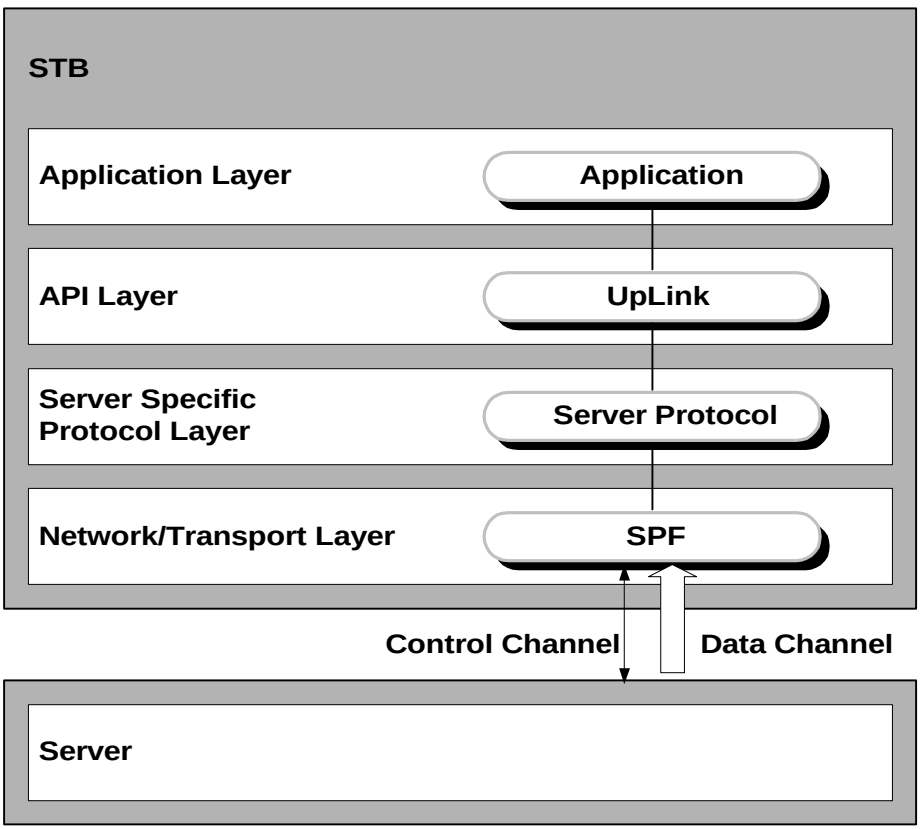
Note

The most important reason for defining a standard API is to enhance the development of application programs for use across a wide variety of networks, running on a variety of devices, and talking to a variety of servers.

Software Structure

UpLink supports several methods of dealing with the control channel. In some cases there is a constant connection; in other cases, such as a modem, there is an on-demand connection. The most common configuration is shown in **Figure 1-1**, showing the UpLink components and a two-way control channel.

Figure 1-2 UpLink Components



When the application needs to access data stored on the server, it makes an API call to UpLink requesting the data by name. UpLink bundles that request into a packet and sends it to the server through the transport protocol native to that server. Requests are always acknowledged (as `SUCCESS` or `FAILURE`) with a reply packet. Some requests also result in data delivery (either synchronously or asynchronously). Depending on the UpLink function used, this data is sent in the control channel or in the MPEG transport stream through the data channel following the reply packet.

Chapter 2: UpLink Function Descriptions

This chapter describes the UpLink API. The functions are summarized in **Table 2-1** and are arranged alphabetically.

RadiSys.
MICROWARE SOFTWARE

UpLink Functions

The UpLink functions are described in detail on the following pages. Functions are presented in alphabetical order. The table shown below, summarizes the UpLink functions.

Table 2-1 Summary of UpLink Functions

Function	Description
`ul_avc_abort()`	Abort Stream Transmission
`ul_avc_chspeed()`	Request Speed Change For Specified Stream Transmission
`ul_avc_continue()`	Continue Paused Stream Transmission
`ul_avc_jump()`	Jump to Specified Point in Stream
`ul_avc_pause()`	Pause Stream Transmission
`ul_avc_play()`	Begin a Stream Transmission
`ul_avc_readstream()`	Transmit Data Asynchronously
`ul_con_sync()`	Verify Server Connection
`ul_init()`	Initialize the UpLink API
`ul_rfa_cd()`	Change Directory
`ul_rfa_deldir()`	Delete Directory
`ul_rfa_delete()`	Delete File
`ul_rfa_dir()`	Get Directory Information

Table 2-1 Summary of UpLink Functions (continued)

Function	Description
`ul_rfa_dir_size()`	Calculate Directory Buffer Size
`ul_rfa_getdata()`	Retrieve Data From Server
`ul_rfa_getinfo()`	Get File Information
`ul_rfa_mkdir()`	Create Directory
`ul_rfa_putdata()`	Write Data To Server
`ul_rfa_setinfo()`	Modify File Attributes
`ul_term()`	Terminate UpLink API

Function Descriptions

The function descriptions are, for the most part, self-explanatory. Each section of a function description is defined below.

Syntax	Shows the function prototype with the required parameters and their data types
Description	Provides a description of the purpose of the function
Parameters	Provides details about each of the parameters
Dependencies	Lists files that are required by the original calling function
Direct Errors	Lists error codes that can be returned by the function call
Indirect Errors	Lists other functions called. See the description of those functions to locate error codes not specified in **Direct Errors** section
See Also	Lists related functions or materials that provide more information about the function

ul_avc_abort()

Abort Stream Transmission

Syntax

```
#include <uplink.h>
error_code
ul_avc_abort(
     scb          *user_scbs[]
     )
```

Description

`ul_avc_abort()` aborts a stream transmission initiated by `ul_avc_play()`.

If successful, `ul_avc_abort()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`user_scbs`	Should be the same value that was passed to `ul_avc_play()`. If `user_scbs` is not NULL, `ul_avc_abort()` will call `_os_ss_abortstream()` for each scb in the NULL terminated array.

Direct Errors

`EOS_HANGUP`	Lost synchronization with daemon
`EOS_PARAM`	Invalid `user_scbs` parameter.
`EOS_UNID`	Bad PID value in one or more of the scbs
`EOS_NOPLAY`	No stream is currently being transmitted
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

```
_os_write()
_os_read()
```

```
_os_write()
_os_ss_abortstream()
```

See Also

ul_avc_play()

ul_avc_chspeed()

Request Speed Change For Specified Stream Transmission

Syntax

```
#include <uplink.h>
error_code
ul_avc_chspeed(
     scb          *user_scbs[],
     int16        speedcode
     )
```

Description

`ul_avc_chspeed()` changes the speed of the stream transmission identified by `user_scbs` to `speedcode`.

If successful, `ul_avc_chspeed()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`user_scbs` — Should be the same value that was passed to `ul_avc_play()`.

`speedcode` — Indicates the direction and rate for `user_scb`. The guaranteed range of values is defined in `uplink.h`. Support for values outside the range specified below are Level 2 gateway dependent.

- `SPEED_RS`(-1)Negative value indicates reverse scan.
- `SPEED_NORM`0Normal play
- `SPEED_FS`1Positive value indicates forward scan

Direct Errors

EOS_HANGUP	Lost synchronization with daemon.
EOS_PARAM	Invalid SCB.
EOS_UNID	Bad PID.
EOS_NOPLAY	No stream transmission is associated with this SCB.
EOS_NOTRDY	The UpLink API has not been initialized

Indirect Errors

_os_read()
_os_write()

See Also

ul_avc_play()

ul_avc_continue()

Continue Paused Stream Transmission

Syntax

```
#include <uplink.h>
error code
ul_avc_continue(
     scb          *user_scbs[]
     )
```

Description

`ul_avc_continue()` resumes a stream transmission identified by `user_scbs` which was paused by `ul_avc_pause()`. If the stream was not paused, this call has no effect.

If successful, `ul_avc_continue()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`user_scbs`	Should be the same value that was passed to `ul_avc_play()`.

Direct Errors

`EOS_HANGUP`	Lost synchronization with daemon
`EOS_PARAM`	Invalid SCB
`EOS_NOPLAY`	No stream associated with this SCB
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

- `ul_avc_abort()`
- `ul_avc_pause()`
- `ul_avc_chspeed()`
- `ul_avc_play()`

ul_avc_jump()

Jump to Specified Point in Stream

Syntax

```
#include <uplink.h>
error_code
ul_avc_jump(
     scb          *user_scbs[],
     u_int32      position
     )
```

Description

`ul_avc_jump()` jumps to the specified point (`time`) in the stream identified by `user_scbs`.

If successful, `ul_avc_jump()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`user_scbs`	Should be the same value that was passed to `ul_avc_play()`.
`position`	Indicates the location which to jump and is formatted as `hh:mm:ss:ff` (hour, minute, second, and frame). For example, the time 7:31:15:02 is represented by 0x071f0f02.

Direct Errors

`EOS_HANGUP`	Lost synchronization with daemon
`EOS_PARAM`	Invalid SCB
`EOS_NOPLAY`	No stream associated with this SCB
`EOS_EOF`	End of file is reached
`EOS_UNID`	ID is unknown
`EOS_ILLPRM`	Illegal parameter detected

EOS_NOTRDY	The UpLink API has not been initialized

Indirect Errors

_os_read()
_os_write()

See Also

- ul_avc_abort()
- ul_avc_pause()
- ul_avc_continue()
- ul_avc_chspeed()
- ul_avc_play()

ul_avc_pause()

Pause Stream Transmission

Syntax

```
#include <uplink.h>
error_code
ul_avc_pause(
     scb          *user_scbs[]
     )
```

Description

`ul_avc_pause()` pauses the transmission of the stream identified by `user_scbs`. Transmission does not resume until the server receives a `ul_avc_continue()` call. If the data stream is already paused, this call has no effect.

If successful, `ul_avc_pause()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`user_scbs`	Should be the same value that was passed to `ul_avc_play()`.

Direct Errors

`EOS_HANGUP`	Lost synchronization with daemon
`EOS_PARAM`	Invalid SCB
`EOS_NOPLAY`	No stream is currently being transmitted
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

- `ul_avc_abort()`
- `ul_avc_continue()`
- `ul_avc_jump()`
- `ul_avc_chspeed()`
- `ul_avc_play()`

ul_avc_play()

Begin a Stream Transmission

Syntax

```
#include <uplink.h>
error_code
ul_avc_play(
     u_int32                start_time,
     const char * const     stream_name,
     scb                    *user_scbs[]
     )
```

Description

`ul_avc_play()` causes the UpLink server to transmit an MPEG-2 Transport stream to the calling application asynchronously.

If successful, `ul_avc_play()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`start_time`	The time in `hh:mm:ss:ff` (hour, minute, second, and frame) at which to begin the play.
`stream_name`	The pathname to the stream to send.
`user_scbs`	A NULL terminated aray of scb pointers. For each scb pointer in the array, `ul_avc_play()` will call `_os_ss_readstream()`. If, `user_scbs` is NULL, the stream will be requested from the UpLink server, but no `_os_ss_readstream()` calls will be made.

Direct Errors

EOS_BPNAM	NULL was specified for stream_name
EOS_HANGUP	Lost synchronization with the UpLink server
EOS_NOBUFS	Packet too large to send using internal buffer
EOS_PARAM	Invalid SCB
EOS_PNNF	Path name not found
EOS_NOTRDY	The UpLink API has not been initialized

Indirect Errors

_os_read()
_os_write()
_os_ss_readstream()
_os_ss_abortstream()

See Also

ul_avc_readstream()

ul_avc_readstream()

Transmit Data Asynchronously

Syntax

```
#include <uplink.h>
error_code
ul_avc_readstream(
     u_int32                offset,
     u_int32                *size,
     const char * const     file_name,
     scb                    *user_scb
     )
```

Description

`ul_avc_readstream()` causes the UpLink server to read the specified amount of data from a file, wrap it in an MPEG-2 Transport stream and asynchronously transmit it.

The PID used for the MPEG-2 Transport stream is the PID specified in `user_scb`.

If successful, `ul_avc_readstream()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`offset`	is the offset within the file of the data to be read.
`size`	is the number of bytes to read.
`file_name`	The name of the file containing the data requested
`user_scb`	A pointer to an scb. The scls pointed to by the scb will receive the requested data.

Direct Errors

EOS_BPNAM	Bad pathlist
EOS_DEVBSY	Data is being downloaded
EOS_HANGUP	Lost synchronization with daemon
EOS_NOBUFS	Packet too large to send using internal buffer
EOS_PARAM	Invalid SCB
EOS_PNNF	Path name not found
EOS_NOTRDY	The UpLink API has not been initialized

Indirect Errors

```
_os_read()
_os_write()
_os_ss_readstream()
_os_ss_abortstream()
```

See Also

ul_rfa_getdata()

ul_con_sync()

Verify Server Connection

Syntax

```
#include <uplink.h>
error_code
ul_con_sync(void)
```

Description

`ul_con_sync()` requests an acknowledgment from the server. This allows the application to verify that the connection with the server is still active.

If successful, `ul_con_sync()` returns SUCCESS. Otherwise, the returned value is an error code.

Direct Errors

`EOS_HANGUP`	Lost synchronization or connection with server
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

ul_init()

Initialize the UpLink API

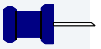
Syntax

```
#include <uplink.h>
error_code
ul_init(
     path_id      control_channel,
     path_id      data_channel
     )
```

Description

`ul_init()` allocates and initializes variables used by the UpLink API. The `control_channel` and `data_channel` parameters are stored internally so that they do not need to be specified in the other UpLink API calls.

If successful, `ul_init()` returns SUCCESS. Otherwise, the returned value is an error code.



Note

Example code for identifying and opening the UpLink control and data channels is provided in the source code for the `david_demo.l` library. The source code for this library is located in $MWOS/SRC/DAVID/DEMOS/LIBSRC/DAVID_DEMO.

Parameters

`control_channel`	An open path ID to be used as the control channel
`data_channel`	An open path ID to be used as the data channel

Indirect Errors

`EOS_MEMFUL`	`malloc()` cannot allocate the global buffer used for packet assemblly.

See Also

`ul_term()`

ul_rfa_cd()

Change Directory

Syntax

```
#include <uplink.h>
error_code
ul_rfa_cd(
     const char * const     dir_name
     )
```

Description

`ul_rfa_cd()` performs a `cd` or `chd` command on the server. Restrictions on `dir_name` are server software dependent.

If successful, `ul_rfa_cd()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`dir_name`	Pathlist to the new directory

Direct Errors

`EOS_BPNAM`	Bad pathlist
`EOS_HANGUP`	Lost synchronization with daemon
`EOS_NOBUFS`	Packet too large to send using internal buffer
`EOS_PNNF`	Path name not found.
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

ul_rfa_deldir()

Delete Directory

Syntax

```
#include <uplink.h>
error_code
ul_rfa_deldir(
     const char * const     dir_name
     )
```

Description

`ul_rfa_deldir()` recursively deletes a directory and all of its contents from the server's file system. Restrictions on `dir_name` are server software dependent.

If successful, `ul_rfa_deldir()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`dir_name`	Pathlist to the directory to delete

Direct Errors

`EOS_HANGUP`	Lost synchronization with daemon
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

`ul_rfa_delete()`
`ul_rfa_mkdir()`

ul_rfa_delete()

Delete File

Syntax

```
#include <uplink.h>
error_code
ul_rfa_delete(
     const char * const     file_name
     )
```

Description

`ul_rfa_delete()` deletes a file from the server. Restrictions on `file_name` are server software dependent. `ul_rfa_delete()` does not delete directories.

If successful, `ul_rfa_delete()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`file_name`	Pathlist to the file to be deleted

Direct Errors

`EOS_BPNAM`	Invalid file name or path list
`EOS_FNA`	File not accessible
`EOS_HANGUP`	Lost synchronization with daemon
`EOS_NOBUFS`	Packet too large to send using internal buffer
`EOS_PNNF`	Path name not found
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

`ul_rfa_putdata()`

ul_rfa_dir()

Get Directory Information

Syntax

```
#include <uplink.h>
error_code
ul_rfa_dir(
     const char * const     dir_name,
     char                   *buf,
     u_int32                *size
     )
```

Description

`ul_rfa_dir()` requests directory information from the server. Restrictions on `dir_name` are server software dependent.

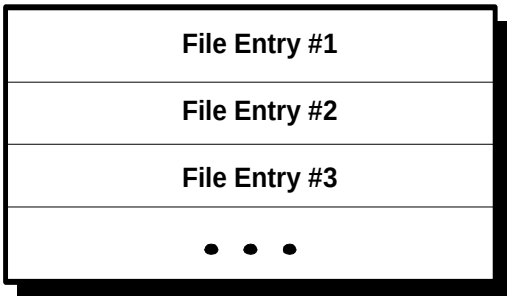
It is important to note the user specifies the maximum size of data to return. The server only returns as many directory entries as fits within that limit.

If successful, `ul_rfa_dir()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

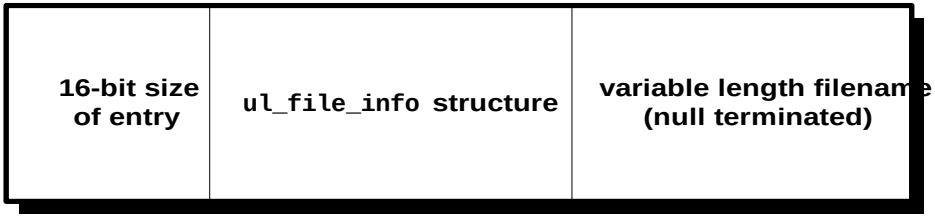
`dir_name`	Pathlist to the directory to access
`size`	Pointer to the size of the buffer. After the call returns, size contains the actual number of bytes returned.
`buf`	Pointer to buffer to receive the directory information. See in **Figure 2-1**.

Figure 2-1 Operation Results



Each file has the form as shown in **Figure 2-2**.

Figure 2-2 Form of Entry



Note

The caller specifies the maximum size of data to return. The server only returns as many directory entries as fits within that limit.

Direct Errors

`EOS_BPNAM`	Bad pathlist
`EOS_HANGUP`	Lost synchronization with daemon
`EOS_NOBUFS`	Packet too large to send using internal buffer
`EOS_PNNF`	Path name not found
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

`ul_rfa_dir_size()`

ul_rfa_dir_size()

Calculate Directory Buffer Size

Syntax

```
#include <uplink.h>
error_code
ul_rfa_dir_size(
     const char * const     dir_name,
     u_int32                *size
     )
```

Description

`ul_rfa_dir_size()` calculates how large a buffer is needed by `ul_rfa_dir()` for the entire directory pointed to by `dir_name`. Restrictions on `dir_name` are server software dependent.

If successful, `ul_rfa_dir_size()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`size`	Pointer to where `ul_rfa_dir_size()` stores the amount of buffer space required
`dir_name`	Pathlist to the directory to access

Direct Errors

`EOS_BPNAM`	Bad pathlist
`EOS_HANGUP`	Lost synchronization with daemon
`EOS_PARAM`	`NULL` parameters for size or name
`EOS_PNNF`	Path name not found
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

`ul_rfa_dir()`

ul_rfa_getdata()

Retrieve Data From Server

Syntax

```
#include <uplink.h>
error_code
ul_rfa_getdata(
     u_int16                pid,
     u_int32                offset,
     u_int32                *size,
     u_char                 mode,
     const char * const     file_name,
     void                   *data
     )
```

Description

`ul_rfa_getdata()` is a synchronous request for data from the server. It is the applications responsibility to ensure an adequate buffer size.

`ul_rfa_getdata()` returns to the caller after the requested data is received.

If successful, `ul_rfa_getdata()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`pid`	Packet ID the server should use when returning the data to the DAVID device. This parameter is ignored if you are sending data via the control channel (see `MODE_CHAN_CTRL`).
`offset`	Offset within the data from which to begin retrieving data
`size`	Number of bytes to read. `ul_rfa_getdata()` returns the actual number of bytes read and places that value in size.

`mode`	Tells the server whether to send the data via the control channel (`MODE_CHAN_CTRL`) or data channel (`MODE_CHAN_DATA`)
`file_name`	Pointer to the name of the data file. Restrictions on `file_name` are server system dependent.
`data`	Requested data is placed in the buffer pointed to by `data`

Direct Errors

`EOS_BPNAM`	Bad pathlist
`EOS_DEVBSY`	The data channel is currently in use for another request
`EOS_HANGUP`	Lost synchronization with daemon
`EOS_NOBUFS`	Packet too large to send
`EOS_PNNF`	Path name not found
`EOS_READ`	Data lost
`EOS_REQ_TIMEOUT`	More than three seconds have elapsed since the last packet was received
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

`ul_rfa_putdata()`

Note

It is the application's responsibility to ensure an adequate buffer size.

ul_rfa_getinfo()

Get File Information

Syntax

```
#include <uplink.h>
error_code
ul_rfa_getinfo(
     ul_file_info           *info,
     const char * const     file_name
     )
```

Description

`ul_rfa_getinfo()` requests information about a file on the server. Restrictions on `file_name` are server software dependent.

If successful, `ul_rfa_getinfo()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`info`	Points to a data structure used to hold information returned concerning the requested file. It contains the following fields: `size` Size (in bytes) of the file or data set `group` Group number of the data `owner` Owner of the data `perm` File permissions; a bit field defined by `UL_PERM_*` constants in `uplink.h` `modified` Last file modification date
`file_name`	Pathlist to the file

Direct Errors

EOS_BPNAM	Bad pathlist
EOS_HANGUP	Lost synchronization with daemon
EOS_NOBUFS	Packet too large to send using internal buffer
EOS_PNNF	Path name not found
EOS_NOTRDY	The UpLink API has not been initialized

Indirect Errors

_os_read()
_os_write()

See Also

ul_rfa_setinfo()

ul_rfa_mkdir()

Create Directory

Syntax

```
#include <uplink.h>
error_code
ul_rfa_makdir(
     const char * const     dir_name,
     u_int16                modes
     )
```

Description

`ul_rfa_mkdir()` creates a directory on the server. Restrictions on `dir_name` are server software dependent.

If successful, `ul_rfa_mkdir()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`dir_name`	Pathlist to the directory to create
`modes`	Bit field specifying the file attributes of the new directory. Values for this field are the `UL_PERM_*` constants defined in `uplink.h`.

Direct Errors

`EOS_HANGUP`	Lost synchronization with daemon
`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`_os_read()`
`_os_write()`

See Also

`ul_rfa_deldir()`
`ul_rfa_cd()`

ul_rfa_putdata()

Write Data To Server

Syntax

```
#include <uplink.h>
error_code
ul_rfa_putdata(
     u_int32                offset,
     u_int32                size,
     const void             *data,
     const char * const     file_name
     )
```

Description

`ul_rfa_putdata()` synchronously writes data to the server. It adds data to or replaces data within a file. If the file does not exist, it is created. If the client wants to replace an entire file, it must first call `ul_rfa_delete()` to delete the file. Restrictions on `file_name` are server software dependent.

`ul_rfa_putdata()` returns to the DAVID device after the data is completely sent and acknowledged.

If successful, `ul_rfa_putdata()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`offset`	Offset within the file to begin storing data
`file_name`	Pathlist to the file
`size`	Number of bytes to write
`data`	Data to store in `file_name`

Direct Errors

EOS_BPNAM	Bad pathlist
EOS_HANGUP	Lost synchronization with daemon
EOS_NOBUFS	Packet too large to send using internal buffer
EOS_PNNF	Path name not found
EOS_NOTRDY	The UpLink API has not been initialized

Indirect Errors

_os_read()
_os_write()

See Also

ul_rfa_getdata()

ul_rfa_setinfo()

Modify File Attributes

Syntax

```
#include <uplink.h>
error_code
ul_rfa_setinfo(
     const ul_file_info * const  info,
     const char * const          file_name
     )
```

Description

`ul_rfa_setinfo()` modifies the attributes of a file on the server. It replaces the attributes of the file pointed to by `file_name` with the attributes specified by the data structure pointed to by `info`. Restrictions on `file_name` are server software dependent.

If successful, `ul_rfa_setinfo()` returns SUCCESS. Otherwise, the returned value is an error code.

Parameters

`info`	Points to a data structure used to specify the new attributes for `file_name`. It contains the following fields: `size`Size (in bytes) of the file or data set `group`Group number of the file `owner`Owner of the data `perm`File permissions; a bit field defined by `UL_PERM_*` constants in `uplink.h` `modified`Last file modification date
`file_name`	Pathlist to the file to modify

Direct Errors

EOS_BPNAM	Bad pathlist
EOS_HANGUP	Lost synchronization with daemon
EOS_NOBUFS	Packet too large to send using internal buffer
EOS_PNNF	Path name not found
EOS_NOTRDY	The UpLink API has not been initialized

Indirect Errors

_os_read()
_os_write()

See Also

ul_rfa_getinfo()

ul_term()

Terminate UpLink API

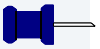
Syntax

```
#include <uplink.h>
error_code
ul_term(void)
```

Description

`ul_term()` de-initializes the UpLink API. This call allows the library to release global resources allocated on behalf of the application. After a `ul_term()` call, no other UpLink API calls are allowed until another `ul_init()` call is made.

If successful, `ul_term()` returns SUCCESS. Otherwise, the returned value is an error code.



Note

It is the responsibility of the application to close the open control and data paths after calling `ul_term()`.
Closing these paths terminates the UpLink session.

Parameters

None. This call does not communicate with the server.

Direct Errors

`EOS_NOTRDY`	The UpLink API has not been initialized

Indirect Errors

`none`

See Also

`ul_init()`

Index

Symbols

A

C

D

E

G

I

J

L

M

P

R

S

T

U

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:_____

Company:_____

Phone:_____

Fax:_____Email:_____

Product Name:

Description of Problem:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Host Platform_____

Target Platform_____

RadiSys.

MICROWARE SOFTWARE