RTP Protocol Modules for TTCN-3 Toolset with TITAN DESCRIPTION

Contents

[1 Introduction 2](#_Toc427320966)

[1.1 How to Read this Document 2](#_Toc427320967)

[1.2 Scope 2](#_Toc427320968)

[1.3 References 2](#_Toc427320969)

[1.4 Abbreviations 2](#_Toc427320970)

[1.5 Terminology 3](#_Toc427320971)

[1.6 System Requirements 3](#_Toc427320972)

[2 Protocol Modules 3](#_Toc427320973)

[2.1 Overview 3](#_Toc427320974)

[2.2 Installation 4](#_Toc427320975)

[2.3 Configuration 4](#_Toc427320976)

[3 Functional specification 4](#_Toc427320977)

[3.1 Protocol version implemented 4](#_Toc427320978)

[3.2 Modifications/deviations related to the protocol specification 4](#_Toc427320979)

[3.2.1 Unimplemented Messages, Information Elements and Constants 4](#_Toc427320980)

[3.2.2 Protocol Modifications/Deviations 4](#_Toc427320981)

[3.3 Encoding/Decoding Functions 4](#_Toc427320982)

[3.4 Media stream handling in RTP Protocol 5](#_Toc427320983)

[3.4.1 General 5](#_Toc427320984)

[3.4.2 Supported media files 5](#_Toc427320985)

[3.4.3 Reading/Writing and Other Related Functions 5](#_Toc427320986)

[4 Error messages 9](#_Toc427320987)

[5 Warning messages 11](#_Toc427320988)

[6 Examples 12](#_Toc427320989)

[6.1 Mapping module 12](#_Toc427320990)

[6.1.1 Open session 12](#_Toc427320991)

[6.1.2 Close session 12](#_Toc427320992)

[6.2 Payload generation 12](#_Toc427320993)

[6.2.1 Comfort noise 12](#_Toc427320994)

[6.2.2 Telephony events and DTMF codes 13](#_Toc427320995)

[6.2.3 Codec handling 13](#_Toc427320996)

[6.2.4 Example code 14](#_Toc427320997)

# Introduction

## How to Read this Document

This is the Function Description for the RTP protocol module. The RTP protocol module is developed for the TTCN-3 Toolset with TITAN. This document should be read together with Product Revision Information [2].

## Scope

The purpose of this document is to specify the content of the RTP protocol modules. Basic knowledge of TTCN-3 **Error! Reference source not found.** and TITAN TTCN-3 Test Executor **Error! Reference source not found.** is valuable when reading this document.

## References

1. ETSI ES 201 873-1 v.3.1.1 (2005-06)  
   The Testing and Test Control Nota­tion version 3. Part 1: Core Language
2. 1/198 17-CRL 113 200/5 Uen  
   User Guide for TITAN TTCN–3 Test Executor
3. 109 21-CNL 113 392-5 Uen  
   RTP Protocol Modules for TTCN-3 Toolset with TITAN, Product Revision Information
4. RFC 3550  
   RTP: A Transport protocol for Real-Time Applications
5. RFC 3389  
   Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)
6. RFC 2833  
   RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals
7. RFC 2190  
   RTP Payload Format for H.263 Video Streams
8. RFC 2435  
   RTP Payload Format for JPEG-compressed Video
9. 1551-CNL 113 769 Uen

Function Description for the SRTP Protocol Module

## Abbreviations

ASP Abstract Service Primitive

RTP Real-time Transport Protocol

RTCP RTP Control Protocol

TTCN-3 Testing and Test Control Notation version 3

UDP User Datagram Protocol

## Terminology

No specific terminology is used.

## System Requirements

Protocol modules are a set of TTCN-3 source code files that can be used as part of TTCN-3 test suites only. Hence, protocol modules alone do not put specific requirements on the system used. However in order to compile and execute a TTCN-3 test suite using the set of protocol modules the following system requirements must be satisfied:

* TITAN TTCN-3 Test Executor Executor version R7A (1.7.pl0) or higher installed. For installation guide see ‎[13]. Please note: This version of the protocol module is not compatible with TITAN releases earlier than R7A.

# Protocol Modules

## Overview

Protocol modules implement the message structure of the related protocol in a formalized way, using the standard specification language TTCN-3. This allows defining of test data (templates) in the TTCN-3 language ‎[1] is and correctly encoding/decoding messages when executing test suites using the Titan TTCN-3 test environment.

Additionally, there are some implemented functions, which are able to read/write media files. The return value of the reading function is in the RTP payload format according to the media. The incoming parameter of the writing function is an octetstring (an RTP payload format packet).

Protocol module uses Titan’s RAW encoding attributes and hence is usable with the Titan test toolset only.

## Installation

* The set of protocol modules can be used for developing TTCN-3 test suites using any text editor. However to make the work more efficient a TTCN-3-enabled text editor is recommended (e.g. nedit, xemacs). Since the RTP protocol is used as a part of a TTCN-3 test suite, this requires TTCN-3 Test Executor be installed before the module can be compiled and executed together with other parts of the test suite. For more details on the installation of TTCN-3 Test Executor see the relevant section of ‎[13].

## Configuration

None.

# Functional specification

## Protocol version implemented

This set of protocol modules implement protocol messages and constants of the RTP protocol [4] with the modifications specified in ‎3.2.

The RTCP messages described in RFC4585 & RFC 5104 has been implemented

## Modifications/deviations related to the protocol specification

### Unimplemented Messages, Information Elements and Constants

None.

### Protocol Modifications/Deviations

Protocol modules contain the following deviations from ‎:

* The protocol module does not support the encryption of the messages. SRTP\_CNL113769 can encrypt and decrypt the messages. See [9]

## Encoding/Decoding Functions

This product contains encoding/decoding functions, which assure correct encoding of messages when sent from Titan and correct decoding of messages when received by Titan. Implemented encoding/decoding functions:

Name Type of formal parameters Type of return value

f\_RTP\_enc (in RTP\_messages\_union pdu) octetstring;

f\_RTP\_dec (in octetstring data) RTP\_messages\_union;

f\_RTP\_packet\_enc (in RTP\_packet pdu) octetstring;

f\_RTP\_packet\_dec (in octetstring data) RTP\_packet;

## Media stream handling in RTP Protocol

### General

Each RTP packet, starts with a fixed RTP header followed by payload format ‎[4].

### Supported media files

JPEG JPEG video codec ‎[6] (Note2)

H263 H263 video codec (‎[5] mode: A)

MPEG4 MPEG4 video codec (Note1)

GIF GIF image codec (Note1)

f3GP 3gp file format (Note1)

fMPEG4 MPEG4 file format (Note1)

UNKNOWN other (Note1)

Note1: The payload is handled as an octetstream.

Note2: The f\_Put\_Media\_Content function sets the fragment\_offset field of the payload header structure to zero when writing the payload header into the file.

### Reading/Writing and Other Related Functions

This product also contains read/write functions so that we can read any blocks and the function returns RTP payload format, and write received RTP payload packets into a file. Implemented encoding/decoding functions:

Types of formal parameters:

Type Description

InitOperType enumerated (OPEN, CREATE)

RTP\_MediaType enumerated (JPEG, H263, MPEG4, GIF,

f3GP, fMPEG4, UNKNOWN)

RTP\_FileInfo record with the following fields:

integer fd, // file description

integer block\_size, // size of block

integer block\_no, // starting block position

integer nof\_blocks\_to\_read, // number of blocks to read

RTP\_MediaType mediaType, // type of media

boolean headerOp, // true: get or put header

integer headerOffset, // size of media header

Media\_RTP\_Header mediaHeader // header of the media

Implemented functions:

Name Type of formal parameters Type of return value

f\_Init\_Media\_Fileinfo (

in charstring pl\_filename,

in integer pl\_block\_size,

in InitOperType pl\_init\_type,

in RTP\_MediaType pl\_media\_type,

in integer pl\_blockno,

in integer pl\_nof\_blocks,

inout RTP\_FileInfo pl\_fileinfo ) none

The f\_Init\_Media\_Fileinfo opens (OPEN) or creates (CREATE) the file to read or write the blocks. This function sets the “FileInfo” parameter that contains the fd, block\_size, block\_no, nof\_blocks\_to\_read, Header\_Offset, etc. values.

Name Type of formal parameters Type of return value

f\_Get\_Media\_Content (

inout RTP\_FileInfo pl\_filename ) octetstring

This function reads the blocks from the file and encapsulates it according to RTP payload format. The function encodes this packet to octetstring.

Name Type of formal parameters Type of return value

f\_Put\_Media\_Content (

in RTP\_FileInfo pl\_fileinfo,

in octetstring data ) integer

This function decodes the incoming octetstring (RTP payload format packet) and writes the appropriate data into the file.

These functions may be useful when we want to manipulate the files directly.

Name Type of formal parameters Type of return value

f\_INIT\_CODEC (

in charstring pl\_filename,

in integer pl\_block\_size,

in InitOperType pl\_init\_type ) integer

The f\_INIT\_CODEC opens (OPEN) or creates (CREAT) the file to read or to write the blocks and sets the size of the blocks. It returns the identifier of the file.

Name Type of formal parameters Type of return value

f\_GET\_CONTENT (

in integer pl\_fd,

in integer pl\_blockno,

in integer pl\_nof\_blocks\_to\_read

in integer pl\_header\_offset ) octetstring

The f\_GET\_CONTENT reads the blocks from the file. It reads nof\_blocks\_to\_read blocks starting from (blockno + header\_offset).

Name Type of formal parameters Type of return value

f\_PUT\_CONTENT (

in integer pl\_fd,

in integer pl\_blockno,

in octetstring pl\_stream,

in integer pl\_header\_offset ) octetstring

This function writes the blocks into the file. It writes nof\_blocks\_to\_read blocks starting from (blockno + header\_offset).

Name Type of formal parameters Type of return value

f\_CLOSE\_CODEC ( in integer pl\_fd ) none

The f\_CLOSE\_CODEC close the file.

There are further auxiliary inside functions that are not for direct use of the user (they are used by the other functions).

Encoding/decoding functions for RTP payload formats of media:

Name Type of formal parameters Type of return value

f\_RTP\_Hdr\_enc (

in Media\_RTP\_Header hdr) octetstring

f\_JPEG\_RTP\_Hdr\_dec (

in octetstring stream ) JPEG\_RTP\_Header

f\_H263\_RTP\_Hdr\_dec (

in octetstring stream ) H263\_RTP\_Header

f\_RTP\_Data\_enc(

in Media\_RTP\_Data rtp\_data ) octetstring

f\_JPEG\_RTP\_Data\_dec(

in octetstring stream ) JPEG\_RTP

f\_H263\_RTP\_Data\_dec(

in octetstring stream ) H263\_RTP

Other inside functions:

f\_Count\_JPEG\_Header\_Offset( in FileInfo\_t fi);

void log\_info\_list()

int f\_Fileinfo\_Check(const int& fd, const Operation& OPERATION)

int f\_Operation\_Check(const int& fd, const int& blockno, const int& nof\_b, const Operation& OPERATION, const int& hdr\_off)

# Error messages

ERROR: “Wrong media type setting!”

ERROR: “INIT\_\_CODEC: empty filename is not allowed”

ERROR: “INIT\_\_CODEC: Block size must be a positive integer”

ERROR: “INIT\_\_CODEC: There is not enough memory.”

ERROR: “INIT\_\_CODEC: Cannot open file '%s'", filename

ERROR: “INIT\_\_CODEC: Cannot create file '%s'", filename

ERROR: “INIT\_\_CODEC: Wrong init\_type setting! Available: OPEN, CREATE."

ERROR: “INIT\_\_CODEC: Cannot gather file info"

ERROR: “GET\_CONTENT: There is not enough memory."

ERROR: “GET\_CONTENT: unsuccesful read (%d)", read\_data

ERROR: “Get\_Media\_Header: Header offset must be a non-negative integer!"

ERROR: “Get\_Media\_Header: Size of file %s is smaller than the size of header!", filename

ERROR: “Count\_JPEG\_Header\_Offset: Cannot set the starting position in file %s", filename

ERROR: “Count\_JPEG\_Header\_Offset: There is not enough memory."

ERROR: “Count\_JPEG\_Header\_Offset: unsuccesful read (%d)", read\_data

ERROR: “PUT\_CONTENT: unsuccesful write to file"

ERROR: “CLOSE\_\_CODEC: There is not enough memory."

ERROR: “CLOSE\_\_CODEC: Unknown file descriptor (%d)", fd

ERROR: “Fileinfo\_Check: Unknown file descriptor (%d)", fd

ERROR: “Fileinfo\_Check: file info list is empty"

ERROR: “Fileinfo\_Check: inconsistent file info list (filename is missing)"

ERROR: “Fileinfo\_Check: inconsistent file info list"

ERROR: “Fileinfo\_Check: Cannot gather file info"

ERROR: “Operation\_Check: The number of blocks to read must be a non-negative integer"

ERROR: “Operation\_Check: Starting block position must be a non-negative integer"

ERROR: “Operation\_Check: Header offset must be a non-negative integer"

ERROR: “Operation\_Check: Size of file %s is smaller than the starting block position", filename

ERROR: “Operation\_Check: Cannot set the starting position in file %s", filename

ERROR: “Operation\_Check: Wrong OPERATION setting! Available: READ, WRITE."

ERROR: “RTP\_Hdr\_enc: The incoming parameter (hdr) is unbound!"

ERROR: “RTP\_Data\_enc: The incoming parameter (rtp\_data) is unbound!"

# Warning messages

WARNING: “INIT\_\_CODEC: File %s contains uncomplete blocks", filename

WARNING: “Operation\_Check: This is an uncomplete block. Size: %d

byte/bytes.", bytes\_to\_operation

# Examples

The “demo” directory of the deliverable contains the following examples and functions:

## Mapping module

The mapping module provides the connection between the RTP protocol module and the UDP test port. It encodes and decodes the RTP messages and manages the opening and closing of RTP sessions.

### Open session

New session is requested by the ASP\_RTP\_Open\_session message. The session\_id contains the requested parameter of the new session.

* id: The unique identifier of the session. It must be omitted. It will be assigned by the mapping module
* local\_address: The local ip address. If it is omitted the default is any address.
* local\_port: The local port number. If omitted a random port number will be used.
* dest\_address and dest\_port: contains the address and port number of the remote host. If specified this address will be the default remote address for the session.

The mapping module answers the open request with ASP\_RTP\_Open\_session\_result message. That message contains the parameters of the new session. The session\_id.id is the unique identifier of the session. It will identify the session during sending and receiving data.

### Close session

The closing of the session is requested by the ASP\_RTP\_Close\_session message. The session\_id.id contains the session identifier.

## Payload generation

### Comfort noise

The following function generates a comfort noise payload according to ‎[6].

Name Type of formal parameters Type of return value

f\_generate\_comfort\_noise (in integer level,

in Coefficient\_list coefficients) octetstring;

Parameters:

• level: Noise level value

• coefficients: List of reflection coefficients

### Telephony events and DTMF codes

The following functions generates a telephony event and DTMF codes payload according to ‎[7].

Name Type of formal parameters Type of return value

f\_generate\_tones\_events

(in Tones\_DTMFs events\_dtmfs) octetstring;

Parameter:

• events\_dtmfs: List of DTMF digits, events or tones.

### Codec handling

The demo program (example.ttcn) introduces many examples of payload generation between two UDP testports. The following functions read and write samles from/to files:

Name Type of formal parameters Type of return value

f\_Init\_Media\_Fileinfo (

in charstring pl\_filename,

in integer pl\_block\_size,

in InitOperType pl\_init\_type,

in RTP\_MediaType pl\_media\_type,

in integer pl\_blockno,

in integer pl\_nof\_blocks,

inout RTP\_FileInfo pl\_fileinfo ) none

f\_Get\_Media\_Content (

inout RTP\_FileInfo pl\_filename ) octetstring

f\_Put\_Media\_Content (

in RTP\_FileInfo pl\_fileinfo,

in octetstring data ) integer

f\_INIT\_CODEC (

in charstring pl\_filename,

in integer pl\_block\_size,

in InitOperType pl\_init\_type ) integer

f\_GET\_CONTENT (

in integer pl\_fd,

in integer pl\_blockno,

in integer pl\_nof\_blocks\_to\_read

in integer pl\_header\_offset ) octetstring

f\_PUT\_CONTENT (

in integer pl\_fd,

in integer pl\_blockno,

in octetstring pl\_stream,

in integer pl\_header\_offset ) octetstring

f\_CLOSE\_CODEC ( in integer pl\_fd ) none

f\_RTP\_Hdr\_enc (

in Media\_RTP\_Header hdr) octetstring

### Example code

module example{

modulepar {

integer BLOCK\_SIZE := 4;

integer BLOCK\_NO := 0;

integer NOF\_BLOCKS\_TO\_READ := 6;

}

import from UDPasp\_Types all;

import from UDPasp\_PortType all;

import from RTP\_Types all;

import from RTP\_Mapping all;

import from RTP\_File\_Types all;

import from RTP\_Media all;

template ASP\_RTP\_Open\_session\_result t\_open\_res:=?;

template ASP\_RTP\_message t\_message:=?;

function TWAIT( in integer sec ) runs on test\_comp

{

timer T\_WAIT;

T\_WAIT.start(int2float(sec));

T\_WAIT.timeout;

}

type component test\_comp{

var RTP\_mapping\_CT v\_mapping\_comp;

var RTP\_mapping\_CT v\_mapping\_comp2;

port RTPasp\_PT RTP1;

port RTPasp\_PT RTP2;

}

type component system\_comp{

port UDPasp\_PT UDP1;

port UDPasp\_PT UDP2;

}

function make\_pattern\_file(in charstring FileName1,

in charstring FileName2,

in Media\_RTP\_Header mrh,

inout ASP\_RTP\_message v\_message)

runs on test\_comp

{

var ASP\_RTP\_message v\_message1;

// Init codec

var integer v\_codec1:=f\_INIT\_CODEC(FileName1, BLOCK\_SIZE, OPEN);

var integer v\_codec2:=f\_INIT\_CODEC(FileName2, BLOCK\_SIZE, CREATE);

// Send and receive message

var integer hdr\_size := f\_PUT\_CONTENT(v\_codec2,0,f\_RTP\_Hdr\_enc(mrh),0)

var integer block\_no := BLOCK\_NO;

var boolean next := true;

do {

// read samples from file

v\_message.data.rtp.data :=

f\_GET\_CONTENT(v\_codec1, block\_no, NOF\_BLOCKS\_TO\_READ, 0);

if ( v\_message.data.rtp.data!=''O ) {

RTP1.send(v\_message);

RTP2.receive(t\_message) -> value v\_message1;

if (f\_PUT\_CONTENT(v\_codec2, block\_no,

v\_message1.data.rtp.data, hdr\_size) <

NOF\_BLOCKS\_TO\_READ\*BLOCK\_SIZE) {

next := false;

}

else {

block\_no := block\_no + NOF\_BLOCKS\_TO\_READ;

}

}

else { next := false; }

} while (next);

// Close codec

f\_CLOSE\_CODEC(v\_codec1);

f\_CLOSE\_CODEC(v\_codec2);

}

function send\_receive\_file( in charstring FileName1,

in charstring FileName2,

in boolean HEADER,

in RTP\_MediaType mt,

inout ASP\_RTP\_message v\_message)

runs on test\_comp

{

var ASP\_RTP\_message v\_message1;

var RTP\_FileInfo FileInfo1, FileInfo2;

f\_Init\_Media\_Fileinfo( FileName1, BLOCK\_SIZE, OPEN, mt, BLOCK\_NO,

NOF\_BLOCKS\_TO\_READ, FileInfo1);

f\_Init\_Media\_Fileinfo( FileName2, BLOCK\_SIZE, CREATE, mt, BLOCK\_NO,

NOF\_BLOCKS\_TO\_READ, FileInfo2);

var boolean next := true;

do {

v\_message.data.rtp.data := f\_Get\_Media\_Content( FileInfo1 );

if ( v\_message.data.rtp.data!=''O ) {

RTP1.send(v\_message);

RTP2.receive(t\_message) -> value v\_message1;

// HEADER == 0 : get file without payload header (.org)

if ( HEADER==false ) {

FileInfo2.headerOffset := 0;

FileInfo2.headerOp := false;

}

if (f\_Put\_Media\_Content(FileInfo2,v\_message1.data.rtp.data)==0){

next := false;

}

else {

FileInfo1.block\_no :=

FileInfo1.block\_no + FileInfo1.nof\_blocks\_to\_read;

FileInfo2.block\_no :=

FileInfo2.block\_no + FileInfo2.nof\_blocks\_to\_read;

}

}

else { next := false; }

} while (next);

f\_CLOSE\_CODEC(FileInfo1.fd);

f\_CLOSE\_CODEC(FileInfo2.fd);

}

testcase TC() runs on test\_comp system system\_comp {

var RTP\_session\_par v\_session\_par;

var ASP\_RTP\_Open\_session v\_open, v\_open2;

var ASP\_RTP\_Open\_session\_result v\_open\_res, v\_open\_res2;

var ASP\_RTP\_message v\_message,v\_message1,v\_message2;

var ASP\_RTP\_Close\_session v\_close;

// Create and start mapping components

v\_mapping\_comp:=RTP\_mapping\_CT.create;

map(v\_mapping\_comp:UDP\_PCO, system:UDP1);

connect(self:RTP1,v\_mapping\_comp:RTP\_SP\_PCO);

v\_mapping\_comp.start(f\_RTP\_EncDec\_Mapping());

v\_mapping\_comp2:=RTP\_mapping\_CT.create;

map(v\_mapping\_comp2:UDP\_PCO, system:UDP2);

connect(self:RTP2,v\_mapping\_comp2:RTP\_SP\_PCO);

v\_mapping\_comp2.start(f\_RTP\_EncDec\_Mapping());

// Open Session

v\_session\_par.id:=omit;

v\_session\_par.local\_address:="localhost";

v\_session\_par.local\_port:=5679;

v\_session\_par.dest\_address:="localhost";

v\_session\_par.dest\_port:=5060;

v\_open.session\_id:=v\_session\_par;

RTP1.send(v\_open);

RTP1.receive(t\_open\_res) -> value v\_open\_res;

// Open Session2

v\_session\_par.id:=omit;

v\_session\_par.local\_address:="localhost";

v\_session\_par.local\_port:=5060;

v\_session\_par.dest\_address:="localhost";

v\_session\_par.dest\_port:=5679;

v\_open2.session\_id:=v\_session\_par;

RTP2.send(v\_open2);

RTP2.receive(t\_open\_res) -> value v\_open\_res2;

// Set message

v\_session\_par:=v\_open\_res.session\_id;

v\_session\_par.local\_address:=omit;

v\_session\_par.local\_port:=omit;

v\_session\_par.dest\_address:="localhost";

v\_session\_par.dest\_port:=5060;

v\_message.session\_id:=v\_session\_par;

v\_message.data:={

rtp:={ version:=2,

padding\_ind:= '0'B,

extension\_ind:='0'B,

CSRC\_count:=0,

marker\_bit:='0'B,

payload\_type:=11,

sequence\_number:=52134,

time\_stamp:='11110000101010101100110000001111'B,

SSRC\_id:='00001111010101010011001111110000'B,

CSRCs:={'11110000101010101100110000001111'B},

ext\_header:=omit

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// make a pattern JPEG file with RTP payload header

var Media\_RTP\_Header mrh1 := {

jpeg\_rtp\_hdr := {{1,0,3,2,51,6},

{1,'0'B,'1'B,2},

{9,8,11,{250,134,255,99,1,23,99,45,32,2,8}}

}};

make\_pattern\_file("sample.media", "jpeg\_pattern.dat", mrh1, v\_message);

// Send and receive JPEG media file

send\_receive\_file("jpeg\_pattern.dat", "jpeg\_rtp.dat",

true, JPEG, v\_message);

send\_receive\_file("jpeg\_pattern.dat", "jpeg\_rtp.org",

false, JPEG, v\_message);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// make a pattern H263 file with RTP payload header

var Media\_RTP\_Header mrh2 := {

h263\_rtp\_hdr := {'0'B,'1'B,'011'B,'100'B,

'001'B,'0'B,'1'B,'1'B,'0'B,

'0110'B,'10'B,'101'B,

'01101011'B}

};

make\_pattern\_file("sample.media", "h263\_pattern.dat", mrh2, v\_message);

// Send and receive H263 media file

send\_receive\_file("h263\_pattern.dat", "h263\_rtp.dat",

true, H263, v\_message);

send\_receive\_file("h263\_pattern.dat", "h263\_rtp.org",

false, H263, v\_message);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Send and receive MPEG4 media file

send\_receive\_file("sample.media", "mpeg4\_rtp.dat",

true, MPEG4, v\_message);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Close the session

v\_close.session\_id:=v\_session\_par;

RTP1.send(v\_close);

RTP2.send(v\_close);

TWAIT(1);

v\_mapping\_comp.stop;

disconnect(self:RTP1,v\_mapping\_comp:RTP\_SP\_PCO);

unmap(v\_mapping\_comp:UDP\_PCO, system:UDP1);

v\_mapping\_comp2.stop;

disconnect(self:RTP2,v\_mapping\_comp2:RTP\_SP\_PCO);

unmap(v\_mapping\_comp:UDP\_PCO, system:UDP2);

}

control{

execute(TC());

}

}