
Information technology — Generic coding of moving pictures and associated audio information —

Part 5: Software simulation

*Technologies de l'information — Codage générique des images animées et
des informations sonores associées —*

Partie 5: Simulation de logiciel



Contents

1 GENERAL..... 1

1.1 SCOPE..... 1

1.2 NORMATIVE REFERENCES 1

2 TECHNICAL ELEMENTS 2

2.1 DEFINITIONS 2

2.2 SYMBOLS AND ABBREVIATIONS 13

3 SYSTEMS SIMULATION..... 14

3.1 SOFTWARE INFORMATION 14

3.2 SYSTEMS DOCUMENTATION FILES 15

3.2.1 *Readme* 15

3.2.2 *doc.tex*..... 15

3.3 SYSTEMS CODE LISTINGS..... 18

3.3.1 *Consumer.C* 18

3.3.2 *Consumer.H*..... 18

3.3.3 *Decoder.C*..... 18

3.3.4 *Decoder.H*..... 19

3.3.5 *Directory.C* 20

3.3.6 *Directory.H*..... 21

3.3.7 *Encoder.C* 22

3.3.8 *Encoder.H*..... 23

3.3.9 *Events.C* 24

3.3.10 *Events.H*..... 24

3.3.11 *IPortFromFile.C* 24

3.3.12 *IPortFromFile.H*..... 25

3.3.13 *IPortFromRam.C* 25

3.3.14 *IPortFromRam.H*..... 26

3.3.15 *InputPort.C* 26

3.3.16 *InputPort.H*..... 27

3.3.17 *Makefile* 28

3.3.18 *OPortToFile.C* 28

3.3.19 *OPortToFile.H*..... 28

3.3.20 *OPortToRam.C* 29

3.3.21 *OPortToRam.H*..... 29

3.3.22 *OutputPort.C*..... 29

3.3.23 *OutputPort.H* 30

3.3.24 *PES.C*..... 31

3.3.25 *PES.H*..... 32

3.3.26 *PESConsumer.C*..... 33

3.3.27 *PESConsumer.H* 34

3.3.28 *PESProducer.C*..... 35

3.3.29 *PESProducer.H*..... 36

3.3.30 *Poly.C* 37

3.3.31 *Poly.H*..... 37

3.3.32 *Producer.C*..... 37

3.3.33 *Producer.H* 38

3.3.34 *Section.C*..... 38

3.3.35	<i>Section.H</i>	38
3.3.36	<i>SectionConsumers.C</i>	39
3.3.37	<i>SectionConsumers.H</i>	41
3.3.38	<i>SectionProducers.C</i>	41
3.3.39	<i>SectionProducers.H</i>	42
3.3.40	<i>TS.C</i>	42
3.3.41	<i>TS.H</i>	43
3.3.42	<i>TSConsumer.C</i>	44
3.3.43	<i>TSConsumer.H</i>	45
3.3.44	<i>TSProducer.C</i>	45
3.3.45	<i>TSProducer.H</i>	46
3.3.46	<i>TimeStamp.C</i>	46
3.3.47	<i>TimeStamp.H</i>	47
3.3.48	<i>Utilities.C</i>	47
3.3.49	<i>Utilities.H</i>	47
3.3.50	<i>decode.C</i>	48
3.3.51	<i>encode.C</i>	49
3.3.52	<i>pesin.C</i>	49
3.3.53	<i>pesout.C</i>	50
4	VIDEO SIMULATION	51
4.1	VIDEO DOCUMENTATION FILES	52
4.1.1	<i>README</i>	52
4.1.2	<i>doc/ARCHITECTURE</i>	53
4.1.3	<i>doc/mpeg2dec.doc</i>	53
4.1.4	<i>doc/mpeg2enc.doc</i>	54
4.2	VIDEO ENCODER LISTINGS	58
4.2.1	<i>encoder/CHANGES</i>	58
4.2.2	<i>encoder/Makefile</i>	58
4.2.3	<i>encoder/config.h</i>	58
4.2.4	<i>encoder/conform.c</i>	58
4.2.5	<i>encoder/fdctref.c</i>	60
4.2.6	<i>encoder/global.h</i>	60
4.2.7	<i>encoder/idct.c</i>	62
4.2.8	<i>encoder/motion.c</i>	62
4.2.9	<i>encoder/mpeg2enc.c</i>	69
4.2.10	<i>encoder/mpeg2enc.h</i>	72
4.2.11	<i>encoder/predict.c</i>	72
4.2.12	<i>encoder/putbits.c</i>	75
4.2.13	<i>encoder/puthdr.c</i>	75
4.2.14	<i>encoder/putmpg.c</i>	76
4.2.15	<i>encoder/putpic.c</i>	77
4.2.16	<i>encoder/putseq.c</i>	78
4.2.17	<i>encoder/putvlc.c</i>	80
4.2.18	<i>encoder/quantize.c</i>	81
4.2.19	<i>encoder/ratectl.c</i>	82
4.2.20	<i>encoder/readpic.c</i>	84
4.2.21	<i>encoder/stats.c</i>	86
4.2.22	<i>encoder/transfrm.c</i>	88
4.2.23	<i>encoder/vlc.h</i>	89
4.2.24	<i>encoder/writpic.c</i>	91
4.3	VIDEO DECODER LISTINGS	92
4.3.1	<i>decoder/CHANGES</i>	92
4.3.2	<i>decoder/EXAMPLES</i>	92
4.3.3	<i>decoder/IEEE1180</i>	92
4.3.4	<i>decoder/Makefile</i>	93
4.3.5	<i>decoder/README</i>	94

4.3.6	<i>decoder/SPATIAL.DOC</i>	94
4.3.7	<i>decoder/TODO</i>	95
4.3.8	<i>decoder/config.h</i>	95
4.3.9	<i>decoder/display.c</i>	95
4.3.10	<i>decoder/getbits.c</i>	101
4.3.11	<i>decoder/getblk.c</i>	101
4.3.12	<i>decoder/gethdr.c</i>	104
4.3.13	<i>decoder/getpic.c</i>	108
4.3.14	<i>decoder/getvlc.c</i>	113
4.3.15	<i>decoder/getvlc.h</i>	117
4.3.16	<i>decoder/global.h</i>	119
4.3.17	<i>decoder/idct.c</i>	121
4.3.18	<i>decoder/idctref.c</i>	122
4.3.19	<i>decoder/motion.c</i>	123
4.3.20	<i>decoder/mpeg2dec.c</i>	124
4.3.21	<i>decoder/mpeg2dec.h</i>	127
4.3.22	<i>decoder/recon.c</i>	127
4.3.23	<i>decoder/spatscal.c</i>	130
4.3.24	<i>decoder/store.c</i>	131
4.3.25	<i>decoder/subspic.c</i>	133
4.3.26	<i>decoder/systems.c</i>	135
4.3.27	<i>decoder/verify.c</i>	136
4.4	VIDEO PARAMETER FILES	138
4.4.1	<i>par/MPEG-1.par</i>	138
4.4.2	<i>par/NTSC.par</i>	138
4.4.3	<i>par/PAL.par</i>	138
4.4.4	<i>par/inter.mat</i>	138
4.4.5	<i>par/intra.mat</i>	138
5	AUDIO SIMULATION.....	140
5.1	VERIFICATION OF THE SIMULATION SOFTWARE.....	140
5.2	AUDIO DOCUMENTATION LISTINGS	143
5.2.1	<i>Readme.1st</i>	143
5.2.2	<i>doc/readme.1st</i>	143
5.2.3	<i>doc/readme.dp.01.txt</i>	144
5.2.4	<i>doc/readme.jmz.01.txt</i>	146
5.2.5	<i>doc/readme.jmz.02.txt</i>	147
5.2.6	<i>doc/readme.jmz.03.txt</i>	147
5.2.7	<i>doc/readme.jmz.04.txt</i>	148
5.2.8	<i>doc/readme.mc.01.txt</i>	148
5.2.9	<i>doc/readme.rb.01.txt</i>	149
5.2.10	<i>doc/readme.sn.01.txt</i>	149
5.2.11	<i>doc/readme.sr.01.txt</i>	149
5.2.12	<i>doc/readme.sr.02.txt</i>	149
5.2.13	<i>doc/readme.ss.01.txt</i>	150
5.3	AUDIO LOW SAMPLING FREQUENCIES ENCODER LISTINGS	151
5.3.1	<i>lsf/encoder/INSTALL</i>	151
5.3.2	<i>lsf/encoder/Makefile</i>	151
5.3.3	<i>lsf/encoder/Makefile.in</i>	151
5.3.4	<i>lsf/encoder/common.c</i>	151
5.3.5	<i>lsf/encoder/common.h</i>	158
5.3.6	<i>lsf/encoder/configure</i>	160
5.3.7	<i>lsf/encoder/encode.c</i>	163
5.3.8	<i>lsf/encoder/encoder.h</i>	169
5.3.9	<i>lsf/encoder/formatBitstream.c</i>	170
5.3.10	<i>lsf/encoder/formatBitstream.h</i>	172
5.3.11	<i>lsf/encoder/huffman.c</i>	173

5.3.12	lsf/encoder/huffman.h	175
5.3.13	lsf/encoder/ieeefloat.c	175
5.3.14	lsf/encoder/ieeefloat.h	179
5.3.15	lsf/encoder/l3bitstream-pvt.h	180
5.3.16	lsf/encoder/l3bitstream.c	180
5.3.17	lsf/encoder/l3bitstream.h	183
5.3.18	lsf/encoder/l3psy.c	184
5.3.19	lsf/encoder/l3psy.h	188
5.3.20	lsf/encoder/l3side.h	188
5.3.21	lsf/encoder/loop-pvt.h	189
5.3.22	lsf/encoder/loop.c	189
5.3.23	lsf/encoder/loop.h	199
5.3.24	lsf/encoder/makefile.unix	199
5.3.25	lsf/encoder/mdct.c	200
5.3.26	lsf/encoder/mdct.h	201
5.3.27	lsf/encoder/musicin.c	201
5.3.28	lsf/encoder/portableio.c	206
5.3.29	lsf/encoder/portableio.h	207
5.3.30	lsf/encoder/psy.c	208
5.3.31	lsf/encoder/reservoir.c	210
5.3.32	lsf/encoder/reservoir.h	211
5.3.33	lsf/encoder/subs.c	211
5.3.34	lsf/encoder/tonal.c	212
5.3.35	lsf/encoder/tables/1cb0	216
5.3.36	lsf/encoder/tables/1cb1	216
5.3.37	lsf/encoder/tables/1cb2	216
5.3.38	lsf/encoder/tables/1cb4	216
5.3.39	lsf/encoder/tables/1cb5	217
5.3.40	lsf/encoder/tables/1cb6	217
5.3.41	lsf/encoder/tables/1th0	217
5.3.42	lsf/encoder/tables/1th1	217
5.3.43	lsf/encoder/tables/1th2	218
5.3.44	lsf/encoder/tables/1th4	218
5.3.45	lsf/encoder/tables/1th5	219
5.3.46	lsf/encoder/tables/1th6	219
5.3.47	lsf/encoder/tables/2cb0	220
5.3.48	lsf/encoder/tables/2cb1	220
5.3.49	lsf/encoder/tables/2cb2	220
5.3.50	lsf/encoder/tables/2cb4	220
5.3.51	lsf/encoder/tables/2cb5	220
5.3.52	lsf/encoder/tables/2cb6	220
5.3.53	lsf/encoder/tables/2th0	220
5.3.54	lsf/encoder/tables/2th1	221
5.3.55	lsf/encoder/tables/2th2	221
5.3.56	lsf/encoder/tables/2th4	222
5.3.57	lsf/encoder/tables/2th5	222
5.3.58	lsf/encoder/tables/2th6	223
5.3.59	lsf/encoder/tables/absthr_0	224
5.3.60	lsf/encoder/tables/absthr_1	226
5.3.61	lsf/encoder/tables/absthr_2	228
5.3.62	lsf/encoder/tables/alloc_0	230
5.3.63	lsf/encoder/tables/alloc_1	231
5.3.64	lsf/encoder/tables/alloc_2	232
5.3.65	lsf/encoder/tables/alloc_3	233
5.3.66	lsf/encoder/tables/alloc_4	233
5.3.67	lsf/encoder/tables/dewindow	234
5.3.68	lsf/encoder/tables/enwindow	235

5.3.69	lsf/encoder/tables/huffcode.....	237
5.3.70	lsf/encoder/tables/huffdec.....	243
5.3.71	lsf/encoder/tables/psy_data.....	244
5.4	AUDIO LOW SAMPLING FREQUENCIES DECODER LISTINGS.....	250
5.4.1	lsf/decoder/INSTALL.....	250
5.4.2	lsf/decoder/Makefile.....	250
5.4.3	lsf/decoder/Makefile.in.....	250
5.4.4	lsf/decoder/common.c.....	251
5.4.5	lsf/decoder/common.h.....	257
5.4.6	lsf/decoder/configure.....	259
5.4.7	lsf/decoder/decode.c.....	262
5.4.8	lsf/decoder/decoder.h.....	270
5.4.9	lsf/decoder/huffman.c.....	271
5.4.10	lsf/decoder/huffman.h.....	273
5.4.11	lsf/decoder/ieeefloat.c.....	273
5.4.12	lsf/decoder/ieeefloat.h.....	277
5.4.13	lsf/decoder/makefile.linux.....	278
5.4.14	lsf/decoder/makefile.unix.....	278
5.4.15	lsf/decoder/musicout.c.....	278
5.4.16	lsf/decoder/portableio.c.....	281
5.4.17	lsf/decoder/portableio.h.....	282
5.4.18	lsf/decoder/tables/1cb0.....	283
5.4.19	lsf/decoder/tables/1cb1.....	283
5.4.20	lsf/decoder/tables/1cb2.....	283
5.4.21	lsf/decoder/tables/1cb4.....	283
5.4.22	lsf/decoder/tables/1cb5.....	283
5.4.23	lsf/decoder/tables/1cb6.....	283
5.4.24	lsf/decoder/tables/1th0.....	284
5.4.25	lsf/decoder/tables/1th1.....	284
5.4.26	lsf/decoder/tables/1th2.....	284
5.4.27	lsf/decoder/tables/1th4.....	285
5.4.28	lsf/decoder/tables/1th5.....	285
5.4.29	lsf/decoder/tables/1th6.....	286
5.4.30	lsf/decoder/tables/2cb0.....	286
5.4.31	lsf/decoder/tables/2cb1.....	286
5.4.32	lsf/decoder/tables/2cb2.....	287
5.4.33	lsf/decoder/tables/2cb4.....	287
5.4.34	lsf/decoder/tables/2cb5.....	287
5.4.35	lsf/decoder/tables/2cb6.....	287
5.4.36	lsf/decoder/tables/2th0.....	287
5.4.37	lsf/decoder/tables/2th1.....	288
5.4.38	lsf/decoder/tables/2th2.....	288
5.4.39	lsf/decoder/tables/2th4.....	289
5.4.40	lsf/decoder/tables/2th5.....	289
5.4.41	lsf/decoder/tables/2th6.....	290
5.4.42	lsf/decoder/tables/absthr_0.....	290
5.4.43	lsf/decoder/tables/absthr_1.....	292
5.4.44	lsf/decoder/tables/absthr_2.....	295
5.4.45	lsf/decoder/tables/alloc_0.....	297
5.4.46	lsf/decoder/tables/alloc_1.....	298
5.4.47	lsf/decoder/tables/alloc_2.....	299
5.4.48	lsf/decoder/tables/alloc_3.....	299
5.4.49	lsf/decoder/tables/alloc_4.....	300
5.4.50	lsf/decoder/tables/dewindow.....	301
5.4.51	lsf/decoder/tables/enwindow.....	302
5.4.52	lsf/decoder/tables/huffdec.....	304
5.5	AUDIO MULTICHANNEL ENCODER LISTINGS.....	306

5.5.1	mc/encoder/common.c	306
5.5.2	mc/encoder/common.h	314
5.5.3	mc/encoder/dyn_cross.c	317
5.5.4	mc/encoder/encode.c	320
5.5.5	mc/encoder/encoder.h	340
5.5.6	mc/encoder/lfe.c	342
5.5.7	mc/encoder/lingual.c	343
5.5.8	mc/encoder/makefile	345
5.5.9	mc/encoder/musicin.c	345
5.5.10	mc/encoder/predisto.c	354
5.5.11	mc/encoder/psy.c	361
5.5.12	mc/encoder/subs.c	363
5.5.13	mc/encoder/tonal.c	364
5.5.14	mc/encoder/tables/1cb0	370
5.5.15	mc/encoder/tables/1cb1	370
5.5.16	mc/encoder/tables/1cb2	370
5.5.17	mc/encoder/tables/1th0	370
5.5.18	mc/encoder/tables/1th1	370
5.5.19	mc/encoder/tables/1th2	371
5.5.20	mc/encoder/tables/2cb0	371
5.5.21	mc/encoder/tables/2cb1	371
5.5.22	mc/encoder/tables/2cb2	372
5.5.23	mc/encoder/tables/2th0	372
5.5.24	mc/encoder/tables/2th1	372
5.5.25	mc/encoder/tables/2th2	373
5.5.26	mc/encoder/tables/absthr_0	373
5.5.27	mc/encoder/tables/absthr_1	375
5.5.28	mc/encoder/tables/absthr_2	378
5.5.29	mc/encoder/tables/alloc_0	380
5.5.30	mc/encoder/tables/alloc_1	381
5.5.31	mc/encoder/tables/alloc_2	382
5.5.32	mc/encoder/tables/alloc_3	382
5.5.33	mc/encoder/tables/dewindow	383
5.5.34	mc/encoder/tables/enwindow	384
5.6	AUDIO MULTICHANNEL DECODER LISTINGS	386
5.6.1	mc/decoder/common.c	386
5.6.2	mc/decoder/common.h	396
5.6.3	mc/decoder/decode.c	398
5.6.4	mc/decoder/decoder.h	411
5.6.5	mc/decoder/makefile	412
5.6.6	mc/decoder/musicout.c	412
5.6.7	mc/decoder/tables/1cb0	417
5.6.8	mc/decoder/tables/1cb1	417
5.6.9	mc/decoder/tables/1cb2	417
5.6.10	mc/decoder/tables/1th0	417
5.6.11	mc/decoder/tables/1th1	418
5.6.12	mc/decoder/tables/1th2	418
5.6.13	mc/decoder/tables/2cb0	419
5.6.14	mc/decoder/tables/2cb1	419
5.6.15	mc/decoder/tables/2cb2	419
5.6.16	mc/decoder/tables/2th0	419
5.6.17	mc/decoder/tables/2th1	420
5.6.18	mc/decoder/tables/2th2	420
5.6.19	mc/decoder/tables/absthr_0	421
5.6.20	mc/decoder/tables/absthr_1	423
5.6.21	mc/decoder/tables/absthr_2	425
5.6.22	mc/decoder/tables/alloc_0	427

5.6.23	<i>mc/decoder/tables/alloc_1</i>	428
5.6.24	<i>mc/decoder/tables/alloc_2</i>	429
5.6.25	<i>mc/decoder/tables/alloc_3</i>	430
5.6.26	<i>mc/decoder/tables/alloc_4</i>	430
5.6.27	<i>mc/decoder/tables/dewindow</i>	431
5.7	AUDIO TOOL CODE LISTINGS	433
5.7.1	<i>tool/pcm2aiff/common.h</i>	433
5.7.2	<i>tool/pcm2aiff/decoder.h</i>	436
5.7.3	<i>tool/pcm2aiff/pcm2aiff.c</i>	437

ANNEXES

A	FLOPPY DISKETTES CONTAINING SOFTWARE	441
B	LIST OF PATENT HOLDERS.....	442
C	BIBLIOGRAPHY	444

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 13818-5, which is a Technical Report of type 3, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC TR 13818 consists of the following parts, under the general title *Information technology — Generic coding of moving pictures and associated audio information*:

- *Part 1: Systems*
- *Part 2: Video*
- *Part 3: Audio*
- *Part 4: Compliance testing*
- *Part 5: Software simulation*
- *Part 6: Extensions for DSM-CC*
- *Part 7: Advanced Audio Coding (AAC)*
- *Part 9: Extension for real time interface for systems decoders*
- *Part 10: Conformance extensions for DSM-CC*

Amendment 1 to this part of ISO/IEC 13818 will be published to make audio bitstreams upgraded to comply with ISO/IEC 13818-3:1997.

Introduction

This part of ISO/IEC 13818 was developed in response to the growing need for a generic coding method of moving pictures and of associated sound for various applications such as digital storage media, television broadcasting and communication. The use of this specification means that motion video can be manipulated as a form of computer data and can be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

Information technology — Generic coding of moving pictures and associated audio information —

Part 5:

Software simulation

1 General

1.1 Scope

This part of ISO/IEC 13818 provides a C language software simulation of an encoder and decoder for Part 1 (Systems), Part 2 (Video), and Part 3 (Audio) of ISO/IEC 13818.

1.2 References

ISO 639:1988, *Code for the representation of names of languages*.

ISO 8859-1:1987, *Information processing - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1*.

ISO/IEC 10918-1:1994, *Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines*. (See also ITU-T Rec. T.81.)

ISO/IEC 11172-1:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 1: Systems*.

ISO/IEC 11172-2:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 2: Video*.

ISO/IEC 11172-3:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3: Audio*.

ISO/IEC 11172-4:1995, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 4: Conformance testing*.

ISO/IEC 11172-5:—¹⁾, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 5: Software simulation*.

ISO/IEC 13818-1:1996, *Information technology - Generic coding of moving pictures and associated audio information : Systems*. (see also ITU-T Rec. H.222.0)

ISO/IEC 13818-2:1996, *Information technology - Generic coding of moving pictures and associated audio information : Video*. (see also ITU-T Rec. H.262)

1) To be published.

ISO/IEC 13818-3:1995, *Information technology - Generic coding of moving pictures and associated audio information : Audio*.

ISO/IEC 13818-4:—¹⁾, *Information technology - Generic coding of moving pictures and associated audio information : Compliance testing*.

Recommendations and reports of the CCIR, 1990, XVIIth Plenary Assembly, Düsseldorf, 1990, Volume XI - Part 1, Broadcasting Service (Television) Rec. 601-2, *Encoding parameters of digital television for studios*.

CCIR Volume X and XI Part 3 Recommendation 648: *Recording of audio signals*.

CCIR Volume X and XI Part 3 Report 955-2: *Sound broadcasting by satellite for portable and mobile receivers, including Annex IV Summary description of advanced digital system II*.

IEEE Standard Specifications for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, IEEE Std 1180-1990, December 6, 1990.

IEC 461:1986, *Time and control code for video tape recordings*.

IEC 908:1987, *Compact disk digital audio system*.

ITU-T Recommendation H.261 (Formerly CCITT Recommendation H.261) "Codec for audiovisual services at $p \times 64$ kbit/s" Geneva, 1990.

2 Technical elements

2.1 Definitions

For the purposes of this part of ISO/IEC 13818, the following definitions apply.

- 2.1.1 16x8 prediction [video]:** A prediction mode similar to field-based prediction but where the predicted block size is 16x8 luminance samples.
- 2.1.2 AC coefficient [video]:** Any DCT coefficient for which the frequency in one or both dimensions is non-zero.
- 2.1.3 access unit [systems]:** A coded representation of a presentation unit. In the case of audio, an access unit is the coded representation of an audio frame.
In the case of video, an access unit includes all the coded data for a picture, and any stuffing that follows it, up to but not including the start of the next access unit. If a picture is not preceded by a `group_start_code` or a `sequence_header_code`, the access unit begins with the picture start code. If a picture is preceded by a `group_start_code` and/or a `sequence_header_code`, the access unit begins with the first byte of the first of these start codes. If it is the last picture preceding a `sequence_end_code` in the bitstream all bytes between the last byte of the coded picture and the `sequence_end_code` (including the `sequence_end_code`) belong to the access unit.
- 2.1.4 adaptive bit allocation [audio]:** The assignment of bits to subbands in a time and frequency varying fashion according to a psychoacoustic model.
- 2.1.5 adaptive multichannel prediction [audio]:** A method of multichannel data reduction exploiting statistical inter-channel dependencies.
- 2.1.6 adaptive noise allocation [audio]:** The assignment of coding noise to frequency bands in a time and frequency varying fashion according to a psychoacoustic model.
- 2.1.7 adaptive segmentation [audio]:** A subdivision of the digital representation of an audio signal in variable segments of time.
- 2.1.8 alias [audio]:** Mirrored signal component resulting from sub-Nyquist sampling.
- 2.1.9 analysis filterbank [audio]:** Filterbank in the encoder that transforms a broadband PCM audio signal into a set of subsampled subband samples.
- 2.1.10 ancillary data [audio]:** part of the bitstream that might be used for transmission of ancillary data.
- 2.1.11 audio access unit [audio]:** For Layers I and II, an audio access unit is defined as the smallest part of the encoded bitstream which can be decoded by itself, where decoded means "fully reconstructed sound". For Layer III, an audio access unit is part of the bitstream that is decodable with the use of previously acquired main information.
- 2.1.12 audio buffer [audio]:** A buffer in the system target decoder for storage of compressed audio data.

2.1.13 audio sequence [audio]: A non-interrupted series of audio frames (base frames plus optional extension frames) in which the following parameters are not changed:

- ID
- Layer
- Sampling Frequency

For Layer I and II, a decoder is not required to support a continuously variable bitrate (change in the bitrate index) of the base stream. Such a relaxation of requirements does not apply to the extension stream.

2.1.14 B-field picture [video]: A field structure B-Picture.

2.1.15 B-frame picture [video]: A frame structure B-Picture.

2.1.16 B-picture; bidirectionally predictive-coded picture [video]: A picture that is coded using motion compensated prediction from past and/or future reference fields or frames.

2.1.17 backward compatibility: A newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream produced according to the newer coding standard.

2.1.18 backward motion vector [video]: A motion vector that is used for motion compensation from a reference frame or reference field at a later time in display order.

2.1.19 backward prediction [video]: Prediction from the future reference frame (field).

2.1.20 Bark [audio]: Unit of critical band rate. The Bark scale is a non-linear mapping of the frequency scale over the audio range closely corresponding with the frequency selectivity of the human ear across the band.

2.1.21 base layer [video]: First, independently decodable layer of a scalable hierarchy.

2.1.22 big picture [video]: A coded picture that would cause VBV buffer underflow as defined in C.7 Annex C of ISO/IEC 13818-2. Big pictures can only occur in sequences where low_delay is equal to 1. "Skipped picture" is a term that is sometimes used to describe the same concept.

2.1.23 bitrate [audio]: The rate at which the compressed bitstream is delivered to the input of a decoder.

2.1.24 bitstream; stream: An ordered series of bits that forms the coded representation of the data.

2.1.25 bitstream verifier [video]: A process by which it is possible to test and verify that all the requirements specified in ISO/IEC 13818-2 are met by the bitstream.

2.1.26 block [video]: An 8-row by 8-column matrix of samples, or 64 DCT coefficients (source, quantised or dequantised).

2.1.27 block companding [audio]: Normalising of the digital representation of an audio signal within a certain time period.

2.1.28 bottom field [video]: One of two fields that comprise a frame. Each line of a bottom field is spatially located immediately below the corresponding line of the top field.

2.1.29 bound [audio]: The lowest subband in which intensity stereo coding is used.

2.1.30 byte aligned: A bit in a coded bitstream is byte-aligned if its position is a multiple of 8-bits from the first bit in the stream.

2.1.31 byte: Sequence of 8-bits.

2.1.32 centre channel [audio]: An audio presentation channel used to stabilise the central component of the frontal stereo image.

2.1.33 channel [audio]: A sequence of data representing an audio signal being transported.

2.1.34 chroma simulcast [video]: A type of scalability (which is a subset of SNR scalability) where the enhancement layer (s) contain only coded refinement data for the DC coefficients, and all the data for the AC coefficients, of the chrominance components.

2.1.35 chrominance format [video]: Defines the number of chrominance blocks in a macroblock.

- 2.1.36 chrominance component [video]:** A matrix, block or single sample representing one of the two colour difference signals related to the primary colours in the manner defined in the bitstream. The symbols used for the chrominance signals are Cr and Cb.
- 2.1.37 coded audio bitstream [audio]:** A coded representation of an audio signal as specified in part 3 of ISO/IEC 13818.
- 2.1.38 coded B-frame [video]:** A B-frame picture or a pair of B-field pictures.
- 2.1.39 coded frame [video]:** A coded frame is a coded I-frame, a coded P-frame or a coded B-frame.
- 2.1.40 coded I-frame [video]:** An I-frame picture or a pair of field pictures, where the first field picture is an I-picture and the second field picture is an I-picture or a P-picture.
- 2.1.41 coded order [video]:** The order in which the pictures are transmitted and decoded. This order is not necessarily the same as the display order.
- 2.1.42 coded P-frame [video]:** A P-frame picture or a pair of P-field pictures.
- 2.1.43 coded picture [video]:** A coded picture is made of a picture header, the optional extensions immediately following it, and the following picture data. A coded picture may be a coded frame or a coded field.
- 2.1.44 coded representation:** A data element as represented in its encoded form.
- 2.1.45 coded video bitstream [video]:** A coded representation of a series of one or more pictures as defined in ISO/IEC 13818-2.
- 2.1.46 coding parameters [video]:** The set of user-definable parameters that characterise a coded bitstream. Bitstreams are characterised by coding parameters. Decoders are characterised by the bitstreams that they are capable of decoding.
- 2.1.47 component [video]:** A matrix, block or single sample from one of the three matrices (luminance and two chrominance) that make up a picture.
- 2.1.48 compression:** Reduction in the number of bits used to represent an item of data.
- 2.1.49 constant bitrate:** Operation where the bitrate is constant from start to finish of the coded bitstream.
- 2.1.50 constrained parameters [video]:** The values of the set of coding parameters defined in 2.4.3.2 of ISO/IEC 11172-2.
- 2.1.51 constrained system parameter stream; CSPS [systems]:** A Program Stream for which the constraints defined in subclause 2.7.9 of ISO/IEC 13818-1 apply.
- 2.1.52 CRC:** The Cyclic Redundancy Check to verify the correctness of data.
- 2.1.53 critical band [audio]:** Psychoacoustic measure in the spectral domain which corresponds to the frequency selectivity of the human ear. This selectivity is expressed in Bark.
- 2.1.54 critical band rate [audio]:** Psychoacoustic function of frequency. At a given audible frequency, it is proportional to the number of critical bands below that frequency. The units of the critical band rate scale are Barks.
- 2.1.55 data element:** An item of data as represented before encoding and after decoding.
- 2.1.56 data partitioning [video]:** A method for dividing a bitstream into two separate bitstreams for error resilience purposes. The two bitstreams have to be recombined before decoding.
- 2.1.57 DC coefficient [video]:** The DCT coefficient for which the frequency is zero in both dimensions.
- 2.1.58 DCT coefficient [video]:** The amplitude of a specific cosine basis function.
- 2.1.59 de-emphasis [audio]:** Filtering applied to an audio signal after storage or transmission to undo a linear distortion due to emphasis.
- 2.1.60 decoded stream:** The decoded reconstruction of a compressed bitstream.
- 2.1.61 decoder input buffer [video]:** The first-in first-out (FIFO) buffer specified in the video buffering verifier.
- 2.1.62 decoder:** An embodiment of a decoding process.

- 2.1.63 decoder sub-loop [video]:** Stages within encoder which produce numerically identical results to the decode process described in ISO/IEC 13818-2 Chapter 7. Encoders capable of producing more than just I-pictures embed a decoder sub-loop to create temporal predictions and to model the behaviour of downstream decoders.
- 2.1.64 decoding (process):** The process defined in ISO/IEC 13818 parts 1, 2 and 3 that reads an input coded bitstream and outputs decoded pictures or audio samples.
- 2.1.65 decoding time-stamp; DTS [systems]:** A field that may be present in a PES packet header that indicates the time that an access unit is decoded in the system target decoder.
- 2.1.66 dequantisation:** The process of rescaling the quantised DCT coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse DCT.
- 2.1.67 digital storage media; DSM:** A digital storage or transmission device or system.
- 2.1.68 discrete cosine transform; DCT:** Either the forward discrete cosine transform or the inverse discrete cosine transform. The DCT is an invertible, discrete orthogonal transformation.
- 2.1.69 display aspect ratio [video]:** The ratio height/width (in SI units) of the intended display.
- 2.1.70 display order [video]:** The order in which the decoded pictures are displayed. Normally this is the same order in which they were presented at the input of the encoder.
- 2.1.71 display process [video]:** The (non-normative) process by which reconstructed frames are displayed.
- 2.1.72 downmix [audio]:** A matrixing of n channels to obtain less than n channels.
- 2.1.73 drift [video]:** Accumulation of mismatch between the reconstructed output produced by the hypothetical decoder sub-loop embedded within an encoder (see definition of "decoder sub-loop") and the reconstructed outputs produced by a (downstream) decoder.
- 2.1.74 DSM-CC:** digital storage media command and control.
- 2.1.75 dual channel mode [audio]:** A mode, where two audio channels with independent programme contents (e.g. bilingual) are encoded within one bitstream. The coding process is the same as for the stereo mode.
- 2.1.76 dual-prime prediction [video]:** A prediction mode in which two forward field-based predictions are averaged. The predicted block size is 16×16 luminance samples. Dual-prime prediction is only used in interlaced P-pictures.
- 2.1.77 dynamic crosstalk [audio]:** A method of multichannel data reduction in which stereo-irrelevant signal components are copied to another channel.
- 2.1.78 dynamic transmission channel switching [audio]:** A method of multichannel data reduction by allocating the most orthogonal signal components to the transmission channels.
- 2.1.79 editing:** The process by which one or more coded bitstreams are manipulated to produce a new coded bitstream. Conforming edited bitstreams must meet the requirements defined in parts 1, 2, and 3 of ISO/IEC 13818.
- 2.1.80 Elementary Stream Clock Reference; ESCR [systems]:** A time stamp in the PES Stream from which decoders of PES streams may derive timing.
- 2.1.81 elementary stream; ES [systems]:** A generic term for one of the coded video, coded audio or other coded bitstreams in PES packets. One elementary stream is carried in a sequence of PES packets with one and only one stream_id.
- 2.1.82 emphasis [audio]:** Filtering applied to an audio signal before storage or transmission to improve the signal-to-noise ratio at high frequencies.
- 2.1.83 encoder:** An embodiment of an encoding process.
- 2.1.84 encoding (process):** A process, not specified in ISO/IEC 13818, that reads a stream of input pictures or audio samples and produces a valid coded bitstream as defined in parts 1, 2, and 3 of ISO/IEC 13818.
- 2.1.85 enhancement layer [video]:** A relative reference to a layer (above the base layer) in a scalable hierarchy. For all forms of scalability, its decoding process can be described by reference to the lower layer decoding process and the appropriate additional decoding process for the enhancement layer itself.

- 2.1.86 entitlement control message; ECM [systems]:** Entitlement Control Messages are private conditional access information which specify control words and possibly other, typically stream-specific, scrambling and/or control parameters.
- 2.1.87 entitlement management message; EMM [systems]:** Entitlement Management Messages are private conditional access information which specify the authorisation levels or the services of specific decoders. They may be addressed to single decoders or groups of decoders.
- 2.1.88 entropy coding:** Variable length lossless coding of the digital representation of a signal to reduce redundancy.
- 2.1.89 event [systems]:** An event is defined as a collection of elementary streams with a common time base, an associated start time, and an associated end time.
- 2.1.90 evil bitstreams:** Bitstreams orthogonal to reality.
- 2.1.91 extension bitstream [audio]:** Information contained in an optional additional bit stream related to the audio base bit stream at the system level, to support bit rates beyond those defined in ISO/IEC 11172-3. The optional extension bit stream contains the remainder of the multichannel and multilingual data.
- 2.1.92 fast reverse playback [video]:** The process of displaying the picture sequence in the reverse of display order faster than real-time.
- 2.1.93 fast forward playback [video]:** The process of displaying a sequence, or parts of a sequence, of pictures in display-order faster than real-time.
- 2.1.94 FFT:** Fast Fourier Transformation. A fast algorithm for performing a discrete Fourier transform (an orthogonal transform).
- 2.1.95 field [video]:** For an interlaced video signal, a "field" is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.
- 2.1.96 field period [video]:** The reciprocal of twice the frame rate.
- 2.1.97 field picture; field structure picture [video]:** A field structure picture is a coded picture with picture_structure is equal to "Top field" or "Bottom field".
- 2.1.98 field-based prediction [video]:** A prediction mode using only one field of the reference frame. The predicted block size is 16x16 luminance samples. Field-based prediction is not used in progressive frames.
- 2.1.99 filterbank [audio]:** A set of band-pass filters covering the entire audio frequency range.
- 2.1.100 fixed segmentation [audio]:** A subdivision of the digital representation of an audio signal into fixed segments of time.
- 2.1.101 flag:** A variable which can take one of only the two values defined in this specification.
- 2.1.102 FLC:** Fixed Length Code.
- 2.1.103 forbidden:** The term "forbidden", when used in the clauses defining the coded bitstream, indicates that the value shall never be used. This is usually to avoid emulation of start codes.
- 2.1.104 forced updating [video]:** The process by which macroblocks are intra-coded from time-to-time to ensure that mismatch errors between the inverse DCT processes in encoders and decoders cannot build up excessively.
- 2.1.105 forward compatibility:** A newer coding standard is forward compatible with an older coding standard if decoders designed to operate with the newer coding standard are able to decode bitstreams of the older coding standard.
- 2.1.106 forward motion vector [video]:** A motion vector that is used for motion compensation from a reference frame or reference field at an earlier time in display order.
- 2.1.107 forward prediction [video]:** Prediction from the past reference frame (field).
- 2.1.108 frame [audio]:** A part of the audio bit stream that corresponds to audio PCM samples from an Audio Access Unit.
- 2.1.109 frame [video]:** A frame contains lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the

frame. For interlaced video a frame consists of two fields, a top field and a bottom field. One of these fields may be temporally located one field period later than the other.

2.1.110 frame period [video]: The reciprocal of the frame rate.

2.1.111 frame picture; frame structure picture [video]: A frame structure picture is a coded picture with picture_structure is equal to "Frame".

2.1.112 frame rate [video]: The rate at which frames are be output from the decoding process.

2.1.113 frame reordering [video]: The process of reordering the reconstructed frames when the coded order is different from the display order. Frame reordering occurs when B-frames are present in a bitstream. There is no frame reordering when decoding low delay bitstreams.

2.1.114 frame-based prediction [video]: A prediction mode using both fields of the reference frame.

2.1.115 free format [audio]: Any bitrate other than the defined bitrates that is less than the maximum valid bitrate for each layer.

2.1.116 future reference frame (field) [video]: A future reference frame(field) is a reference frame(field) that occurs at a later time than the current picture in display order.

2.1.117 granules [Layer II] [audio]: The set of 3 consecutive subband samples from all 32 subbands that are considered together before quantisation. They correspond to 96 PCM samples.

2.1.118 granules [Layer III] [audio]: 576 frequency lines that carry their own side information.

2.1.119 group of pictures [video]: A notion defined only in ISO/IEC 11172-2 (MPEG-1 Video). In ISO/IEC 13818-2, a similar functionality can be achieved by the mean of inserting group of pictures headers.

2.1.120 Hann window [audio]: A time function applied sample-by-sample to a block of audio samples before Fourier transformation.

2.1.121 header: A block of data in the coded bitstream containing the coded representation of a number of data elements pertaining to the coded data that follow the header in the bitstream.

2.1.122 Huffman coding: A specific method for entropy coding.

2.1.123 hybrid filterbank [audio]: A serial combination of subband filterbank and MDCT.

2.1.124 hybrid scalability [video]: Hybrid scalability is the combination of two (or more) types of scalability.

2.1.125 I-field picture [video]: A field structure I-Picture.

2.1.126 I-frame picture [video]: A frame structure I-Picture.

2.1.127 I-picture; intra-coded picture [video]: A picture coded using information only from itself.

2.1.128 IDCT: Inverse Discrete Cosine Transform.

2.1.129 IMDCT [audio]: Inverse Modified Discrete Cosine Transform.

2.1.130 intensity stereo [audio]: A method of exploiting stereo irrelevance or redundancy in stereophonic audio programmes based on retaining at high frequencies only the energy envelope of the right and left channels.

2.1.131 interlace [video]: The property of conventional television frames where alternating lines of the frame represent different instances in time. In an interlaced frame, one of the field is meant to be displayed first. This field is called the first field. The first field can be the top field or the bottom field of the frame.

2.1.132 intra coding [video]: Coding of a macroblock or picture that uses information only from that macroblock or picture.

2.1.133 ITU-T Rec. H.222.0 | ISO/IEC 13818 (multiplexed) stream [systems]: A bitstream composed of 0 or more elementary streams combined in the manner defined in ITU-T Rec. H.222.0 | ISO/IEC 13818-1.

2.1.134 joint stereo coding [audio]: Any method that exploits stereophonic irrelevance or stereophonic redundancy.

2.1.135 joint stereo mode [audio]: A mode of the audio coding algorithm using joint stereo coding.

2.1.136 layer [audio]: One of the levels in the coding hierarchy of the audio system defined in ISO/IEC 13818-3.

- 2.1.137 layer [systems]:** One of the levels in the data hierarchy of the video and system specifications defined in ISO/IEC 13818 parts 1 and 2.
- 2.1.138 layer [video]:** In a scalable hierarchy denotes one out of the ordered set of bitstreams and (the result of) its associated decoding process (implicitly including decoding of **all** layers below this layer).
- 2.1.139 layer bitstream [video]:** A single bitstream associated to a specific layer (always used in conjunction with layer qualifiers, e. g. "enhancement layer bitstream").
- 2.1.140 level [video]:** A defined set of constraints on the values which may be taken by the parameters of this specification within a particular profile. A profile may contain one or more levels. In a different context, level is the absolute value of a non-zero coefficient (see "run").
- 2.1.141 LFE [audio]:** Low Frequency Enhancement channel. A limited bandwidth channel for low frequency audio effects in a multichannel system.
- 2.1.142 low frequency enhancement channel [audio]:** A limited bandwidth channel for low frequency audio effects in a multichannel system.
- 2.1.143 lower layer [video]:** A relative reference to the layer immediately below a given enhancement layer (implicitly including decoding of **all** layers below this enhancement layer).
- 2.1.144 luminance component [video]:** A matrix, block or single sample representing a monochrome representation of the signal and related to the primary colours in the manner defined in the bitstream. The symbol used for luminance is Y.
- 2.1.145 macroblock [video]:** The four 8 by 8 blocks of luminance data and the two (for 4:2:0 chrominance format), four (for 4:2:2 chrominance format) or eight (for 4:4:4 chrominance format) corresponding 8 by 8 blocks of chrominance data coming from a 16 by 16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the sample data and sometimes to the coded representation of the sample values and other data elements defined in the macroblock header of the syntax defined in this part of this specification. The usage is clear from the context.
- 2.1.146 mapping [audio]:** Conversion of an audio signal from time to frequency domain by subband filtering and/or by MDCT.
- 2.1.147 masking [audio]:** A property of the human auditory system by which an audio signal cannot be perceived in the presence of another audio signal.
- 2.1.148 masking threshold [audio]:** A function in frequency and time below which an audio signal cannot be perceived by the human auditory system.
- 2.1.149 Mbit [video]:** 1 000 000 bits.
- 2.1.150 MCP [video]:** Motion Compensated Predictor.
- 2.1.151 MDCT [audio]:** Modified Discrete Cosine Transform which corresponds to the Time Domain Aliasing Cancellation Filter Bank.
- 2.1.152 mismatch [video]:** Numerical discrepancy between the data reconstructed from the same coded bitstream by two decoding processes. With the exception of IDCT, the specification of ISO/IEC 13818-2 defines the decoding process absolutely unambiguously. Therefore, if both decoding processes are implemented according the specifications ISO/IEC 13818-2, mismatch can only be caused by different implementations of IDCT.
- 2.1.153 motion compensation [video]:** The use of motion vectors to improve the efficiency of the prediction of sample values. The prediction uses motion vectors to provide offsets into the past and/or future reference frames or reference fields containing previously decoded sample values that are used to form the prediction error.
- 2.1.154 motion estimation [video]:** The process of estimating motion vectors during the encoding process.
- 2.1.155 motion vector [video]:** A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture or field to the coordinates in a reference frame or reference field.
- 2.1.156 MS stereo [audio]:** A method of exploiting stereo irrelevance or redundancy in stereophonic audio programmes based on coding the sum and difference signal instead of the left and right channels.
- 2.1.157 multichannel [audio]:** A combination of audio channels used to create a spatial sound field.

- 2.1.158 multilingual [audio]:** A presentation of dialogue in more than one language.
- 2.1.159 NIT [systems]:** Network Information Table as defined in table 2-23 of ISO/IEC 13818-1.
- 2.1.160 non-intra coding [video]:** Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.
- 2.1.161 non-tonal component [audio]:** A noise-like component of an audio signal.
- 2.1.162 Nyquist sampling:** Sampling at or above twice the maximum bandwidth of a signal.
- 2.1.163 opposite parity [video]:** The opposite parity of top is bottom, and vice versa.
- 2.1.164 P-field picture [video]:** A field structure P-Picture.
- 2.1.165 P-frame picture [video]:** A frame structure P-Picture.
- 2.1.166 P-picture; predictive-coded picture [video]:** A picture that is coded using motion compensated prediction from past reference fields or frame.
- 2.1.167 pack [systems]:** A pack consists of a pack header followed by zero or more packets. It is a layer in the system coding syntax described in 2.5.3.3 on page 51 of ISO/IEC 13818-1.
- 2.1.168 packet [systems]:** A packet consists of a header followed by a number of contiguous bytes from an elementary data stream. It is a layer in the system coding syntax described in 2.4.3 of ISO/IEC 13818-1.
- 2.1.169 packet data [systems]:** Contiguous bytes of data from an elementary stream present in a packet.
- 2.1.170 packet identifier; PID [systems]:** A unique integer value used to associate elementary streams of a program in a single or multi-program Transport Stream as described in 2.4.3 of ISO/IEC 13818-1.
- 2.1.171 padding [audio]:** A method to adjust the average length of an audio frame in time to the duration of the corresponding PCM samples, by conditionally adding a slot to the audio frame.
- 2.1.172 parameter:** A variable within the syntax of this specification which may take one of a range of values. A variable which can take one of only two values is a flag or indicator and not a parameter.
- 2.1.173 parity (of field) [video]:** The parity of a field can be top or bottom.
- 2.1.174 parser:** Functional stage of a decoder which extracts from a coded bitstream series of bits representing coded elements (FLC or VLC).
- 2.1.175 past reference frame (field) [video]:** A past reference frame(field) is a reference frame(field) that occurs at an earlier time than the current picture in display order.
- 2.1.176 PAT [systems]:** Program Association Table as defined in clause 2.4.4.3 of ISO/IEC 13818-1.
- 2.1.177 payload [systems]:** Payload refers to the bytes which follow the header bytes in a packet. For example, the payload of a Transport Stream packet includes the PES_packet_header and its PES_packet_data_bytes, or pointer_field and PSI sections, or private data; but a PES_packet_payload consists of only PES_packet_data_bytes. The Transport Stream packet header and adaptation fields are not payload.
- 2.1.178 PES [systems]:** An abbreviation for Packetized Elementary Stream.
- 2.1.179 PES packet [systems]:** The data structure used to carry elementary stream data. It consists of a PES packet header followed by PES packet payload and is described in 2.4.3.6 and 2.4.3.7 of ISO/IEC 13818-1.
- 2.1.180 PES packet header[systems]:** The leading fields in a PES packet up to and not including the PES_packet_data_byte fields, where the stream is not a padding stream. In the case of a padding stream the PES packet header is similarly defined as the leading fields in a PES packet up to and not including padding_byte fields.
- 2.1.181 PES Stream [systems]:** A PES Stream consists of PES packets, all of whose payloads consist of data from a single elementary stream, and all of which have the same stream_id. Specific semantic constraints apply.
- 2.1.182 picture [video]:** Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. A “coded picture” is defined in ISO/IEC 13818-2. For progressive video, a picture is identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.

- 2.1.183 picture data [video]:** In the VBV operations, picture data is defined as all the bits of the coded picture, all the header(s) and user data immediately preceding it if any (including any stuffing between them) and all the stuffing following it, up to (but not including) the next start code, except in the case where the next start code is an end of sequence code, in which case it is included in the picture data.
- 2.1.184 polyphase filterbank [audio]:** A set of equal bandwidth filters with special phase interrelationships, allowing for an efficient implementation of the filterbank.
- 2.1.185 prediction [audio]:** The use of a predictor to provide an estimate of the subband sample in one channel from the subband samples in other channels.
- 2.1.186 prediction error:** The difference between the actual value of a sample or data element and its predictor.
- 2.1.187 prediction:** The use of a predictor to provide an estimate of the sample value or data element currently being decoded.
- 2.1.188 predictor:** A linear combination of previously decoded sample values or data elements.
- 2.1.189 presentation channel [audio]:** audio channels at the output of the decoder corresponding to the loudspeaker positions left, centre, right, left surround and right surround.
- 2.1.190 presentation time-stamp; PTS [systems]:** A field that may be present in a PES packet header that indicates the time that a presentation unit is presented in the system target decoder.
- 2.1.191 presentation unit; PU [systems]:** A decoded Audio Access Unit or a decoded picture.
- 2.1.192 profile [video]:** A defined subset of the syntax of this specification.
- 2.1.193 profile-and-level combination [video]:** Point of conformance for video bitstreams and decoders. Defined profile-and-level combinations are defined in Chapter 8 of ISO/IEC 13818-2. In the case of a bitstream, the profile-and-level combination is derived from the `profile_and_level_indication`. A decoder may comply with several profile-and level combinations.
- 2.1.194 program [systems]:** A program is a collection of program elements. Program elements may be elementary streams. Program elements need not have any defined time base; those that do, have a common time base and are intended for synchronised presentation.
- 2.1.195 Program Clock Reference; PCR [systems]:** A time stamp in the Transport Stream from which decoder timing is derived.
- 2.1.196 program element[systems]:** A generic term for one of the elementary streams or other data streams that may be included in a program.
- 2.1.197 Program Specific Information; PSI [systems]:** PSI consists of normative data which is necessary for the demultiplexing of Transport Streams and the successful regeneration of programs and is described in 2.4.4 of ISO/IEC 13818-1. One case of PSI, the non-mandatory network information table, is privately defined.
- 2.1.198 progressive [video]:** The property of film frames where all the samples of the frame represent the same instances in time.
- 2.1.199 psychoacoustic model [audio]:** A mathematical model of the masking behaviour of the human auditory system.
- 2.1.200 quantisation matrix [video]:** A set of sixty-four 8-bit values used by the dequantiser.
- 2.1.201 quantised DCT coefficients [video]:** DCT coefficients before dequantisation. A variable length coded representation of quantised DCT coefficients is transmitted as part of the coded video bitstream.
- 2.1.202 quantiser scale [video]:** A scale factor coded in the bitstream and used by the decoding process to scale the dequantisation.
- 2.1.203 random access:** The process of beginning to read and decode the coded bitstream at an arbitrary point.
- 2.1.204 reconstructed frame [video]:** A reconstructed frame consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. A reconstructed frame is obtained by decoding a coded frame.

- 2.1.205 reconstructed picture [video]:** A reconstructed picture is obtained by decoding a coded picture. A reconstructed picture is either a reconstructed frame (when decoding a frame picture), or one field of a reconstructed frame (when decoding a field picture). If the coded picture is a field picture, then the reconstructed picture is the top field or the bottom field of the reconstructed frame.
- 2.1.206 reference decoder [video]:** A decoder that implements precisely the decoding process as specified in ISO/IEC 13818-2 and uses a reference IDCT. The reference decoder is capable of decoding compliant bitstreams of any defined profile-and-level.
- 2.1.207 reference IDCT [video]:** Embodiment of the saturated mathematical integer-number IDCT specified in Annex A of ISO/IEC 13818-2.
- 2.1.208 reference field [video]:** A reference field is one field of a reconstructed frame. Reference fields are used for forward and backward prediction when P-pictures and B-pictures are decoded. Note that when field P-pictures are decoded, prediction of the second field P-picture of a coded frame uses the first reconstructed field of the same coded frame as a reference field.
- 2.1.209 reference frame [video]:** A reference frame is a reconstructed frame that was coded in the form of a coded I-frame or a coded P-frame. Reference frames are used for forward and backward prediction when P-pictures and B-pictures are decoded.
- 2.1.210 reordering delay [video]:** A delay in the decoding process that is caused by frame reordering.
- 2.1.211 requantisation [audio]:** Decoding of coded subband samples in order to recover the original quantised values.
- 2.1.212 reserved:** The term "reserved" when used in the clauses defining the coded bitstream indicates that the value may be used in the future for ISO/IEC defined extensions.
- 2.1.213 run [video]:** The number of zero coefficients preceding a non-zero coefficient, in the scan order. The absolute value of the non-zero coefficient is called "level".
- 2.1.214 sample aspect ratio [video]:** (abbreviated to **SAR**). This specifies the distance between samples. It is defined (for the purposes of this specification) as the vertical displacement of the lines of luminance samples in a frame divided by the horizontal displacement of the luminance samples. Thus its units are (metres per line) ÷ (metres per sample).
- 2.1.215 saturation [video]:** Limiting a value that exceeds a defined range by setting its value to the maximum or minimum of the range as appropriate.
- 2.1.216 scalability [video]:** Scalability is the ability of a decoder to decode an ordered set of bitstreams to produce a reconstructed sequence. Moreover, useful video is output when subsets are decoded. The minimum subset that can thus be decoded is the first bitstream in the set which is called the base layer. Each of the other bitstreams in the set is called an enhancement layer. When addressing a specific enhancement layer, "lower layer" refer to the bitstream which precedes the enhancement layer.
- 2.1.217 scalable hierarchy [video]:** coded video data consisting of an ordered set of more than one video bitstream.
- 2.1.218 scalefactor [audio]:** Factor by which a set of values is scaled before quantisation.
- 2.1.219 scalefactor band [audio]:** A set of frequency lines in Layer III which are scaled by one scalefactor.
- 2.1.220 scalefactor index [audio]:** A numerical code for a scalefactor.
- 2.1.221 scrambling [systems]:** The alteration of the characteristics of a video, audio or coded data stream in order to prevent unauthorised reception of the information in a clear form. This alteration is a specified process under the control of a conditional access system.
- 2.1.222 side information:** Information in the bitstream necessary for controlling the decoder.
- 2.1.223 skipped macroblock [video]:** A macroblock for which no data is encoded.
- 2.1.224 slice [video]:** A series of consecutive macroblocks that start, in the coded bitstream, by a slice_start_code and that continue up to the next start code (or to the first stuffing bytes if the next start_code is preceded by stuffing bytes)..
- 2.1.225 slot [audio]:** A slot is an elementary part in the audio bit stream. In Layer I a slot equals four bytes, in Layers II and III one byte.

- 2.1.226 SNR scalability [video]:** A type of scalability where the enhancement layer (s) contain only coded refinement data for the DCT coefficients of the lower layer.
- 2.1.227 source stream:** A single non-multiplexed stream of samples before compression coding.
- 2.1.228 source; input [video]:** Term used to describe the video material or some of its attributes before encoding.
- 2.1.229 spatial prediction [video]:** prediction derived from a decoded frame of the lower layer decoder used in spatial scalability.
- 2.1.230 spatial scalability [video]:** A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer without using motion vectors. The layers can have different frame sizes, frame rates or chrominance formats.
- 2.1.231 splicing[systems]:** The concatenation, performed on the system level, of two different elementary streams. The resulting system stream conforms totally to ISO/IEC 13818-1. The splice may result in discontinuities in timebase, continuity counter, PSI, and decoding.
- 2.1.232 spreading function [audio]:** A function that describes the frequency spread of masking effects.
- 2.1.233 start codes [systems]:** 32-bit codes embedded in the coded bitstream that are unique. They are used for several purposes including identifying some of the layers in the coding syntax.
- 2.1.234 STD input buffer [systems]:** A first-in first-out buffer at the input of a system target decoder for storage of compressed data from elementary streams before decoding.
- 2.1.235 stereo mode [audio]:** Mode, where two audio channels which form a stereo pair (left and right) are encoded within one bitstream. The coding process is the same as for the dual channel mode.
- 2.1.236 stereo-irrelevant [audio]:** a portion of a stereophonic audio signal which does not contribute to spatial perception.
- 2.1.237 still picture [systems]:** A coded still picture consists of a video sequence containing exactly one coded picture which is intra-coded. This picture has an associated PTS and the presentation time of succeeding pictures, if any, is later than that of the still picture by at least two picture periods.
- 2.1.238 stuffing (bits); stuffing (bytes):** Code-words that may be inserted at particular locations in the coded bitstream that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream which would otherwise be lower than the desired bitrate.
- 2.1.239 subband [audio]:** Subdivision of the audio frequency band.
- 2.1.240 subband filterbank [audio]:** A set of band filters covering the entire audio frequency range. In ISO/IEC 13818-3, the subband filterbank is a polyphase filterbank.
- 2.1.241 subband samples [audio]:** The subband filterbank within the audio encoder creates a filtered and subsampled representation of the input audio samples. The filtered samples are called subband samples. From 32 time-consecutive input audio samples, one subband sample is generated within each of the 32 subbands.
- 2.1.242 surround channel [audio]:** An audio presentation channel added to the front channels (L and R or L, R, and C) to enhance the spatial perception.
- 2.1.243 synword [audio]:** A 12-bit code embedded in the audio bit stream that identifies the start of a base frame or an extension frame.
- 2.1.244 synthesis filterbank [audio]:** Filterbank in the decoder that reconstructs a PCM audio signal from subband samples.
- 2.1.245 System Clock Reference; SCR [systems]:** A time stamp in the Program Stream from which decoder timing is derived.
- 2.1.246 system header [systems]:** The system header is a data structure defined in subclause 2.5.3.5 of ISO/IEC 13818-1, that carries information summarising the system characteristics of the ITU-T Rec. H.222.0 | ISO/IEC 13818 multiplexed Program Stream.
- 2.1.247 system target decoder; STD [systems]:** A hypothetical reference model of a decoding process used to describe the semantics of an ITU-T Rec. H.222.0 | ISO/IEC 13818 multiplexed bitstream.

- 2.1.248 temporal prediction [video]:** prediction derived from reference frames or fields other than those defined as spatial prediction.
- 2.1.249 temporal scalability [video]:** A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer using motion vectors. The layers have identical frame size, and chrominance formats, but can have different frame rates.
- 2.1.250 time-stamp [systems]:** A term that indicates the time of a specific action such as the arrival of a byte or the presentation of a Presentation Unit.
- 2.1.251 tonal component [audio]:** A sinusoid-like component of an audio signal.
- 2.1.252 top field [video]:** One of two fields that comprise a frame. Each line of a top field is spatially located immediately above the corresponding line of the bottom field.
- 2.1.253 top layer [video]:** the topmost layer (with the highest layer_id) of a scalable hierarchy.
- 2.1.254 Transport Stream packet header [systems]:** The leading fields in a Transport Stream packet, up to and including the continuity_counter field.
- 2.1.255 triplet [audio]:** A set of 3 consecutive subband samples from one subband. A triplet from each of the 32 subbands forms a granule.
- 2.1.256 variable bitrate:** Operation where the bitrate varies with time during the decoding of a coded bitstream.
- 2.1.257 variable length coding:** A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.
- 2.1.258 VLC:** Variable Length Code.
- 2.1.259 VLD:** Variable Length Decoder.
- 2.1.260 video buffering verifier; VBV [video]:** A hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.
- 2.1.261 video sequence [video]:** The highest syntactic structure of coded video bitstreams. It contains a series of one or more coded frames.
- 2.1.262 xxx profile bitstream [video]:** a bitstream of a scalable hierarchy with a profile indication corresponding to xxx. Note that this bitstream is only decodable together with all its lower layer bitstreams (unless it is a base layer bitstream).
- 2.1.263 xxx profile decoder [video]:** decoder able to decode one or a scalable hierarchy of bitstreams of which the top layer conforms to the specifications of the xxx profile (with xxx being any of the defined Profile names).
- 2.1.264 xxx profile scalable hierarchy [video]:** set of bitstreams of which the top layer conforms to the specifications of the xxx profile.
- 2.1.265 zig-zag scanning order [video]:** A specific sequential ordering of the DCT coefficients from (approximately) the lowest spatial frequency to the highest.

2.2 Symbols and abbreviations

The ISO/IEC 9899:1990 standard for C and the ISO standard for C++ programming language specify the syntax used in this document. ISO C and C++ require adherence to IEEE standards which specify numerical accuracy and rounding behavior for mathematical operators.

3 Systems simulation

3.1 Software information

This is Release 1.0 of the MSYS Toolkit, a set of C++ classes for building and manipulating MPEG System streams. With these classes one build MPEG System multiplexors and demultiplexors and other MPEG System-related applications. Examples of an encoder (in encode.C) and a decoder (in decode.C) are provided.

3.2 Systems documentation files

3.2.1 Readme

GENERAL INFO 6/1/95

This is Release 1.0 of the MSYS Toolkit, a set of C++ classes for building and manipulating MPEG System streams. With these classes one build MPEG System multiplexors and demultiplexors and other MPEG System-related applications. Examples of an encoder (in encode.C) and a decoder (in decode.C) are provided.

A technical disclaimer: The purpose of this system is instructional - it was not written to be terribly efficient in terms of storage or speed (encode outputs a transport packet every 1-2 msec; decode handles a transport packet roughly every .5-1.0 msec on a Sparc 10).

TO BUILD THE TOOLKIT

First make sure that the GNU tools (specifically g++) are installed. Now type make. If you want to build other applications besides the four specified in the Makefile, then you will have to modify the Makefile.

TO RUN THE SAMPLES

To run the sample encoder type "encode <filename>". A number of transport packets will be constructed and placed in the file <filename>. To run the decoder type "decode <filename>" and the transport packets in the file <filename> will be decoded. The individual PES streams are written to files named "stream<n>" where n is the pid value if the elementary stream.

pesout will generate a sequence of PES packets.: type "pesout <filename>". To parse a sequence of PES packets (it is assumed that these are from the same elementary stream) type "pesin <filename>". Try this with the PES streams extracted when "decode" is run.

See the source code comments on each of the sample programs for more detailed descriptions of usage.

Read the file doc.text to learn about the library.

Read the documentation notes in each .H file to learn more about each class.

Experiment with the sample applications to learn how to build applications.

THINGS LEFT TO DO

There is still lots of additional work that can be done. For example:

1. The generation of indefinite length PES packets (i.e. PES packet with zero length fields). The decoder will correctly parse them.
2. Versioning. The insertion and deletion of elementary streams and programs is not complete.
3. Clocks and timestamps have to be reworked completely. Currently the Encoder maintains a single clock based on the transport stream output rate for stamping PCRs. Instead there should be a notion of a clock coming along with a selected elementary stream on a per-program basis. There should also be enough understanding of the elementary stream structure (e.g. picture start codes, frame rates and picture types) to build and timestamp meaningful PES packets.
4. STD buffer simulation.

There are also lots of little things that need fixing - you will find comments starting with the prefix "// FIX --" marking these locations in the code.

EXPERIENCE WITH ISO COMMITTEE BITSTREAMS

The decoder has been run on a number of the bitstreams exchanged by the ISO committee over the last two years. Although it would not be appropriate to release the results of those runs into the public domain, I would be happy to correspond with others who has access to the ISO bitstreams and either have decoded or expect to decode them.

Finally, write back in the happy event that you wish to contribute code to MSYS.

Enjoy. Guy Cherry

email: guyc@crl.labs.tek.com
US Mail: Guy Cherry,
Communication Systems Research Lab,
Tektronix, Inc.,
Box 500 MS 50-370
Beaverton OR 97077
Phone: 503-627-1123 (... 5, 8, 13, 21, ...)
Fax: 503-627-1736

3.2.2 doc.tex

The MSYS Toolkit
Release 1.0 - 6/1/95
Guy Cherry and Ravi Koruru
Video Systems Program
Tektronix, Inc.

1. Introduction

The ISO/IEC 13818-1 specification is an international standard that deals with combining/multiplexing one or more streams of video, audio and other data into single or multiple MPEG streams suitable for storage or transmission [1]. This document presents a C++ toolkit useful for building and experimenting with multiplexors, demultiplexors and other applications that obey this standard. Use of these classes is illustrated by building sample multiplexor and demultiplexor applications.

Within the standard's scope, multimedia streams can be multiplexed or combined either to produce Program Streams (PS) or Transport Streams

(TS). A program stream contains packets of variable length and is designed for relatively error-free environments and interactive multimedia applications. On the other hand, a transport stream contains fixed length packets (188 bytes) and is designed for environments where errors are more likely, such as storage or transmission in lossy or noisy media.

In order to parse and experiment with transport streams, we have built a C++ toolkit that provides a rich set of objects for developing MSYS applications such as transport stream multiplexors, demultiplexors, and analyzers. The library has been designed to be modular, easily reconfigurable and extensible. By modularity, we mean that changes requiring the modification of specific functionality are localized. By reconfigurability, we mean that MSYS objects can be wired together in many different ways. By extensibility, we mean that improving the toolkit's functionality is typically a matter of deriving one or more new objects from existing objects thus achieving code sharing and better integration with the rest of the toolkit.

The purpose of this document is to offer a high-level guided tour of the MSYS tool kit, present its current status and outline areas for further work. The rest of this document is organized as follows. The library design and the main classes of the library are described in Section 2. The use of these classes is illustrated by building an MSYS multiplexor section 3 and an MSYS demultiplexor Section 4. Finally, we suggest ways of extending and improving the library in Section 5.

To keep this document small, the reader is assumed have read the relevant portions of the ISO/IEC 13818-1 specification [1] that deal with transport streams. Other useful background information is available in [2], [3] and [4].

2. The MSYS Class Library

The main classes provided by the library are shown in Figure 1. Based on their functionality, the MSYS classes are segregated into five categories:

1. Those that encapsulate syntactic elements of a transport stream. Examples are the TS class, the PES class and the various PSI section classes (PATSection, MAPSection, etc).
2. Those that encapsulate devices. Examples are the InputPort class and OutputPort class.
3. Those that encapsulate processes. Processes have the property that they take input and produce certain output. A process can be simple or composite. PATProducer, PESProducer, TSProducer, etc. are examples of simple classes.
4. Those that help in overall management. Examples are the Directory and EventManager classes. "Events" are generated during the execution of the process classes. The EventManager class is responsible for handling the events in a flexible manner.
5. Those that encapsulate applications. Encoder and Decoder are examples of application classes.

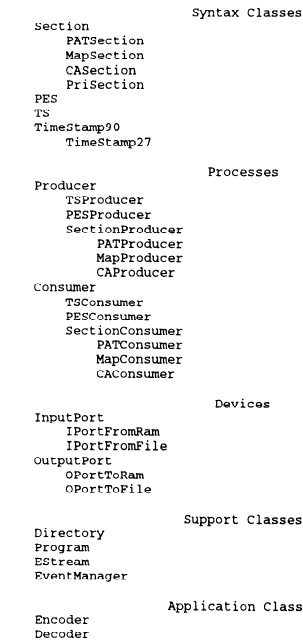


Figure 1. MSYS Toolkit Classes

Each of these categories are discussed in the following subsections.

2.1 Syntactic Classes

The rationale for syntactic classes is that these structures are the fundamental objects that make up a transport stream and defining them as individual classes provides a natural handle to manipulate them. Thus, each one of these entities, that is, elementary streams, program streams, transport packets, PES packets, PSI sections, time stamps, etc. are each modeled by a class.

These classes are typically used in an encoder as templates for specifying packet structure. For example a TSProducer manages a TS object which an application can manipulate to adjust the format of the packets being produced. Similarly, on the decoder side, a TSConsumer will manipulate its TS object to reflect the structure of the most recently received transport packet. With a reference to this object an application can monitor the structure of incoming packets.

The following are the MSYS syntactic objects:

1. The TS class embodies the TS header structure as defined in the ISO MPEG 2 standard along with methods to update and retrieve this structure. In other words, it contains information on transport_error_code, priority, scrambling_code, etc. Methods supported include methods that print the structure and two config functions that allow different fields of the class to be updated.
2. The PES class is essentially the PES header structure as defined in the ISO MPEG 2 standard along with methods to update and retrieve this structure. In other words, it contains information on PES packet length, scrambling code, priority, DTS, PTS, ES_CR, trick mode, etc. Methods supported include a print function and a variety of "config" functions that allow different fields of the class to be updated.
3. The TimeStamp90 and TimeStamp27 classes are structures with timestamp information. A few simple methods are supported for performing basic arithmetic.
4. The Section class, although there is no such syntactic unit in a transport stream, forms the base class for the different PSI sections. This subtyping was done since all PSI sections have many common fields and code duplication is reduced. A Section contains fields corresponding to the section pointer, table_id, section_syntax_indicator, section_length, version_number, current_next_indicator, section_number and the last_section_number fields in the PSI section.
5. The PAT, Map, CA and PRI Sections are subclasses of Section, adding syntax fields to complete the various sections in the MPEG System standard. For example, the PATSection provides the transport_stream_id. All the sections provide a print method.

2.2. Process Classes

Given that the syntactic classes are available, we need ways to generate and parse transport streams in a standard compliant manner. The Producer and Consumer classes are defined for this purpose and are introduced below.

1. **Producer.** A Producer class embodies the core properties needed for all types of producer in the context of MPEG transport stream. Simply put, each of the Producer objects is required to "produce" the specified number of bytes and "write" to an OutputPort. Depending on the producer type, the bytes written may be PSI information, PES header information, transport packet header information, timing information and/or bytes that belong to an elementary stream. A Producer is always created in the context of an Encoder object and provides two basic operations. The "connect" method takes an OutputPort as argument and provides the ability to connect a producer to any output port at any given time. The "send_partial" method instructs a producer to send the specified number of bytes (only part of a complete structure) to the connected output port. The send_partial method is called, for example, to fill transport packet payloads with parts of PES packets and PSI payloads. Before the send_partial method is called the "load" method must be called.

In order to promote code generality and reuse, the send_partial method is written in a subtype independent manner using a protected virtual function "fill_buffer" that is expected to be defined by each of the producer subclasses. The fill_buffer method is expected to fill the producer's internal buffer "buf" with bytes and return the length of the fill. Currently, some of the Producer subclasses do not define this method because the Encoder class is written such that the method is not required for these classes. If a subclass needs to define a fill_buffer method and does not define it, the base class fill_buffer method is triggered which prints an error message.

Producers can have producer states ("pstates"). States are queried, for example, by a transport packet producer (TSProducer) to determine when to terminate a PES packet and begin another.

2. **TSProducer.** The TSProducer class contains a TS object and defines a send_ts_packet method that uses this object as a template for building and sending TS packets. Currently the fill_buffer method is not implemented for TSProducer, disallowing calls to send_partial (thus, only complete transport packets can be produced).
3. **SectionProducer.** The SectionProducer class embodies the core properties and methods needed for all types of PSI sections in a MPEG transport stream. The main function it defines is fill_buffer which in turn is defined in terms of the abstract virtual function send_section that is implemented by each of the PSI subclasses. The send_section method uses the current output port object to "send" the bytes of the appropriate PSI section, including the CRC. However, since fill_buffer temporarily sets the current OutputPort object to a RAM object that begins at the address contained in buf, the invocation of send_section results in the bytes being assembled in the buffer. These bytes are later transferred to actual output port by the generic code contained in the send_partial method in the producer class.
4. **PAT, Map, CA Producers.** The main function the PAT, Map, and CA Producers implement is the send_section method. Each implementation of the method uses its corresponding section template in generating the output.
5. **PESProducer.** The main methods defined by a PESProducer are fill_buffer and send_pes_packet. The fill_buffer function is implemented in a similar manner to that of the SectionProducer class. The OutputPort object is temporarily set to memory before send_pes_packet is invoked. This results in storage of PES packet bytes beginning at the location pointed to by buf. These bytes are later transferred to actual output port by the generic code contained in the send_partial method in the producer class.
6. **Consumer.** The Consumer class embodies the core properties exhibited by all types of syntactic unit consumers. Simply put, each consumer object reads the specified number of bytes from the currently connected input port. A consumer object is always created within the context of a Decoder object and provides three methods. The "connect" method connects the consumer to an InputPort. The "read_partial" method reads the specified number of bytes from the currently connected input port. Since different consumer subtypes have different ways of parsing the input bytes, the read_partial method is declared as a virtual function. Each of the subtypes is expected to define this function. The third method "get_cstate" returns the current state of a consumer and its use is discussed later in this section.

Discounting various errors that might occur, two conditions can occur in a Consumer object while reading bytes from the InputPort. If its syntactic unit is completely received, it triggers a parse-completion event. If the bytes received are not enough to complete the syntactic unit no action is taken. For example, a PES packet is typically large and spread over multiple transport packets. Thus a PES consumer may have to be invoked several times before a "PESHeaderParsed" is triggered.

7. **TSConsumer.** The TSConsumer provides a read_ts_packet method which when invoked, parses the transport and adaptation field if any, followed by invoking read_partial method of the consumer that is attached to the PID found in the transport header. Thus, the consumer that is called can be a PES consumer or one of the PSI section consumers (PAT, MAP or CA Consumer).
8. **SectionConsumer.** A SectionConsumer Class embodies the core properties and methods needed for the various PSI sections in a MPEG transport stream. Two main methods read_partial and read_section can be invoked on an object of this class. The class defines the read_partial method which is shared by all its subclasses. The other method is the abstract virtual function read_section which is implemented by each of the PSI section consumers. The read_partial method uses the current InputPort object to read the bytes of the appropriate PSI section, including the CRC into its internal buffer "buf". Once a PSI section is completely assembled in buf, the current InputPort object is temporarily set to an InputFromRam object that is initialized to point to buf. Then the virtual function read_section is invoked that results in bytes being read from the InputFromRam object.
9. **PAT, Map, and CA Consumers.** The main method implemented by each of these classes is read_section that interprets the input bytes according to the corresponding PSI section. On completely reading an entire PSI section, all these classes trigger an appropriate PSI section-parsed event.
10. **PESConsumer.** On each invocation, the PESConsumer object reads the specified number of PES packet bytes and assembles them in its PES buffer, buf, until it recognizes the end of the PES packet. At this point, similar to the section consumer, a IPortFromRam object is created that is initialized to buf and is temporarily made the InputPort. A read_pes_packet function is invoked that reads bytes from the IPortFromRam object, parses the PES packet, sends the payload to the OutputPort, and triggers a PES parsed event.

2.3. Device Classes

The classes that can be classified as devices are the InputPort and OutputPort. The InputPort classes provide many methods including the ability to read bits and bytes from them. Similarly, the OutputPort Classes support methods to write bits and bytes to them.

1. **InputPort.** The InputPort classes provide methods including the ability to read bits and bytes from InputPorts. It is possible to pipe the input bits through a CRC decoder by invoking the start_crc method and checking for CRC match by calling check_crc. This results in the generation of CRC for all the bits beginning at the bit at which start_crc is called until the point check_crc is called. Currently, two subclasses are defined for the InputPort class: IPortFromFile and IPortFromRam.
2. **IPortFromRam and IPortFromFile.** IPortFromRam is an in-memory version of an input port. An IPortFromRam object is created with a buffer address and length as arguments. Reading a bit or byte from the object results in the transfer of the next bit or byte from the buffer. An IPortFromFile is the file version of an input port. It takes a file name as argument at creation time.
3. **OutputPort.** The OutputPort classes provide methods including the ability to write bits and bytes to OutputPorts. It is possible to send the output bits through a CRC encoder by invoking the start_crc method.
4. **OPortToRam and OPortToFile.** OPortToRam is an in-memory version of an OutputPort. An OPortToRam object is created with a buffer address, length, and an outputmode flag as arguments. Simply put, writing a bit or byte from the object results in the transfer of the bit or byte to the buffer. An OPortToFile is the file version of an output port. It takes a file name as argument at creation time. If the outputmode flag is set, then the output is in ascii otherwise, the output is in binary.

2.4. Management Classes

The Directory, Program, EStream and EventManager classes provide for organizing and managing applications.

1. **Directory, Program and EStream.** A Directory is essentially a list of Programs which, in turn, are lists of EStreams. Directory has methods for adding, removing and querying Programs. Program has similar methods for managing its EStreams. We will examine some of the methods in the discussion of the sample multiplexor and demultiplexor code.
2. **EventManager.** The EventManager class provides a general callback framework with regards to a predefined set of events. Event "handlers" are registered with an EventManager object by invoking a "register" method, specifying the event of interest and the address of the event handler.

An event handler is called when the "trigger" method is invoked on the EventManager. This triggering results in the invocation of the corresponding event handler, if any. Currently at most one handler can be specified for each possible event type.

All event handlers have the same function prototype:

```
void Handler (EventType type, void* client_data, void* call_data);
```

The pointer client_data is specified by the application when the event is registered and can be used to pass useful data from the register location in the code to the event handler. The structure call_data is specified by the EventManager and varies from EventType to EventType. For example whenever a TSParsed event is triggered a pointer to the TS object is passed to the event handler.

2.5. Application Classes

The application classes provided are the Encoder and Decoder classes. The reasons for separating these from other classes are two-fold. First, they are made of several of the basic objects. Second, given the basic syntactic classes, processes, devices and management

classes, there is considerable leeway in defining them.

1. Encoder: A Encoder object defines a one-point access to a Directory, an OutputPort, a list of Producers for that transport stream, and certain characteristics of the transport stream. A Encoder object created with a Directory object and the required bandwidth as arguments. However this information can dynamically changed. The Encoder class methods are discussed further in the following sections.
2. Decoder: A Decoder object defines a one-point access to an EventManager, a Directory, an InputPort and a list of Consumers that can read from that InputPort. A Decoder object is created in the context of an EventManager object. The Decoder class methods are discussed further in the following sections.
3. mux: A Sample Multiplexor Program

In this section, we will show and walk through the C++ like code for the "mux", a working multiplexor developed using the MSYS toolkit. For simplicity, the programs to be multiplexed are hard-coded into the mux program. mux has the following usage:

mux output_filename

Such a program can be built from the MSYS toolkit with the knowledge of EStream, Program, Directory, PES, TS and Encoder interfaces. An edited version of the code for the mux program is given below.

```

1 #include "Directory.H"
2 #include "Encoder.H"
3 #include "OPortToFile.H"
4 #include "PESProducer.H"
5 #include "TSProducer.H"
6
7 Encoder* encoder;
8 void config_mux ();
9 main (int argc, char* argv[])
10 {
11     // create a Directory and Programs
12     Directory* dir = new Directory();
13
14     Program* p1 = new Program(7);
15     ESTream* s1 = new ESTream(MPEG2VIDEO, UNASSIGNED, 10000);
16     p1->add_estream(s1);
17     ESTream* s2 = new ESTream(MPEG2AUDIO, UNASSIGNED, 512);
18     p1->add_estream(s2);
19     p1->activate();
20
21     Program* p2 = new Program(8);
22     ESTream* s3 = new ESTream(MPEG2VIDEO, UNASSIGNED, 10000);
23     p2->add_estream(s3); p2->activate();
24
25     dir->add_program(p1); dir->add_program(p2);
26
27     // create and update Encoder
28     Encoder* encoder = new Encoder(dir, 25000);
29     encoder->update();
30
31     // build oport and connect encoder
32     OPortToFile* oport = new OPortToFile(argv[1], 0);
33     encoder->connect(oport);
34
35     configure_mux ();
36
37     // send PSI followed by 10 "ordinary" packets
38     encoder->send_packet(0);
39     encoder->send_packet(p1->pid);
40     encoder->send_packet(p2->pid);
41     for (int i = 0; i < 10; i++) encoder->send_scheduled_packet(i);
42 }

```

The program has the following steps:

1. Create a Directory object with the hard-coded programs and constituent elementary streams (lines 12-25). After the creation of the Directory object, a Program object is created in line 14 with a PID of 7. In line 15, the EStream object s1 is created with type MPEG2VIDEO, a PID value of UNASSIGNED, which means that no particular value of PID is requested, and a required bandwidth of 10000 bits per second (bps). s1 is then added to the program p1 by calling the addEstream method. Similar actions occur for the Estream object s2. The activate method enables the program stream for multiplexing. In lines 21-23, Program p2 is created, an EStream s2 is created and added, and p2 is finally activated. Those two program streams then get added into the directory through the add_program method (line 25).
2. Create an Encoder object with a pointer to the directory and the total bandwidth available to this Encoder (lines 28,29).
3. Create an OutputPort object of type OutputPortToFile with the specified filename as parameter (line 32).
4. Invoke the Encoder to 'connect' to the OutputPort (line 33).
5. Configure the transport stream packet and PES packet producers for each of the programs. The templates for transport packets for the two program streams and the PES packets for the three elementary is initialized in this function.

```

1 void configure_mux()
2 {
3     // get TS and PES elements from Encoder and configure
4     TS* ts1 = encoder->get_ts(s1->pid);
5     PES* pes1 = encoder->get_pes(s1->pid);
6     PES* pes2 = encoder->get_pes(s2->pid);
7     TS* ts2 = encoder->get_ts(s3->pid);
8     PES* pes3 = encoder->get_pes(s3->pid);
9     PES* pes4 = encoder->get_pes(s4->pid);
10
11     ts1->config_basic('0', 'A', 'L', '0', 7, 'P');
12     ts1->config_adaptation('D', 'R', 'H', 222, NULL, 12, 24, 17, 16);
13     pes1->config_basic(int(id_video_stream_0 + 1),
14         214, '0', 'H', '0', 'C', '0',
15         688, 650, 601, 123345, NULL, NULL, 24);
16
17     pes2->config_basic(int(id_audio_stream_0 + 1),
18         101, '0', 'L', '0', 'C', '0',
19         788, 750, 701, 23345, NULL, NULL, 16);
20
21     ts2->config_basic('0', 'A', 'L', '0', 7, 'P');
22     ts2->config_adaptation('D', 'R', 'H', 1111, NULL, 18, 10, 19, 5);
23     pes3->config_basic(int(id_video_stream_0 + 2),

```

```

24         231, '0', 'H', '0', 'C', '0', 'Tek
25         888, '50', 'A01', 3345, NULL, NULL, 8);
26
27     pes4->config_basic(int(id_audio_stream_0 + 2),
28         231, '0', 'H', '0', 'C', '0',
29         6666, 7777, NULL, NULL, NULL, NULL, 11);
30
31     pes1->fileName = "streama";
32     pes2->fileName = "streamb";
33     pes3->fileName = "streamc";
34     pes4->fileName = "streamd";
35 }

```

6. Send transport packets containing PSI information followed by some transport packets that contain PES data (lines 38-40). Note that send_packet takes a PID as an argument while send_scheduled_packet takes none. The first form is used to send the PSI information. The PID needs to be sent over-ruling the default scheduling policy implemented within the multiplexor. The second form is used to send packets as determined by the scheduling policy. The currently implemented scheduling policy is the non-real-time one, so that the elementary streams in the PSI information, and the available bandwidth makes its decisions.

4. demux: A Sample Demultiplexor

Writing a demultiplexor is simpler than writing a multiplexor because minimal configuration setup is required. All demux has to do is to read from the InputPort and parse the input transport packets as they arrive. The demux program has the following usage:

demux input_filename

A simple demux can be built from the MSYS toolkit with the knowledge of interfaces to the EventManger, Directory, Decoder, and InputPort classes. An edited version of demux is given in below.

```

1 #include "Directory.H"
2 #include "Decoder.H"
3 #include "IPortFromFile.H"
4 #include "Events.H"
5 #include "Section.H"
6
7 // Globals
8 int PacketNumber = 0;
9
10 // Callbacks
11 void print_pes (EventType, void*, void*);
12 void print_map (EventType, void*, void*);
13 print_badcrc (EventType, void*, void*);
14
15 main (int argc, char* argv[])
16 {
17     // create an EventManager
18     EventManager* manager = new EventManager();
19
20     // create Directory
21     Directory* dir = new Directory();
22
23     // create a Decoder with a handles to manager
24     Decoder* decoder = new Decoder(manager);
25     decoder->install_dir(dir);
26
27     // register callbacks
28     manager->Register(MapParsed, print_map, decoder);
29     manager->Register(PESParsed, print_pes, NULL);
30     manager->Register(BadCRC, print_badcrc, NULL);
31
32     // build import and connect
33     IPortFromFile* iport = new IPortFromFile(argv[1]);
34     decoder->connect(iport);
35
36     // read some packets
37     for (; PacketNumber++) decoder->ReadPacket();
38 }

```

The following actions are done in the program:

1. Create an EventManager object and a Directory object.
2. Create a Decoder object with handles to the the EventManager and Directory.
3. Register handlers with the EventManager for the events of interest. In the program, handlers are functions that print some values to stdout. For example, print_pes is invoked by the EventManager when PESparsed event is generated by the PESConsumer object.
4. Create and connect an InputPort object to the Decoder. In this case, the argument to IPortFromFile object is the file name given as command line argument.
5. Read packets from the InputPort until end of transport stream is encountered.
5. Future Work

The MSYS tool kit development is on-going. Some of the known limitations and areas for further improvement are:

1. The toolkit cannot handle the cases when a section is longer than a single transport packet or the case where a table consists of multiple sections.
2. The toolkit cannot generate a PES packet of indefinite length (i.e. the case where the length field has a value of zero). The demultiplexer will correctly handle PES packets of indefinite length, however.
3. Clocks and timestamps have to be reworked completely. Currently the Encoder maintains a single clock based on the transport stream output rate for stamping PCRs. Instead there should be a notion of a clock coming along with a selected elementary stream on a per-program basis. There should also be enough understanding of the elementary stream structure (e.g. picture start codes, frame rates and picture types) to build and timestamp meaningful PES packets.
4. Buffer simulation is not done. Thus conditions such as buffer overflow and underflow in a demultiplexer cannot be tested.
4. A better GUI.

References

- [1] ISO/IEC JTC1/SC29/WG11 138818-1, MPEG 2 System IS, November 13 1994
- [2] Transport Layer Functional Description for the Grand Alliance ATV System, October 11, 1993.
- [3] "MPEG-2 - A Tutorial Introduction to the Systems Layer", P.A. Sarginson, (BBC R&D).
- [4] "The MPEG 2 System Tutorial", G.W. Cherry, trionix Inc.

3.3 Systems code listings

3.3.1 Consumer.C

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Consumer class implementation */

#include "Utilities.H"
#include "TSCConsumer.H"

Consumer::Consumer (Decoder* d)
{
    decoder = d;
}

void Consumer::connect (InputPort* p)
{
    iport = p;
}

void Consumer::flush ()
{
}

int Consumer::read_partial (int n)
{
    sys_message("unexpected virtual function call - Consumer::read_partial");
    for (int i = 0; i < n; i++)
        (void) iport->read_byte();
    return 1;
}

void Consumer::set_cstate (CState cs)
{
    cstate = cs;
    return;
}

CState Consumer::get_cstate ()
{
    return cstate;
}
```

3.3.2 Consumer.H

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Consumer class */
#ifdef consumer_h
#define consumer_h

#include "InputPort.H"

class Decoder;

typedef int CState;
```

```
#define CSTART 0

class Consumer
{
public:
    Consumer (Decoder*);
    void connect (InputPort*);
    virtual int read_partial (int);
    virtual void set_cstate (CState);
    virtual CState get_cstate ();
    virtual void flush ();
protected:
    decoder* decoder;
    InputPort* iport;
    CState cstate;
};

/*
 * DOCUMENTATION
 *
 * Consumers read things from InputPorts. Typical subclasses are
 * TSCConsumer, PESConsumer, PATConsumer, MapConsumer, etc. Although
 * some of these classes define methods for reading directly from the
 * InputPort and parsing immediately (e.g. TSCConsumer), Consumers can
 * also be set up to read bytes from the InputPort into a buffer, buf.
 * In this case parsing/analysis is deferred until the complete
 * syntactic unit (e.g. a PES packet or PAT Section) is in the buffer.
 *
 * As streams are read and analysed various events are triggered (e.g.
 * TSParsedEvent, UnexpectedPid, etc.)
 *
 * Public members are:
 *
 * Consumer (Decoder*);
 *     The only constructor. All Consumers have a Decoder object
 *     available. This is used to access or manipulate information
 *     related to the overall transport stream. The Decoder object is
 *     also used to access the EventManager for the generation of events.
 *
 * void connect (InputPort*);
 *     To connect to the InputPort.
 *
 * virtual void flush ();
 *     A virtual function defined by subclasses. PESConsumer, for
 *     example, flushes it's elementary stream port.
 *
 * virtual int read_partial (int);
 *     Will read a specified number of bytes from the InputPort.
 *     The specifics are different in each subclass.
 *
 * virtual void set_cstate (CState);
 *     These two functions get and set the Consumer's state. A Consumer
 *     may be in state CSTART or it may be in one of the states that are
 *     specific to one of the subclasses. This is set to CSTART, for
 *     example, by a TSCConsumer when a payload_unit_start_indicator is
 *     found to be '1'. In the subclasses it is managed and utilized in
 *     subsequent calls to read_partial.
 */
#endif
```

3.3.3 Decoder.C

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Decoder class implementation */

#include "Decoder.H"
#include "TSCConsumer.H"
#include "SectionConsumers.H"
#include "PESConsumer.H"

Decoder::Decoder (EventManager* m)
{
    manager = m;
    tscons = new TSCConsumer(this, new TS());
    net_pid = NULL;
    head_crc = NULL;
    packet_number = 0;
}

void Decoder::install_netpid (int id)
{
    // if id is a new net_pid, then destroy old consumer and install new one
    if (id != net_pid)
    {
        net_pid = id;
        add_cons(net_pid, new PriConsumer(this, new PriSection()));
    }
    // else do nothing
}

void Decoder::install_dir (Directory* d)
{
    // switch to new directory
    dir = d;
}
```

```

// loop through all programs in dir and build new list
ConsumerRecord* new_head_crec = NULL;
ProgramRecord* prec = dir->head_prec;
while (prec)
{
    Program* prog = prec->program;
    new_head_crec = install_prog_on_new_list (prog, new_head_crec);
    prec = prec->next_prec;
}

// now save old head_crec and install new head_crec
ConsumerRecord* old_head_crec = head_crec;
head_crec = new_head_crec;
add_cons(0, new PATConsumer(this, new PATSection));
add_cons(1, new CAConsumer(this, new CAsSection));

// now look for things on old list to clean up
ConsumerRecord* crec = old_head_crec;
while (crec)
{
    delete crec;
    crec = crec->next_crec;
}

void Decoder::install_prog (Program* newprog)
{
    // find and remove existing program from dir
    Program* oldprog = dir->get_program(newprog->program_number);
    if (oldprog) dir->remove_program(oldprog);

    // add newprog to dir
    dir->add_program(newprog);

    // now install program on consumer list
    install_prog_on_current_list (newprog);
}

void Decoder::connect (InputPort* ip)
{
    iport = ip;
    // connect tsport
    tscons->connect(ip);
    // payload Consumers connect just before they are used
}

void Decoder::flush ()
{
    ConsumerRecord* crec = head_crec;
    while (crec)
    {
        crec->cons->flush();
        crec = crec->next_crec;
    }
}

void Decoder::read_packet ()
{
    tscons->read_ts_packet();
    packet_number++;
}

Directory* Decoder::get_dir ()
{
    return dir;
}

int Decoder::get_networkpid ()
{
    return net_pid;
}

EventManager* Decoder::get_manager ()
{
    return manager;
}

TS* Decoder::get_ts ()
{
    return tscons->ts;
}

TS* Decoder::get_ts (int pid)
{
    ConsumerRecord* crec = find_crec(pid);
    if (crec == NULL) return NULL;
    return crec->ts;
}

PES* Decoder::get_pes (int pid)
{
    ConsumerRecord* crec = find_crec(pid);
    PESConsumer* pescons = (PESConsumer*) crec->cons;
    return pescons->pes;
}

void Decoder::add_cons (int id, Consumer* cons)
{
    ConsumerRecord* new_crec = new ConsumerRecord(id, cons);
    new_crec->next_crec = head_crec;
    head_crec = new_crec;
}

Consumer* Decoder::get_cons (int id)
{
    // look down list
    ConsumerRecord* crec = find_crec(id);
    if (crec == NULL) return NULL;
    return crec->cons;
}

void Decoder::print_pids ()
{
    ConsumerRecord* crec = head_crec;
    while (crec)
    {
        printf("%d ", crec->pid);
        crec = crec->next_crec;
    }
}

void Decoder::install_prog_on_current_list (Program* prog)
{
    // put map table consumer on list if not there already
    ConsumerRecord* map_crec = find_crec(prog->pid);

```

```

if (!map_crec)
    // place new program map consumer on consumer list
    {
        map_crec = new ConsumerRecord(prog->pid,
            new MapConsumer(this,
                new MapSection(prog)));
        map_crec->next_crec = head_crec;
        head_crec = map_crec;
    }

// put children estreams on list if not there already
ESTreamRecord* erec = prog->head_erec;
while (erec)
{
    ESTream* estream = erec->estream;
    ConsumerRecord* crec = find_crec(estream->pid);
    if (!crec)
    {
        crec = new ConsumerRecord(estream->pid,
            new PESConsumer(this, new PES));
        crec->next_crec = head_crec;
        head_crec = crec;
    }
    erec = erec->next_erec;
}

ConsumerRecord* Decoder::install_prog_on_new_list (Program* prog,
    ConsumerRecord* head_of_list)
{
    /*
    Takes a program and a consumer list Checks to see if any
    consumers already exist on the current installed consumer list.
    These get moved over to the new list. Otherwise new consumers are
    created and placed on the new list
    */

    // put map table consumer on list if not there already
    ConsumerRecord* map_crec = find_and_remove_crec(prog->pid);
    if (!map_crec)
    {
        // place new program map consumer on consumer list
        {
            map_crec = new ConsumerRecord(prog->pid, new MapConsumer(this,
                new MapSection(prog)));
        }
        map_crec->next_crec = head_of_list;
        head_of_list = map_crec;

    // put children estreams on list if not there already
    ESTreamRecord* erec = prog->head_erec;
    while (erec)
    {
        ESTream* estream = erec->estream;
        ConsumerRecord* crec = find_and_remove_crec(estream->pid);
        if (!crec)
        {
            crec = new ConsumerRecord(estream->pid,
                new PESConsumer(this, new PES));
        }
        crec->next_crec = head_of_list;
        head_of_list = crec;
        erec = erec->next_erec;
    }

    return head_of_list;
}

ConsumerRecord* Decoder::find_crec (int id)
{
    ConsumerRecord* crec = head_crec;
    while (crec)
    {
        if (crec->pid == id)
        {
            return crec;
        }
        crec = crec->next_crec;
    }
    return NULL;
}

ConsumerRecord* Decoder::find_and_remove_crec (int id)
{
    ConsumerRecord* cur_crec = head_crec;
    ConsumerRecord* prev_crec;
    while (cur_crec)
    {
        if (cur_crec->pid == id)
        {
            if (cur_crec == head_crec)
            {
                head_crec = head_crec->next_crec;
            }
            else
            {
                prev_crec->next_crec = cur_crec->next_crec;
            }
            return cur_crec;
        }
        prev_crec = cur_crec;
        cur_crec = cur_crec->next_crec;
    }
    return NULL;
}

ConsumerRecord::ConsumerRecord (int i, Consumer* c)
{
    pid = i;
    ts = new TS();
    ts->pid = pid;
    cons = c;
    next_crec = NULL;
}

```

3.3.4 Decoder.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.

```

```

*
*   Disclaimer of Warranty
*
* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

/* Decoder class */
#ifndef decoder_h
#define decoder_h

#include "Consumer.H"
#include "TS.H"
#include "PES.H"
#include "InputPort.H"
#include "Directory.H"
#include "Events.H"

class TSConsumer;
class ConsumerRecord;

class Decoder
{
public:
    Decoder (EventManager*);

    // connect and read
    void connect (InputPort*);
    void read_packet ();
    void flush ();

    // access functions
    void install_netpid (int);
    void install_dir (Directory*);
    void install_prog (Program*);
    Directory* get_dir ();
    int get_networkpid ();
    EventManager* get_manager ();
    TS* get_ts ();
    TS* get_ts (int);
    PES* get_pes (int);
    Consumer* get_cons (int);

    // utilities
    int packet_number;
    void print_pids ();
private:
    EventManager* manager;
    Directory* dir;
    InputPort* lport;
    int net_pid;
    Consumer* pat_cons;
    Consumer* ca_cons;
    TSConsumer* tscons;
    ConsumerRecord* head_crc;
    void add_cons (int, Consumer*);
    void install_prog_on_current_list (Program*);
    ConsumerRecord* install_prog_on_new_list (Program*, ConsumerRecord*);
    ConsumerRecord* find_crc (int);
    ConsumerRecord* find_and_remove_crc (int);
};

class ConsumerRecord
{
public:
    ConsumerRecord (int, Consumer*);
    TS* ts;
    int pid;
    Consumer* cons;
    ConsumerRecord* next_crc;
};

/*
DOCUMENTATION

A Decoder is basically an encapsulation of the environment present
as a transport stream is read. The Decoder object manages such
global information as the Network Table pid value, the current packet
number and the number of programs and streams per program (as
represented in the Directory structure).

Internally a Decoder manages a lookup table associating each active
pid value with a Consumer and other pid-specific information. When
the pid value is read from a new packet the corresponding record is
accessed and installed. The associated Consumer is used to read the
payload of the transport packet. An associated TS object is
installed as the current TS object in the decoder. This object is
examined to determine, for example, whether the continuity counter
incremented correctly since the last packet with the same pid value.

The Decoder supports the public method:

    void read_packet ();
    Called to read a new transport packet.
*/
#endif

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
*
*   Usage Restrictions
*
* License is granted to copy, to use, and to make and to use derivative
* works for research and evaluation purposes only.
*
*   Disclaimer of Warranty
*

```

```

* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

/* Directory, Program and EStream implementations */

#include "Utilities.H"
#include "Directory.H"

extern "C"
{
#include <stdio.h>
}

/* Directory functions */

Directory::Directory ()
{
    head_prec = NULL;
}

void Directory::add_program (Program* p)
{
    ProgramRecord* new_prec = new ProgramRecord(p);
    new_prec->next_prec = head_prec;
    head_prec = new_prec;
}

void Directory::remove_program (Program* p)
{
    ProgramRecord* cur_prec = head_prec;
    ProgramRecord* prev_prec;
    while (cur_prec)
    {
        if (cur_prec->program == p)
        {
            if (cur_prec == head_prec)
            {
                head_prec = head_prec->next_prec;
            }
            else
            {
                prev_prec->next_prec = cur_prec->next_prec;
            }
            delete cur_prec;
            return;
        }
        prev_prec = cur_prec;
        cur_prec = cur_prec->next_prec;
    }
}

void Directory::remove_program (int pn)
{
    ProgramRecord* cur_prec = head_prec;
    ProgramRecord* prev_prec;
    while (cur_prec)
    {
        if (cur_prec->program->program_number == pn)
        {
            if (cur_prec == head_prec)
            {
                head_prec = head_prec->next_prec;
            }
            else
            {
                prev_prec->next_prec = cur_prec->next_prec;
            }
            delete cur_prec;
            return;
        }
        prev_prec = cur_prec;
        cur_prec = cur_prec->next_prec;
    }
}

Program* Directory::get_program (int pn)
{
    ProgramRecord* prec = head_prec;
    while (prec)
    {
        if (prec->program->program_number == pn)
        {
            return prec->program;
        }
        prec = prec->next_prec;
    }
    return NULL;
}

int Directory::get_num_programs ()
{
    ProgramRecord* prec = head_prec;
    int n = 0;
    while (prec)
    {
        n++;
        prec = prec->next_prec;
    }
    return n;
}

void Directory::print ()
{
    ProgramRecord* prec = head_prec;
    while (prec)
    {
        prec->program->print();
        prec = prec->next_prec;
    }
}

```

```

}

/* Program functions */

Program::Program ()
{
    program_number = UNASSIGNED;
    pid = UNASSIGNED;
    active = ACTIVE;
    head_erec = NULL;
}

Program::Program (int pn)
{
    program_number = pn;
    pid = UNASSIGNED;
    active = ACTIVE;
    head_erec = NULL;
}

Program::Program (int pn, int id)
{
    program_number = pn;
    pid = id;
    active = ACTIVE;
    head_erec = NULL;
}

void Program::activate ()
{
    active = ACTIVE;
}

void Program::deactivate ()
{
    active = NOTACTIVE;
}

void Program::add_estream (ESTream* es)
{
    EStreamRecord* new_erec = new EStreamRecord(es);
    new_erec->next_erec = head_erec;
    head_erec = new_erec;
}

void Program::set_pcr_estream (ESTream* es)
{
    pcr_estream = es;
}

void Program::remove_estream (ESTream* es)
{
    EStreamRecord* cur_erec = head_erec;
    EStreamRecord* prev_erec;
    while (cur_erec)
    {
        if (cur_erec->estream == es)
        {
            if (cur_erec == head_erec)
            {
                head_erec = head_erec->next_erec;
            }
            else
            {
                prev_erec->next_erec = cur_erec->next_erec;
            }
            delete cur_erec;
            return;
        }
        prev_erec = cur_erec;
        cur_erec = cur_erec->next_erec;
    }
}

ESTream* Program::get_estream (int id)
{
    EStreamRecord* erec = head_erec;
    while (erec)
    {
        if (erec->estream->pid == id) return erec->estream;
        erec = erec->next_erec;
    }

    return NULL;
}

int Program::get_num_streams ()
{
    EStreamRecord* erec = head_erec;
    int n = 0;

    while (erec)
    {
        n++;
        erec = erec->next_erec;
    }
    return n;
}

void Program::print ()
{
    printf(" Program %d:\n", program_number);
    printf("  pid: %d\n", pid);
    EStreamRecord* erec = head_erec;
    while (erec)
    {
        erec->estream->print();
        erec = erec->next_erec;
    }
}

ProgramRecord::ProgramRecord (Program* prog)
{
    program = prog;
    next_prec = NULL;
}

/* EStream functions */

ESTream::ESTream (StreamType type, int id, int b)
{
    stream_type = type;
    pid = id;
    bandwidth = b;
}

```

```

}

void EStream::print ()
{
    printf("  EStream of type: %d pid: %d\n", stream_type, pid);
}

EStreamRecord::EStreamRecord (ESTream* es)
{
    estream = es;
    next_erec = NULL;
}

```

3.3.6 Directory.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Directory, etc. classes */

#ifndef directory_h
#define directory_h

enum StreamType {MPEG1VIDEO = 1, MPEG2VIDEO,
                 MPEG1AUDIO, MPEG2AUDIO, MPEGPRIVATE};

enum ProgramState {ACTIVE, NOTACTIVE};

const int UNASSIGNED = -1;

class EStream;
class EStreamRecord;
class Program;
class ProgramRecord;
class Directory;

class Directory
{
public:
    Directory ();
    void add_program (Program*);
    void remove_program (Program*);
    void remove_program (int); // remove by program_number
    Program* get_program (int); // find by program_number
    int get_num_programs ();
    ProgramRecord* head_prec;
    void print ();
private:
};

class Program
{
public:
    Program ();
    Program (int);
    Program (int, int);
    int program_number;
    int pid;
    EStream* pcr_estream;
    int active;
    void activate ();
    void deactivate ();
    void set_pcr_estream (ESTream*);
    void add_estream (ESTream*);
    void remove_estream (ESTream*);
    void remove_estream (int);
    EStream* get_estream (int); // find by pid
    int get_num_streams ();
    EStreamRecord* head_erec;
    void print ();
private:
};

class ProgramRecord
{
public:
    ProgramRecord (Program*);
    Program* program;
    ProgramRecord* next_prec;
};

class EStream
{
public:
    EStream(StreamType, int, int); // type, pid, bandwidth
    StreamType stream_type;
    int bandwidth;
    int pid;
    void print ();
};

class EStreamRecord
{
public:
    EStreamRecord (ESTream*);
    EStream* estream;
    EStreamRecord* next_erec;
};

```

```

/*
DOCUMENTATION

Directory, Program and EStream are used together to manage a list of
Programs where each Program contains a list of elementary streams
(EStreams).

These classes are used by the Encoder and Decoder class as a database
of program content information. Encoding applications build a
Directory and install it in the Encoder. The Encoder then uses the
Directory to assemble the multiplex. Decoding applications build a
Directory which the Decoder will manipulate to reflect the stream
contents. The Directory can then be examined by the application.

Directories manage a list of Programs and allow retrieval, insertion
and deletion.

Programs manage a list of EStreams and allow retrieval, insertion,
and deletion. Programs also have program_numbers, pids (for the
MapSection), and pcr_estreams. They can be active or inactive. The
meaning of active or inactive depends on the user of the Directory.

EStreams have a StreamType, a bandwidth and a pid.
*/

#endif

```

3.3.7 Encoder.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Encoder class implementation */

#include "Encoder.H"
#include "SectionProducers.H"
#include "PESProducer.H"
#include "TSProducer.H"
#include "Utilities.H"

Encoder::Encoder (EventManager* m)
{
    manager = m;
    tsprod = new TSProducer(this);
    head_prec = NULL;
    packet_number = 0;
    sys_time = 0;
    sys_const = 20000000;
    // set producer list to null
    head_prec = NULL;
}

void Encoder::set_bitrate (int r)
{
    bitrate = r;
    packet_time = (1504 * 27000) / bitrate;
}

void Encoder::install_dir (Directory* d)
{
    dir = d;

    // build null packet TS and add producer
    TS* nullts = new TS();
    nullts->config_basic('0', '0', 'L', '0', '0', '0');
    nullts->pid = 8191;
    (void) add_prod(nullts, new Producer(this), NULLPROD, NULL);

    // build PAT packet TS and add producer
    TS* patts = new TS();
    patts->config_basic('0', 'A', 'H', '0', '0', 'P');
    patts->pid = 0;
    (void) add_prod(patts, new PATProducer(this, new PATSection()), PSI, NULL);

    // FIX - set up conditional access producer as well

    // now setup prec's based on new directory
    build_records();
}

void Encoder::build_records ()
{
    // loop through programs in directory
    ProgramRecord* prec = dir->head_prec;
    while (prec)
    {
        // check pid for MapTable
        if (prec->program->pid == UNASSIGNED)
        {
            prec->program->pid = new_pid();
            if (prec->program->active == ACTIVE)
            {
                TS* mapts = new TS();
                mapts->pid = prec->program->pid;
                mapts->config_basic('0', 'A', 'H', '0', '0', 'P');
                (void) add_prod(mapts,
                    new MapProducer(this, new MapSection(prec->program)),
                    PSI, NULL);
            }
        }
    }
}

```

```

    }
    else
    {
        // FIX - check to see if program is active before sending Map table
    }
    // loop through programs estreams check estreams
    EStreamRecord* erec = prec->program->head_erec;
    while (erec)
    {
        if (erec->estream->pid == UNASSIGNED)
        {
            erec->estream->pid = new_pid();
            if (prec->program->active == ACTIVE)
            {
                TS* ts = new TS();
                ts->pid = erec->estream->pid;
                ts->config_basic('0', 'A', 'H', '0', '0', 'P');
                ProducerRecord* prodrec = add_prod(ts,
                    new PESProducer(this, new PES()), ELEM,
                    erec->estream->bandwidth);
                if (arec->estream == prec->program->pcr_estream)
                {
                    prodrec->pcr_flag = 1;
                    prodrec->next_pcr = 0;
                }
            }
        }
        else
        {
            // FIX - further work on Encoder::build_records
        }
        erec = erec->next_erec;
    }
    prec = prec->next_prec;
}

void Encoder::connect (OutputPort* op)
{
    oport = op;
    // connect tsport
    tsprod->connect(op);
    // payload Producers connect just before they are used
}

void Encoder::send_packet (int pid)
{
    // get prec
    ProducerRecord* prec = find_prec(pid);
    send_packet(prec);
}

void Encoder::send_scheduled_packet ()
{
    ProducerRecord* min_prec = NULL;
    double min_cur_gap = 100.0;
    ProducerRecord* prec = head_prec;
    while (prec)
    {
        if (prec->cur_gap < min_cur_gap)
        {
            min_cur_gap = prec->cur_gap;
            min_prec = prec;
        }
        prec = prec->next_prec;
    }
    send_packet(min_prec);
}

void Encoder::send_packet (ProducerRecord* prec togo)
{
    if (prec_togo == NULL)
    {
        prec_togo = find_prec(8191);
        if (prec_togo == NULL)
        {
            sys_message("can't find pid 8191 in Encoder table");
            return;
        }
    }

    // set up environment
    packet_number++;
    if (packet_number > 1) sys_time = sys_time + packet_time;

    // prepare and bind ts structure
    prec_togo->ts->inc_cc();
    if ((prec_togo->pcr_flag == 1) && (sys_time > prec_togo->next_pcr))
    {
        prec_togo->ts->add_pcr(sys_time);
        prec_togo->next_pcr = sys_time + Timestamp27(270000);
    }
    else
    {
        prec_togo->ts->delete_pcr();
    }
    tsprod->ts = prec_togo->ts;

    // send packet
    tsprod->send_ts_packet(prec_togo->prod);

    // adjust gaps
    ProducerRecord* prec = head_prec;
    while (prec)
    {
        if (prec == prec_togo)
            prec->cur_gap = prec->cur_gap + prec->gap;
        else
            prec->cur_gap = prec->cur_gap - 1;
        prec = prec->next_prec;
    }
}

Directory* Encoder::get_dir ()
{
    return dir;
}

TS* Encoder::get_ts (int pid)
{
    ProducerRecord* prec = find_prec(pid);
    if (prec)
    {
        return prec->ts;
    }
}

```

```

    return NULL;
}

PES* Encoder::get_pes (int pid)
{
    ProducerRecord* prec = find_prec(pid);
    if (prec)
    {
        PESProducer* pesprod = (PESProducer*) prec->prod;
        return pesprod->pes;
    }
    return NULL;
}

ProducerRecord* Encoder::add_prod (TS* ts, Producer* prod, ProdType t, int b)
{
    ProducerRecord* prec = new ProducerRecord(ts, prod, t);
    prec->next_prec = head_prec;
    prec->stream_band = b;
    head_prec = prec;
    recalc_mux();
    return prec;
}

void Encoder::delete_prod (int pid)
{
    ProducerRecord* cur_prec = head_prec;
    ProducerRecord* prev_prec;
    while (cur_prec)
    {
        if (cur_prec->ts->pid == pid)
        {
            if (cur_prec == head_prec)
            {
                head_prec = head_prec->next_prec;
            }
            else
            {
                prev_prec->next_prec = cur_prec->next_prec;
            }
            delete cur_prec;
            return;
        }
        prev_prec = cur_prec;
        cur_prec = cur_prec->next_prec;
    }
    recalc_mux();
}

ProducerRecord* Encoder::find_prec (int pid)
{
    ProducerRecord* prec = head_prec;
    while (prec)
    {
        if (prec == NULL)
        {
            sys_error("unexpected NULL prec");
        }
        if (prec->ts->pid == pid)
        {
            return prec;
        }
        prec = prec->next_prec;
    }
    return NULL;
}

Producer* Encoder::get_prod (int pid)
{
    ProducerRecord* prec = find_prec(pid);
    if (prec == NULL) return NULL;
    return prec->prod;
}

int Encoder::new_pid ()
{
    static int counter = 16;
    return counter++;
}

int max (int a, int b)
{
    return (a < b) ? b : a;
}

int Encoder::recalc_mux ()
{
    // sum non-psi and non-null bandwidths
    int elem_total = 0;
    int npsi = 0;
    ProducerRecord* prec = head_prec;
    while (prec)
    {
        if (prec->type == ELEM) elem_total = elem_total + prec->stream_band;
        if (prec->type == PSI) npsi++;
        prec = prec->next_prec;
    }

    // calculate total psi and set psi bandwidths
    psi_rate = max(80, int(double(bitrate) / 300.0));
    prec = head_prec;
    while (prec)
    {
        if (prec->type == PSI) prec->stream_band = psi_rate / npsi;
        prec->gap = (1.0 / (double(prec->stream_band) / double(bitrate))) - 1.0;
        prec = prec->next_prec;
    }

    // calculate null bandwidth
    if (elem_total + psi_rate > bitrate)
    {
        sys_error("stream bandwidth's is greater than total available");
    }
    prec = find_prec(8191);
    prec->stream_band = bitrate - (elem_total + psi_rate);
    prec->gap = (1.0 / (double(prec->stream_band) / double(bitrate))) - 1.0;

    return 1;
}

ProducerRecord::ProducerRecord (TS* t1, Producer* p, ProdType t2)
{
    ts = t1;
    type = t2;
    prod = p;
    pcr_flag = 0;
    next_prec = NULL;
}

```

3.3.8 Encoder.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Encoder class */
#ifndef encoder_h
#define encoder_h

#include "Events.H"
#include "Producer.H"
#include "Directory.H"
#include "OutputPort.H"
#include "TS.H"

class TSProducer;
class ProducerRecord;
class TS;
class PES;

enum ProdType (ELEM, PSI, NULLPROD);

class Encoder
{
public:
    Encoder (EventManager*);

    // connect and send
    void connect (OutputPort*);
    void send_scheduled_packet ();
    void send_packet (int);

    // access functions
    void set_bitrate (int);
    void install_dir (Directory*);
    Directory* get_dir ();
    TS* get_ts (int);
    PES* get_pes (int);
    int packet_number;
    Timestamp27 sys_time;
    Timestamp27 sys_const;

    // other utilities
    int new_pid ();
private:
    EventManager* manager;
    Directory* dir;
    TSProducer* tsprod;
    OutputPort* oport;
    int bitrate; // bits per second in 1000's
    int psi_rate;
    Timestamp27 packet_time;
    ProducerRecord* head_prec;
    void send_packet (ProducerRecord*);
    ProducerRecord* add_prod (TS*, Producer*, ProdType, int);
    void delete_prod (int);
    Producer* get_prod (int);
    ProducerRecord* find_prec (int);
    void build_records ();
    int recalc_mux ();
};

class ProducerRecord
{
public:
    ProducerRecord (TS*, Producer*, ProdType);
    ProdType type;
    TS* ts;
    int pcr_flag;
    Timestamp27 next_pcr; // nonzero only for pcr_pids
    Producer* prod;
    int stream_band;
    double gap;
    double cur_gap;
    ProducerRecord* next_prec;
};

/*
DOCUMENTATION

An Encoder is basically an encapsulation of the environment present
during a transport stream multiplex operation. The Encoder object
manages such global information as the overall transport stream
bitrate, the OutputPort, the Network Table pid value and the number
of programs and number of streams per program (as represented in a
Directory structure).

Internally an Encoder manages a lookup table associating each
active pid value with a Producer and other pid-specific information.
The multiplex algorithm will calculate the next outgoing pid value by
examining this table and will then use the corresponding Producer to
produce the payload for that pid value. At this time a TS object
associated with the current pid value will also be installed as the

```

Encoder's current TS object.

The Encoder also maintains a 27MHz system clock. This clock is incremented each time a transport packet is produced (in constant increments depending on the overall bitrate) and can be queried by other modules (such as the PES packet generator when PTS and DTS are generated). The encoder increments a packet_number field.

The Encoder supports two packet-producing functions:

```
void send_scheduled_packet ();
    Will call the scheduling algorithm to determine the next pid value
    to be sent in the mux and send a packet with that pid value.

void send_packet (int n);
    Will not invoke the scheduling algorithm but will send a packet
    with pid value n.
*/

#endif
```

3.3.9 Events.C

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Event implementation */

#include "Events.H"

EventManager::EventManager ()
{
    for (int i = 0; i < (int) NumberEventTypes; i++)
    {
        Callbacks[i] = NULL;
    }
}

void EventManager::Register (EventType t, Callback f, void* d)
{
    CallbackRecord* crec = new CallbackRecord(t, f, d);
    Callbacks[t] = crec;
}

void EventManager::Trigger (EventType t, void* call_data)
{
    if ((t < 0) || (t >= NumberEventTypes)) return;

    CallbackRecord* crec = Callbacks[t];
    if (crec) crec->callback(crec->type, crec->client_data, call_data);
}

CallbackRecord::CallbackRecord (EventType t, Callback c, void* d)
{
    type = t;
    callback = c;
    client_data = d;
}

Wrapper::Wrapper (int ii)
{
    i = ii;
}

Wrapper::Wrapper (char cc)
{
    c = cc;
}
```

3.3.10 Events.H

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
```

```
*/

/* Events.H */
#ifndef events_h
#define events_h

extern "C"
{
    #include <stdio.h>
}

enum EventType
{
    TSParsed,           // TS*
    PESHeaderParsed,    // PES*
    PATParsed,          // PATSection*
    MapParsed,          // MapSection*
    CAParsed,           // CASection*
    BadCRC,             // Section*
    Discontinuity,      // TS*
    RandomAccess,       // TS*
    PCREvent,           // TS*
    PTSDTSEvent,        // PES*
    UnexpectedPid,      // Wrapper* with int as pid
    Termination,        // NULL
    NumberEventTypes
};

typedef void Callback (EventType, void*, void*);
```

```
class CallbackRecord;

class EventManager
{
public:
    EventManager ();
    void Register (EventType, Callback, void*);
    void Trigger (EventType, void*);
private:
    CallbackRecord* Callbacks[NumberEventTypes];
};
```

```
class CallbackRecord
{
public:
    EventType type;
    Callback* callback;
    void* client_data;
    CallbackRecord (EventType, Callback, void*);
};
```

```
class Wrapper
{
public:
    Wrapper(int);
    Wrapper(char);
    int i;
    char c;
};
```

/* DOCUMENTATION

Applications can make use of the event mechanism implemented in the EventManager class. There are various places in the MSYS library where events are "triggered". For example when a transport packet is parsed the event "TSParsed" is triggered. At this point if there is a callback "registered" with the EventManager for the TSParsed event, then that callback routine will be called.

Callbacks must all have the same type:

```
typedef void Callback (EventType, void*, void*);
```

where
EventType is one of the values listed above.

The first void* is called the "client_data". It is any pointer specified by the application at the time of registration.

The second void* is called "call_data". It is a pointer passed by the EventManager to an object that is specific to that event. For example, the TSParsed event will pass a TS object representing the parse to the callback. The type of call data associated with each EventType is listed above with the EventTypes.

Currently only one callback can be registered for each EventType. They are stored and indexed in a simple array along with their client_data.

A class Wrapper is provided for passing ints and chars as client_data and call_data.

Public methods are:

EventManager ();
The only constructor.

void Register (EventType, Callback, void*);
Used to register a callback with an EventType. The third argument is client_data.

void Trigger (EventType, void*);
Used to trigger an event. The second argument is call_data.

*/

#endif

3.3.11 IPortFromFile.C

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
```

```

* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

```

```

/* IPortFromFile implementation */

```

```

#include "IPortFromFile.H"
#include "Utilities.H"

```

```

extern "C"

```

```

{
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
int read (int, char*, int);
int write (int, char*, int);
int close (int);
}

```

```

IPortFromFile::IPortFromFile (char* fileName, int flag) : InputPort()
{
// opens file and sets pos
cycle_flag = flag;
file_name = fileName;
fd = open(file_name, O_RDONLY);
if (fd > 0)
{
pos = -1; // forces read on first call to read_bit
}
else
{
sys_message("unable to open file in IPortFromFile::IPortFromFile");
}
}

```

```

char IPortFromFile::input_bit ()
// returns next bit as a char or BADBIT for an error
{
int result = inc_bit ();
if (result) return bit2char();
return BADBIT;
}

```

```

char IPortFromFile::input_byte ()
{
int result;

if (flen == 0)
{
flen = read(fd, fbuf, CacheSize);
if (flen == 0)
{
if (cycle_flag == NOCYCLE) sys_halt("End of Input Stream");
close(fd);
fd = open(file_name, O_RDONLY);
if (fd > 0) flen = read(fd, fbuf, CacheSize);
if (flen == 0) sys_halt("Unable to Cycle on input file");
if (flen < 0) sys_error("error reading input file");
}
else if (flen < 0)
{
sys_error("error reading input file");
}
bufpos = 0;
}
}

```

```

if (pos <= 0)
{
byte = fbuf[bufpos++];
if (bufpos == flen) flen = 0;
}
else
{
sys_error("can't read_byte; not on byte boundary");
}
return byte;
}

```

```

int IPortFromFile::inc_bit ()
// increments pos or reads from file if necessary
{
int result;

if (pos <= 0)
{
(void) input_byte(); // it has already set 'byte'
pos = 7;
}
else
{
pos--;
}
return 1;
}

```

```

char IPortFromFile::bit2char ()
// returns ONE or ZERO depending on current bit,
{
return (byte & (1 << pos)) ? ONE : ZERO;
}

```

3.3.12 IPortFromFile.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
*
* Usage Restrictions
*
* License is granted to copy, to use, and to make and to use derivative
* works for research and evaluation purposes only.
*
* Disclaimer of Warranty
*
* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

```

```

* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

```

```

/* IPortFromFile is an InputPort that reads from a file */
#ifndef iportfromfile_h
#define iportfromfile_h

```

```

#include "InputPort.H"

```

```

// FIX -- CacheSize should be specified in the constructor call.
#define CacheSize 4096
#define CYCLE 1
#define NOCYCLE 2

```

```

class IPortFromFile : public InputPort
{
public:
IPortFromFile (char*, int);
protected:
virtual char input_bit ();
virtual char input_byte ();
private:
int cycle_flag;
char* file_name;
int fd;
char fbuf[CacheSize];
int flen;
int bufpos;
char byte;
int pos;
int inc_bit ();
char bit2char ();
};

```

```

/* DOCUMENTATION

```

IPortFromFile is a subclass of InputPort. It provides implementations for the two virtual functions input_bit and input_byte and, except for constructors, its public interface is inherited from the base class. The port can be used in a CYCLE or NOCYCLE modes. When in CYCLE mode the port will loop back to the beginning of the file when the EOF is encountered, thus providing a ongoing stream of bits. When in NOCYCLE mode the port will call sys_halt when the EOF is reached.

File reads are cached to improve performance. The current cache size is set in the code to 4096.

Public methods are:

IPortFromFile (char*, int);
The second (integer) parameter is the cycle flag (either CYCLE or NOCYCLE).

```

*/
#endif

```

3.3.13 IPortFromRam.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
*
* Usage Restrictions
*
* License is granted to copy, to use, and to make and to use derivative
* works for research and evaluation purposes only.
*
* Disclaimer of Warranty
*
* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

```

```

* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

```

```

/* IPortFromRam implementation */

```

```

#include "IPortFromRam.H"
#include "Utilities.H"

```

```

IPortFromRam::IPortFromRam (char* b, int length) : InputPort()
{
bits = b;
bits_ptr = 0;
bits_length = length;
pos = -1; // forces read on first call to read_bit
}

```

```

char IPortFromRam::input_bit ()
// returns next bit as a char or BADBIT for an error
{
int result = inc_bit ();
if (result) return bit2char();
return BADBIT;
}

```

```

char IPortFromRam::input_byte ()

```

```

{
    if (pos <= 0)
    {
        byte = bits[bits_ptr];
        bits_ptr++;
        if (bits_ptr >= bits_length) bits_ptr = 0;
    }
    else
    {
        sys_error("can't read_byte; not on byte boundary");
    }
    return byte;
}

int IPortFromRam::inc_bit ()
// increments pos or reads from buffer if necessary
{
    if (pos <= 0)
    {
        byte = bits[bits_ptr];
        bits_ptr++;
        if (bits_ptr >= bits_length) bits_ptr = 0;
        pos = 7;
    }
    else
    {
        pos--;
    }
    return 1;
}

char IPortFromRam::bit2char ()
// returns ONE or ZERO depending on current bit;
{
    return (byte & (1 << pos)) ? ONE : ZERO;
}

```

3.3.14 IPortFromRam.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* IPortFromRam is an InputPort that reads bits from a ram buffer */
#ifndef iptportfromram_h
#define iptportfromram_h

#include "InputPort.H"

class IPortFromRam : public InputPort
{
public:
    IPortFromRam (char*, int);
protected:
    virtual char input_bit ();
    virtual char input_byte ();
private:
    char* bits;
    int bits_length;
    int bits_ptr;
    char byte;
    int pos;
    int inc_bit ();
    char bit2char ();
};

/*
DOCUMENTATION

IPortFromRam is a subclass of InputPort. It provides implementations
for the two virtual functions input_bit and input_byte and, except for
constructors, its public interface is inherited from the base class.

When the port is constructed a buffer and buffer length is specified.
Data is pulled from the buffer as the port is read. When the end of
the buffer is reached subsequent reads loop back to the beginning.

Public methods are:

    IPortFromRam (char*, int);
        The first argument specifies a buffer. The second argument
        specifies its length.
*/
#endif

```

3.3.15 InputPort.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *

```

```

 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

```

```

/* InputPort implementation */

```

```

#include "InputPort.H"
#include "Utilities.H"

```

```

extern "C"
{
    #include <stdio.h>
    #include <fcntl.h>
}

```

```

InputPort::InputPort ()
{
    crc_flag = FALSE;
    poly = NULL;
}

```

```

char InputPort::read_bit ()
{
    char c = input_bit();
    if (crc_flag && poly) poly->push_bit(c);
    return c;
}

```

```

char InputPort::read_byte ()
{
    int i;
    int bit;
    char byte = input_byte ();
    if (crc_flag && poly) {
        for (i=0; i < 8; i++) {
            bit = ((0x80 >> i) & byte);
            if (bit) bit = ONE; else bit = ZERO;
            poly->push_bit( bit);
        }
    }
    return byte;
}

```

```

int InputPort::read_pattern (char* pattern)
{
    // doesn't scan yet
    int n = strlen(pattern);
    int i;
    char b[32];
    int result = TRUE;
    char str[128];

    i = 0;
    while (i < n)
    {
        b[i] = read_bit();
        if (b[i] != pattern[i])
        {
            result = FALSE;
        }
        i++;
    }

    if (result == FALSE)
    {
        b[i] = '\0';
        sprintf(str,
            "(*s) does not match pattern (%s) InputPort::read_pattern",
            b, pattern);
        sys_message(str);
    }

    return result;
}

```

```

int InputPort::read_markerbit ()
{
    char c = read_bit();
    if (c == ONE)
    {
        return ONE;
    }
    else
    {
        sys_message("markerbit missed in InputPort::read_markerbit");
        return BADBIT;
    }
}

```

```

char InputPort::read_reserved_bits (int nbits)
// returns last bit read
{
    return read_nbits(nbits);
}

```

```

char InputPort::read_nbits (int n)
// returns last bit read
{
    char b;

    for (int i = 0; i < n; i++)
    {
        b = read_bit();
        if (b == BADBIT)
        {
            sys_error("bad bit found\n");
        }
    }
    return b;
}

```

```

)

int InputPort::read_uimbsf(int nbits, char* message)
{
    int value = 0;
    int i = 0;
    char b;

    while (i < nbits)
    {
        b = read_bit();
        if (b == '1')
            value = 2 * value + 1;
        else if (b == '0')
            value = 2 * value;
        else
        {
            sys_message(message);
            return value;
        }
        i++;
    }
    return value;
}

int power (int b, int n)
{
    if (n < 0)
    {
        sys_message("negative exponent in power - returning 0\n");
        return 0;
    }
    int res;
    res = 1;
    while (n > 0)
    {
        res = b * res;
        n--;
    }
    return res;
}

int InputPort::read_tcimbsf(int nbits, char* message)
{
    if (nbits > 8)
    {
        sys_message("can't read 2's comp with more than 8 bits; reading unsigned");
        return read_uimbsf(nbits, message);
    }
    else
    {
        int pos = read_uimbsf(nbits, message);
        int pow = power(2, nbits);
        if (pos > pow / 2)
            return -(pow - pos);
        else
            return pos;
    }
}

TimeStamp90* InputPort::read_timestamp90 ()
{
    int bit32 = read_uimbsf(1, "reading bit32");
    int bits30_31 = read_uimbsf(2, "reading bits 30..31");
    read_markerbit();
    int bits15_29 = read_uimbsf(15, "reading bits 15..29");
    read_markerbit();
    int bits0_14 = read_uimbsf(15, "reading bits 0..14");
    read_markerbit();
    return new TimeStamp90((bits30_31 << 30) + (bits15_29 << 15) + bits0_14, bit32);
}

TimeStamp27* InputPort::read_timestamp27_pes_format ()
{
    int bit32 = read_uimbsf(1, "reading bit32");
    int bits30_31 = read_uimbsf(2, "reading bits 30..31");
    read_markerbit();
    int bits15_29 = read_uimbsf(15, "reading bits 15..29");
    read_markerbit();
    int bits0_14 = read_uimbsf(15, "reading bits 0..14");
    read_markerbit();
    int ext = read_uimbsf(9, "reading extension bits");
    read_markerbit();
    return new TimeStamp27((bits30_31 << 30) + (bits15_29 << 15) + bits0_14, bit32, ext);
}

TimeStamp27* InputPort::read_timestamp27_ts_format ()
{
    int bit32 = read_uimbsf(1, "reading bit32");
    int bits0_31 = read_uimbsf(32, "reading bits 0..31");
    read_reserved_bits(6);
    int ext = read_uimbsf(9, "reading extension bits");
    return new TimeStamp27(bits0_31, bit32, ext);
}

void InputPort::start_crc ()
{
    if (poly) delete poly;
    poly = new Poly();
    crc_flag = TRUE;
}

void InputPort::stop_crc ()
{
    crc_flag = FALSE;
}

int InputPort::check_crc ()
{
    // first turn off crc_flag
    stop_crc();
    char buf[32];
    for (int i = 31; i >= 0; i--)
    {
        buf[i] = read_bit();
    }
    return poly->poly_cmp(buf);
}

void InputPort::print_crc ()
{
}

```

```

if (!poly) return;
printf("poly: ");
poly->print_bits();
printf("\n");
printf("data: ");
for (int i = 0; i < 32; i++)
{
    char c = input_bit();
    puts(c, stdout);
}
printf("\n");
}

```

3.3.16 InputPort.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

```

```

/* InputPorts are used to receive bitstreams from other processes. */
#ifndef inputport_h
#define inputport_h

#include "TimeStamp.H"
#include "Poly.H"

#define ZERO '0'
#define ONE '1'
#define BADBIT '2'
#define TRUE 1
#define FALSE 0

```

```

class InputPort
{
public:
    InputPort ();
    char read_bit ();
    char read_byte ();
    int read_pattern (char*);
    int read_markerbit ();
    char read_reserved_bits (int);
    char read_nbits (int);
    int read_uimbsf (int, char*);
    int read_tcimbsf (int, char*);
    TimeStamp90* read_timestamp90 ();
    TimeStamp27* read_timestamp27_pes_format ();
    TimeStamp27* read_timestamp27_ts_format ();
    void start_crc ();
    int check_crc ();
    void print_crc ();
    void stop_crc ();
protected:
    virtual char input_bit () = 0;
    virtual char input_byte () = 0;
private:
    int crc_flag;
    Poly* poly;
};

/*
DOCUMENTATION

```

InputPorts provide a stream of data. This data can be extracted from the port in a variety of ways with calls to member functions such as read_byte, read_nbits, read_pattern, etc.

A CRC check can be started, and stopped and checked as described below.

Subclasses of InputPort provide various implementations. For example, IPortFromFile will read bytes from a disk file. IPortFromRam will read bytes from a ram buffer. These subclasses must implement two virtual functions: input_bit and input_byte, which are then used by the generic code implemented in this base class.

Bits returned from the port are represented as the chars '0' and '1' (only an actual bit is read from the port, of course, but internally bits are efficiently managed as chars).

Public methods are:

InputPort ();
The only constructor - never called directly by an application.

char read_bit ();

char read_byte ();
This function requires that the port be on a byte boundary.

int read_pattern (char*);
int read_markerbit ();
char read_reserved_bits (int);
char read_nbits (int);
int read_uimbsf (int, char*);
int read_tcimbsf (int, char*);
TimeStamp90* read_timestamp90 ();
TimeStamp27* read_timestamp27_pes_format ();
TimeStamp27* read_timestamp27_ts_format ();
These each read from the port and return a corresponding object.

```
void start_crc ();
void stop_crc ();
int check_crc ();
void print_crc ();
start_crc will initialize an internal degree 32 polynomial and
arrange so that subsequent calls to any of the read functions will
perform the appropriate bitshift operation on the polynomial.
stop_crc will stop the bitshift. check_crc will read the next 32
bits from the InputPort and check them against the polynomial.
*/

#endif
```

3.3.17 Makefile

```
INCLFLAGS = -I.
LDLFLAGS =
CC = g++
CFLAGS = -g -c $(INCLFLAGS)
#CFLAGS = -g -c -DCRC_DEBUG $(INCLFLAGS)
LIB = -lm

# General sources and objects
GSRC = TS.C PES.C TimeStamp.C Section.C Utilities.C
GOBJ = TS.o PES.o TimeStamp.o Section.o Utilities.o

# TS Encoder sources and objects
TESRC = Directory.C Encoder.C Producer.C PESProducer.C \
        TSProducer.C OPortToRam.C \
        OutputPort.C OPortToFile.C Poly.C \
        InputPort.C IPortFromFile.C \
        SectionProducers.C Events.C
TEOBJ = Directory.o Encoder.o Producer.o PESProducer.o \
        TSProducer.o OPortToRam.o \
        OutputPort.o OPortToFile.o Poly.o \
        InputPort.o IPortFromFile.o \
        SectionProducers.o Events.o

# TS Decoder sources and objects
TDSRC = Decoder.C Directory.C Events.C Consumer.C TSConsumer.C \
        InputPort.C IPortFromFile.C IPortFromRam.C Poly.C \
        OutputPort.C OPortToFile.C \
        PESConsumer.C SectionConsumers.C
TDOBJ = Decoder.o Directory.o Events.o Consumer.o TSConsumer.o \
        InputPort.o IPortFromFile.o IPortFromRam.o Poly.o \
        OutputPort.o OPortToFile.o \
        PESConsumer.o SectionConsumers.o

.SUFFIXES : .o .c .s .C

EXECUS = encode decode pesout pesin

all: $(EXECUS)

encode: encode.o $(TEOBJ) $(GOBJ)
$(CC) $(LDLFLAGS) -o encode encode.o $(TEOBJ) $(GOBJ) $(LIB)

decode: decode.o $(TDOBJ) $(GOBJ)
$(CC) $(LDLFLAGS) -o decode decode.o $(TDOBJ) $(GOBJ) $(LIB)

pesout: pesout.o $(TEOBJ) $(GOBJ)
$(CC) $(LDLFLAGS) -o pesout pesout.o $(TEOBJ) $(GOBJ) $(LIB)

pesin: pesin.o $(TDOBJ) $(GOBJ)
$(CC) $(LDLFLAGS) -o pesin pesin.o $(TDOBJ) $(GOBJ) $(LIB)

.C.o :
$(CC) $(CFLAGS) <

clean:
rm -f *.o core stream[0-9]* *.out *.bits

purge:
rm -f *.o core stream[0-9]* *.out *.bits *.ts $(EXECUS)
```

3.3.18 OPortToFile.C

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* OPortToFile implementation */

#include "OPortToFile.H"
#include "Utilities.H"

extern "C"
{
int read (int, char*, int);
int write (int, char*, int);
int close (int);
}
```

```
OPortToFile::OPortToFile (char* f, int tf) : OutputPort(tf)
{
fd = open(f, O_CREAT | O_RDWR | O_TRUNC, 0666);
assert(fd);
next_out = 0;
bufpos = 0;
}

OPortToFile::~OPortToFile()
{
write(fd, fbuf, bufpos);
close(fd);
}

void OPortToFile::output_bit (char b)
{
buf[next_out] = b;
next_out++;
if (next_out > 7)
{
output_byte();
next_out = 0;
}
}

void OPortToFile::output_byte (char b)
{
int result;
if (next_out != 0)
{
sys_error("not on byte boundary in output_byte");
}
fbuf[bufpos++] = b;
if (bufpos == CacheSize)
{
result = write(fd, fbuf, CacheSize);
bufpos = 0;
}
}

void OPortToFile::output_flush ()
{
int result = write(fd, fbuf, bufpos);
bufpos = 0;
}

char inline bit (char c)
{
return (c == '0') ? ((char) 0) : ((char) 1);
}

void OPortToFile::output_byte ()
{
int result;
for (int i = 0; i < 8 - next_out; i++) write_bit('0');

if (textflag)
result = write(fd, fbuf, 8);
else
{
char b =
bit(buf[0]) << 7 |
bit(buf[1]) << 6 |
bit(buf[2]) << 5 |
bit(buf[3]) << 4 |
bit(buf[4]) << 3 |
bit(buf[5]) << 2 |
bit(buf[6]) << 1 |
bit(buf[7]) << 0;
fbuf[bufpos++] = b;
if (bufpos == CacheSize)
{
write(fd, fbuf, CacheSize);
bufpos = 0;
}
}
}
```

3.3.19 OPortToFile.H

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* OPortToFile is an OutputPort that puts bits to a file */
#ifndef oporttofile_h
#define oporttofile_h

#include "OutputPort.H"

// FIX -- CacheSize should be a parameter to the OPortToFile constructor
#define CacheSize 4096

extern "C"
{
#include <stdio.h>
#include <fcntl.h>
}
```

```

class OPortToFile: public OutputPort
{
public:
    OPortToFile (char*, int);
    virtual ~OPortToFile();
protected:
    virtual void output_bit (char);
    virtual void output_byte (char);
    virtual void output_flush ();
private:
    char buf[8];
    int next_out;
    int fd;
    char fbuf[CacheSize];
    int flen;
    int bufpes;
    void output_byte ();
};

/*
DOCUMENTATION

OPortToFile is a subclass of OutputPort. It provides implementations
for the three virtual funtions output_bit, output_byte and
output_flush and, except for constructors, its public interface is
inherited from the base class.

File writes are cached to improve performance. The current cache size
is set in the code to 4096.

Public methods are:

OPortToFile (char*, int);
    The second (integer) parameter is the text_flag. When set to true
    the port will output char's '0' and '1'. This is useful for
    debugging.
*/
#endif

```

3.3.20 OPortToRam.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* OPortToRam implementation */

#include "Utilities.H"
#include "OPortToRam.H"

OPortToRam::OPortToRam (char* b, int length, int tf) : OutputPort(tf)
{
    bits = b;
    bits_ptr = 0;
    bits_length = length;
    next_out = 0;
}

void OPortToRam::output_bit (char b)
{
    buf[next_out] = b;
    next_out++;
    if (next_out > 7)
    {
        output_byte();
        next_out = 0;
    }
}

void OPortToRam::output_byte (char b)
{
    if (next_out != 0)
    {
        sys_error("not on byte boundary in output_byte");
    }
    bits[bits_ptr] = b;
    bits_ptr++;
}

void OPortToRam::output_flush ()
{
    if (next_out > 0) output_byte ();
}

inline char bit (char c)
{
    return (c == '0') ? ((char) 0) : ((char) 1);
}

void OPortToRam::output_byte ()
{
    for (int i = 0; i < 8 - next_out; i++) write_bit('0');

    if (textflag)
    {
        for (int i = 0; i < 8; i++)
        {
            bits[bits_ptr] = buf[i];
            bits_ptr++;
        }
    }
}

```

```

    }
}
else
{
    char b =
        bit(buf[0]) << 7 |
        bit(buf[1]) << 6 |
        bit(buf[2]) << 5 |
        bit(buf[3]) << 4 |
        bit(buf[4]) << 3 |
        bit(buf[5]) << 2 |
        bit(buf[6]) << 1 |
        bit(buf[7]) << 0;
    bits[bits_ptr] = b;
    bits_ptr++;
}
}

```

3.3.21 OPortToRam.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* OPortToRam is an OutputPort that puts bits to ram */
#ifndef oporttoram_h
#define oporttoram_h

#include "OutputPort.H"

class OPortToRam: public OutputPort
{
public:
    OPortToRam (char*, int, int);
protected:
    virtual void output_bit (char);
    virtual void output_byte (char);
    virtual void output_flush ();
private:
    char buf[8];
    int next_out;
    char* bits;
    int bits_length;
    int bits_ptr;
    void output_byte ();
};

/*
DOCUMENTATION

OPortToRam is a subclass of OutputPort. It provides implementations
for the three virtual funtions output_bit, output_byte and
output_flush and, except for constructors, its public interface is
inherited from the base class.

When the port is constructed a buffer and buffer length is specified.
Data is written to the buffer as the port is written to. When the end of
the buffer is reached subsequent reads loop back to the beginning.

Public methods are:

OPortToRam (char*, int, int);
    The first argument specifies a buffer. The second argument
    specifies its length. The third parameter is the text_flag. When
    set to true the port will output char's '0' and '1'. This is
    useful for debugging.
*/
#endif

```

3.3.22 OutputPort.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party

```

```

* patents.
*/

/* OutputPort implementation */

#include "Utilities.H"
#include "OutputPort.H"

extern "C"
{
#include <stdio.h>
}

OutputPort::OutputPort (int tf)
{
    textflag = tf;
    crc_flag = FALSE;
    poly = NULL;
}

void OutputPort::write_bit (char c)
{
    output_bit(c);
    if (crc_flag && poly) poly->push_bit(c);
}

void OutputPort::write_byte (char c)
{
    for (int i = 7; i >= 0; i--)
        output_bit((c & (1 << i)) ? '1' : '0');
    if (poly) sys_message("write_byte called with crc on");
}

void OutputPort::flush ()
{
    output_flush();
}

void OutputPort::write_pattern (char* pattern)
{
    int n = strlen(pattern);

    for (int i = 0; i < n; i++)
    {
        write_bit(pattern[i]);
    }
}

void OutputPort::write_uimbsf(int value, int nbits)
{
    char rep[nbits];
    int i;

    for (i = 0; i < nbits; i++)
    {
        rep[i] = (((int) value / 2) == value) ? '0' : '1';
        value = value / 2;
    }

    for (i = nbits-1; i >= 0; i--)
    {
        (rep[i] == '0') ? write_bit('0') : write_bit('1');
    }
}

void OutputPort::write_tcimbsf(int value, int nbits)
{
    if (value >= 0)
    {
        write_uimbsf(value, nbits);
        return;
    }
    else
    {
        int vcomp = - value;
        char rep[nbits];
        int i;

        // calculate 1's comp of base 2 rep for -value
        for (i = 0; i < nbits; i++)
        {
            rep[i] = (((int) vcomp / 2) == vcomp) ? '1' : '0';
            vcomp = vcomp / 2;
        }
        // add 1
        for (i = 0; i < nbits; i++)
        {
            if (rep[i] == '1')
            {
                rep[i] = '0';
            }
            else
            {
                rep[i] = '1';
                break;
            }
        }
        // output bits
        for (i = nbits-1; i >= 0; i--)
        {
            (rep[i] == '0') ? write_bit('0') : write_bit('1');
        }
    }
}

void OutputPort::write_markerbit ()
{
    write_bit('1');
}

void OutputPort::write_reserved_bits (int nbits)
{
    for (int i = 0; i < nbits; i++)
    {
        write_bit('1');
    }
}

void OutputPort::write_timestamp90 (TimeStamp90* ts)
{
    write_uimbsf(ts->bit32, 1);
    write_uimbsf(ts->bits0_31 >> 30, 2);
    write_markerbit();
}

```

```

    write_uimbsf(ts->bits0_31 >> 15, 15);
    write_markerbit();
    write_uimbsf(ts->bits0_31, 15);
    write_markerbit();
}

void OutputPort::write_timestamp27_pes_format (TimeStamp27* ts)
{
    write_uimbsf(ts->bit32, 1);
    write_uimbsf(ts->bits0_31 >> 30, 2);
    write_markerbit();
    write_uimbsf(ts->bits0_31 >> 15, 15);
    write_markerbit();
    write_uimbsf(ts->bits0_31, 15);
    write_markerbit();
    write_uimbsf(ts->ext, 9);
    write_markerbit();
}

void OutputPort::write_timestamp27_ts_format (TimeStamp27* ts)
{
    write_uimbsf(ts->bit32, 1);
    write_uimbsf(ts->bits0_31, 32);
    write_reserved_bits(6);
    write_uimbsf(ts->ext, 9);
}

void OutputPort::start_crc ()
{
    if (poly) delete poly;
    poly = new Poly();
    crc_flag = TRUE;
}

void OutputPort::stop_crc ()
{
    crc_flag = FALSE;
}

void OutputPort::write_crc ()
{
    // write the complement of the crc
    for (int i = 31; i >= 0; --i)
    {
        if (poly->shift_reg[i])
        {
            write_bit('0');
        }
        else
        {
            write_bit('1');
        }
    }
}

```

3.3.23 OutputPort.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
*
* Usage Restrictions
*
* License is granted to copy, to use, and to make and to use derivative
* works for research and evaluation purposes only.
*
* Disclaimer of Warranty
*
* These software programs are available to the user without any license
* fee or royalty on an 'as is' basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

/* OutputPorts are used to send bitstreams to other processes. */
#ifdef outputport_h
#define outputport_h

#include "TimeStamp.H"
#include "Poly.H"

#define ZERO '0'
#define ONE '1'
#define BADBIT '2'
#define TRUE 1
#define FALSE 0

class OutputPort
{
public:
    OutputPort (int);
    void write_bit (char);
    void write_byte (char);
    void flush ();
    void write_pattern (char*);
    void write_uimbsf (int, int);
    void write_tcimbsf (int, int);
    void write_markerbit ();
    void write_reserved_bits (int);
    void write_timestamp90 (TimeStamp90*);
    void write_timestamp27_pes_format (TimeStamp27*);
    void write_timestamp27_ts_format (TimeStamp27*);
    void start_crc ();
    void stop_crc ();
    void write_crc ();
protected:
    virtual void output_bit (char) = 0;
    virtual void output_byte (char) = 0;
    virtual void output_flush () = 0;
    int textflag;
private:
    int crc_flag;
    Poly* poly;
}

```

```

);

/*
DOCUMENTATION

OutputPorts provide a generic target for the output of data. This
data can be written to the port with calls to functions such as
write_byte, write_bits, write_pattern, etc.

A CRC check can be stated, and stopped and written to the port as
described below.

Subclasses of OutputPort provide various implementations. For
example, OPortToFile will write bytes from a disk file, OPortToRam
will write bytes to a ram buffer. These subclasses must implement
three virtual functions: output_bit, output_byte, and output_flush
which are then used by the generic code implemented in this base
class.

Bits specified to the port are represented as the chars '0' and '1'
(only an actual bit is written to the port, of course, but internally
bits are efficiently managed as chars).

Public methods are:

OutputPort (int);
    The only constructor - never called directly by an application.
    The argument is the text_flag. When set to true the port will output
    char's '0' and '1'. This is useful for debugging.

void write_bit (char);

void write_byte (char);
    This function requires that the port be on a byte boundary.

void flush ();
    The actions taken here depend on the subclass.

void write_pattern (char*);
void write_uimbsf (int, int);
void write_tcimbsf (int, int);
void write_markerbit ();
void write_reserved_bits (int);
void write_timestamp90 (TimeStamp90*);
void write_timestamp27_pes_format (TimeStamp27*);
void write_timestamp27_ts_format (TimeStamp27*);
    These each write a corresponding object to the port.

void start_crc ();
void stop_crc ();
void write_crc ();
    start_crc will initialize an internal degree 32 polynomial and
    arrange so that subsequent calls to any of the write functions will
    perform the appropriate bitshift operation on the polynomial.
    stop_crc will stop the bitshift. write_crc will output the
    polynomial as the next 32 bits.
*/

#endif

```

3.3.24 PES.C

```

/* Copyright (C) 1995. Tektronix Inc. All Rights Reserved
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* PES class, TrickMode class and PESExtension class implementations */

#include "PES.H"

extern "C"
{
#include <malloc.h>
#include <stdio.h>
#include <ctype.h>
}

PES::PES ()
{
    DTS = (TimeStamp90*) NULL;
    PTS = (TimeStamp90*) NULL;
    ESCR = (TimeStamp27*) NULL;
    ES_rate = (int) NULL;
    trick_mode = (TrickMode*) NULL;
    additional_copy_info = (char) NULL;
    previous_CRC = (int) NULL;
    extension = (PESExtension*) NULL;
    number_stuffing_bytes = 0;
}

PES::PES (PES* p)
{
    stream_id = p->stream_id;
    PES_packet_length = p->PES_packet_length;
    scrambling_code = p->scrambling_code;
    priority_code = p->priority_code;
    alignment_indicator_code = p->alignment_indicator_code;
    copyright_code = p->copyright_code;
    original_code = p->original_code;
    DTS = p->DTS;

    PTS = p->PTS;
    ESCR = p->ESCR;
    ES_rate = p->ES_rate;
    trick_mode = p->trick_mode;
    additional_copy_info = p->additional_copy_info;
    previous_CRC = p->previous_CRC;
    extension = p->extension;
    number_stuffing_bytes = p->number_stuffing_bytes;
    file_name = p->file_name;
}

int PES::get_length ()
{
    if (PES_packet_length == 0) return NOLENGTH;
    return PES_packet_length + 6;
}

int PES::get_header_data_length ()
{
    // start with known minimal length and increment
    int header_length = 6;

    // if private or padding then it is what it is
    if ((stream_id == id_private_stream_2) ||
        (stream_id == id_padding_stream))
        return header_length;

    header_length += 3;
    if (PTS && !DTS) header_length += 5;
    if (PTS && DTS) header_length += 10;
    if (ESCR) header_length += 6;
    if (ES_rate) header_length += 3;
    if (trick_mode) header_length += 1;
    if (additional_copy_info) header_length += 1;
    if (previous_CRC) header_length += 2;
    if (extension) header_length += extension->get_length();

    return header_length;
}

int PES::get_header_length ()
{
    return get_header_data_length() + number_stuffing_bytes;
}

void PES::config_basic (int id, int len, char s, char p, char a,
    char c, char o, TimeStamp90* pts, TimeStamp90* dts,
    TimeStamp27* escr, int rate,
    char ac, int pcr, int ns)
{
    stream_id = id;
    PES_packet_length = len;
    scrambling_code = s;
    priority_code = p;
    alignment_indicator_code = a;
    copyright_code = c;
    original_code = o;
    if (dts) DTS = dts; else DTS = NULL;
    if (pts) PTS = pts; else PTS = NULL;
    if (escr) ESCR = escr; else ESCR = NULL;
    if (rate) ES_rate = rate; else ES_rate = 0;
    additional_copy_info = ac;
    previous_CRC = pcr;
    number_stuffing_bytes = ns;
}

void PES::config_trick (char c, char f1, char i, char f2, int r)
{
    if (!trick_mode) trick_mode = new TrickMode();
    trick_mode->trick_mode_control_code = c;
    trick_mode->field_id_code = f1;
    trick_mode->intra_slice_refresh_code = i;
    trick_mode->frequency_truncation_code = f2;
    trick_mode->field_rep_control = r;
}

void PES::config_extension ()
{
    if (!extension) extension = new PESExtension();
    // FIX - add real extension fields
}

void PES::print ()
{
    printf("stream_id: %d\n", stream_id);
    printf("PES_packet_length: %d\n", PES_packet_length);
    printf("scrambling_code: %c\n", scrambling_code);
    printf("PES_priority: %c\n", priority_code);
    printf("data_alignment_indicator: %c\n", alignment_indicator_code);
    printf("copyright: %c\n", copyright_code);
    printf("copy_or_original: %c\n", original_code);
    printf("PES_header_length: %d\n", get_header_length());

    if (PTS)
        printf("PTS: bit32 = %d; bits0..31 = %d\n",
            PTS->bit32, PTS->bits0_31);
    if (DTS)
        printf("DTS: bit32 = %d; bits0..31 = %d\n",
            DTS->bit32, DTS->bits0_31);
    if (ESCR)
        printf("ESCR: b32 = %d; bits0..31 = %d; ext = %d\n",
            ESCR->bit32, ESCR->bits0_31, ESCR->ext);
    if (ES_rate)
        printf("ES_rate: %d\n", ES_rate);
    if (trick_mode)
        trick_mode->print();
    if (additional_copy_info)
    {
        printf("additional copy info: ");
        for (int i = 0; i < 7; i++)
            if (additional_copy_info & (1 << i)) printf("1"); else printf("0");
        printf("\n");
    }
    if (previous_CRC)
        printf("previous_CRC: %d\n", previous_CRC);
    if (extension)
        extension->print();
}

```

```

    if (number_stuffing_bytes)
        printf("stuffing bytes: %d\n", number_stuffing_bytes);

    int length = get_length();
    if (length == NOLENGTH)
        printf("pes payload of unknown length\n");
    else
        printf("pes payload of length %d\n",
            length - get_header_length());
}

void TrickMode::print ()
{
    switch (trick_mode_control_code)
    {
        case 'F':
            printf("trick_mode: Fast Forward (%c, %c, %c)\n",
                field_id_code,
                intra_slice_refresh_code,
                frequency_truncation_code);
            break;
        case 'S':
            printf("trick_mode: Slow Motion (%d)\n", field_rep_control);
            break;
        case 'Z':
            printf("trick_mode: Freeze Frame (%c)\n", field_id_code);
            break;
        case 'R':
            printf("trick_mode: Fast Reverse (%c, %c, %c)\n",
                field_id_code,
                intra_slice_refresh_code,
                frequency_truncation_code);
            break;
    }
}

PESEExtension::PESEExtension ()
{
    PES_private_data = NULL;
    pack_field_length = 0;
    pack_header = NULL;
    pp_seq_counter_code = '0';
    P_STD_buffer_code = '0';
    PES_extension_field_length = 0;
}

int PESEExtension::get_length ()
{
    int length = 1;
    if (PES_private_data) length += 16;
    length += pack_field_length;
    if (pp_seq_counter_code == 'P') length += 2;
    if (P_STD_buffer_code == 'B') length += 2;
    length += PES_extension_field_length;
    return length;
}

void PESEExtension::print ()
{
    if (PES_private_data)
        for (int i = 0; i < 16; i++)
            printf("%c", PES_private_data[i]);

    if (pack_field_length)
        printf("pack_header of length: %d\n", pack_field_length);

    if (pp_seq_counter_code == 'P')
    {
        printf("pp_seq_counter: %d\n", pp_seq_counter);
        printf("original_stuff_length: %d\n", original_stuff_length);
    }

    if (P_STD_buffer_code == 'B')
    {
        printf("P_STD_buffer_scale: %c\n", P_STD_buffer_scale);
        printf("P_STD_buffer_size: %d\n", P_STD_buffer_size);
    }

    if (PES_extension_field_length)
    {
        printf("PES_extension_field of length: %d\n",
            PES_extension_field_length);
    }
}

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* PES class, TrickMode class and PESEExtension class */
#ifdef pes_h
#define pes_h

#include "TimeStamp.H"
class TrickMode;
class PESEExtension;

// stream_id values
enum StreamId
{
    id_program_stream_map = 188,
    id_private_stream_1,
    id_padding_stream,
    id_private_stream_2,
    id_audio_stream_0 = 192,
    id_video_stream_0 = 224,
    id_ECM = 256,
    id_EMM,
    id_DSMCC,
    id_MHEG,
    id_private_0,
    id_program_stream_dir = 511
};

#define NOLENGTH -1

class PES
{
public:
    int stream_id; // see table above
    int PES_packet_length; // NULL or calculated
    char scrambling_code; // (S,0) - Scrambled
    char priority_code; // (H,L) - High or Low
    char alignment_indicator_code; // (A,0) - Aligned
    char copyright_code; // (C,0) - Copyrighted
    char original_code; // (O,C) - Original or Copy
    TimeStamp90* DTS; // may be NULL
    TimeStamp90* PTS; // may be NULL
    int ES_rate; // may be NULL
    int ES_27; // may be 0

    TrickMode* trick_mode; // may be NULL
    char additional_copy_info; // 7 bit copy info or NULL
    int previous_CRC; // 16 bit CRC or NULL
    PESEExtension* extension; // may be NULL
    int number_stuffing_bytes; // 0..32
    char* file_name; // for test files

    PES ();
    PES (PES*);
    int get_length (); // total: PES_packet_length + 6
    int get_header_length (); // includes stuffing bytes; includes
    flags
    int get_header_data_length (); // excludes stuffing bytes; includes flags

    void config_basic (int, int, char, char, char, char, char,
        TimeStamp90*, TimeStamp90*, TimeStamp27*,
        int, char, int, int);
    // StreamId, PES_packet_length, Scrambling, Priority, Alignment,
    // Copyright, Original, DTS, PTS, ESCR, ES_rate,
    // additional_copy_info, prev_CRC, num_stuff

    void config_trick (char, char, char, char, int);
    // trick_mode_control, field_id, intra_slice, freq_truc, field_rep

    void config_extension ();

    void print ();
};

class TrickMode
{
public:
    char trick_mode_control_code; // (F,S,Z,R) or NULL
    char field_id_code; // (1,2,B)
    char intra_slice_refresh_code; // (I,0)
    char frequency_truncation_code; // (D,3,6,A)
    int field_rep_control; // 0..31
    void print ();
};

/* TrickMode mnemonics
 * F: Fastmotion; S: Slowmotion; Z: Freeze; R: Fast Reverse
 * (1,2,B) means field 1, field 2 or both are displayed
 * (I,0) means only 1 frames are being sent or not
 * (D,3,6,A) denotes only DC coeffs, first 3, first 6 or all coeffs are sent
 */

class PESEExtension
{
public:
    char* PES_private_data; // 16 bytes or NULL
    int pack_field_length; // 0..255
    char* pack_header; // a pack_header or NULL
    char pp_seq_counter_code; // (P,0) - Program packet counter included
    int pp_seq_counter; // 0..127
    int original_stuff_length; // 0..127
    char P_STD_buffer_code; // (B,0) - Buffer parameters included
    char P_STD_buffer_scale; // (0,1)
    int P_STD_buffer_size; // 0..2*13-1
    int PES_extension_field_length; // 0..127

    PESEExtension ();
    int get_length (); // total
    void print();
};

/*
 * DOCUMENTATION
 *
 * PES and its associated classes, TrickMode and PESEExtension, are
 * examples of syntax classes; its objects encapsulate the structure of
 * a PES syntax. When generating packets (say in an encoder
 * application) PES objects are used as templates. Changing the
 * template will change the structure of the packets being generated.
 * When parsing packets (say in a decoder application) PES objects are
 * set to reflect the structure of the packet being parsed and can then
 * be examined by the application.
 *
 * Mnemonic chars are used to represent the various values of the fields (e.g.
 * the priority_code can be 'H' or 'L' for High or Low). These are
 * listed in the above comments.
 *
 * All the fields are exposed as public members for simplicity.
 */

Various configure methods are provided to easily set multiple fields.
*/

#endif

```

3.3.26 PESConsumer.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* PES class implementation */

#include "Utilities.H"
#include "PESConsumer.H"
#include "IPortFromRam.H"
#include "Decoder.H"
#include "OPortToFile.H"

extern "C"
{
#include <stdio.h>
#include <fcntl.h>
#include <malloc.h>
int write (int, char*, int);
}

PESConsumer::PESConsumer (Decoder* d, PES* p) : Consumer(d)
{
    pes = p;
    cstate = CSTART;
    pes_out_flag = TRUE;
    esport = NULL;
}

static int char2int (char c)
{
    int result = 0;
    for (int i = 7; i >= 0; i--)
    {
        if ((c & (1 << i)) == 0) result = result * 2;
        else result = result * 2 + 1;
    }
    return result;
}

/* This function is called to read TP payloads. */
int PESConsumer::read_partial (int n)
{
    int i = 0;
    while (i < n)
    {
        switch (cstate)
        {
            case CSTART:
            {
                buf = (char*) malloc (512);
                buf_index = 0;
                cstate = PCHEAD1;
                break;
            }
            case PCHEAD1:
            {
                if (buf_index < 9)
                {
                    buf[buf_index] = iport->read_byte();
                    buf_index++; i++;
                }
                if (buf_index == 9)
                {
                    header_length = char2int(buf[8]) + 9;
                    cstate = PCHEAD2;
                }
                break;
            }
            case PCHEAD2:
            {
                if (buf_index < header_length)
                {
                    buf[buf_index] = iport->read_byte();
                    buf_index++; i++;
                }
                if (buf_index == header_length)
                {
                    InputPort* save_port = iport;
                    iport = new IPortFromRam(buf, header_length);
                    read_pes_header();
                    delete iport;
                    iport = save_port;
                    if (pes_out_flag)
                        write_elementary_stream_chunk(buf, header_length);
                    delete buf; buf_index = 0; // done with buf
                    cstate = PCPAYLOAD;
                }
                break;
            }
            case PCPAYLOAD:
            {
                // dispense with remaining n-i payload bytes
                int plen = n - i;
                char* pbuf = (char*) malloc(plen);
                for (int j = 0; j < plen; j++)
                {
                    pbuf[j] = iport->read_byte();
                    i++;
                }
                write_elementary_stream_chunk(pbuf, plen);
                delete pbuf;
                assert(i == n);
                // stay in PCPAYLOAD state;
                // state changes when payload_unit_start_indicator is seen
                break;
            }
            default:
                sys_error("unknown PESConsumer state");
        }
    }
    return i;
}

void PESConsumer::flush ()
{
    esport->flush();
}

void PESConsumer::write_elementary_stream_chunk (char* pbuf, int plen)
{
    // prepare esport
    if (!esport)
    {
        // use decoder->get_ts()->pid or pes->stream_id?
        int cur_pid = decoder->get_ts()->pid;
        // FIX - provide access function to current pid from decoder?
        char fileName[32];
        sprintf(fileName, "stream%d", cur_pid);
        esport = new OPortToFile(fileName, 0);
    }

    // output loop
    for (int i = 0; i < plen; i++)
    {
        esport->write_byte(pbuf[i]);
    }
}

void PESConsumer::read_pes_header ()
{
    int result = iport->read_pattern("00000000000000000000000000000001");

    pes->stream_id = iport->read_uimbf(8, "reading stream_id");

    pes->PES_packet_length = iport->read_uimbf(16, "reading PES_packet_length");

    if ((pes->stream_id != id_private_stream_2) && (pes->stream_id != id_padding_stream))
    {
        // read '01'
        int result = iport->read_pattern("10");

        // read PES_scrambling_control; currently uses only '00' or '11'
        char scramble1 = iport->read_bit();
        char scramble2 = iport->read_bit();
        if ((scramble1 == ZERO) && (scramble2 == ZERO))
            pes->scrambling_code = '0';
        else
            pes->scrambling_code = 'S';

        // read PES_priority
        char priority = iport->read_bit();
        if (priority == ONE)
            pes->priority_code = 'H';
        else
            pes->priority_code = 'L';

        // read data_alignment_indicator
        char align = iport->read_bit();
        if (align == ONE)
            pes->alignment_indicator_code = 'A';
        else
            pes->alignment_indicator_code = '0';

        // read copyright
        char copyright = iport->read_bit();
        if (copyright == ONE)
            pes->copyright_code = 'C';
        else
            pes->copyright_code = '0';

        // read copy_or_original
        char original = iport->read_bit();
        if (original == ONE)
            pes->original_code = 'C';
        else
            pes->original_code = '0';

        // read 8 flags
        char pts_flag = iport->read_bit();
        char dts_flag = iport->read_bit();
        char esrc_flag = iport->read_bit();
        char esrate_flag = iport->read_bit();
        char dsm_trick_flag = iport->read_bit();
        char additional_copy_flag = iport->read_bit();
        char crc_flag = iport->read_bit();
        char extension_flag = iport->read_bit();

        // read PES_header_data_length
        int PES_header_length =
            iport->read_uimbf(8, "reading PES_header_length");

        // read optional fields
        // read pts,dts
        if ((pts_flag == ONE) && (dts_flag == ZERO))
        {
            iport->read_pattern("0010");
            pes->PTS = iport->read_timestamp90();
            pes->DTS = NULL;
            decoder->get_manager()->Trigger(PTS_DTS_Event, pes);
        }
        else if ((pts_flag == ONE) && (dts_flag == ONE))
        {
            iport->read_pattern("0011");
            pes->PTS = iport->read_timestamp90();
            iport->read_pattern("0001");
            pes->DTS = iport->read_timestamp90();
            decoder->get_manager()->Trigger(PTS_DTS_Event, pes);
        }
        else
        {
            pes->PTS = NULL;
            pes->DTS = NULL;
        }

        // read esrc
        if (esrc_flag == ONE)
        {
            iport->read_reserved_bits(2);
            pes->ESCR = iport->read_timestamp27_pes_format();
        }
    }
}

```

```

// read es_rate
if (esrate_flag == ONE)
{
    iport->read_markerbit();
    pes->ES_rate = iport->read_uimbf(22, "reading ES_rate");
    iport->read_markerbit();
}

// read DSM_trick_mode
if (dsm_trick_flag == ONE)
{
    read_trick_mode();
}

// read additional_copy_info
if (additional_copy_flag == ONE)
{
    iport->read_markerbit();
    pes->additional_copy_info = NULL;
    char c;
    for (int i = 0; i < 7; i++)
    {
        c = iport->read_bit();
        if (c == '1')
        {
            pes->additional_copy_info |= (1 << i);
        }
    }
}

// read previous_PES_packet_CRC
if (crc_flag == ONE)
{
    pes->previous_CRC = iport->read_uimbf(16, "reading previous_CRC");
}

// read PES extension fields
if (extension_flag == ONE)
{
    read_extension();
}

// skip the stuffing bytes. N.B. the '9' added to PES_header_length
pes->number_stuffing_bytes =
    PES_header_length + 9 - pes->get_header_data_length();
for (int i = 0; i < pes->number_stuffing_bytes; i++)
    iport->read_byte();
}

// call PESParsed callback
decoder->get_manager()->Trigger(PESHeaderParsed, pes);
}

void PESConsumer::read_trick_mode ()
{
    char intra;
    pes->trick_mode = new TrickMode();
    int code = iport->read_uimbf(3, "reading trick_control_code");
    switch (code)
    {
        case 0:
            pes->trick_mode->trick_mode_control_code = 'F';
            read_field_id();
            intra = iport->read_bit();
            if (intra == ONE)
            {
                pes->trick_mode->intra_slice_refresh_code = 'I';
            }
            else
            {
                pes->trick_mode->intra_slice_refresh_code = '0';
            }
            read_frequency_truncation();
            break;
        case 1:
            pes->trick_mode->trick_mode_control_code = 'S';
            pes->trick_mode->field_rep_control =
                iport->read_uimbf(5, "reading field rep_control");
            break;
        case 2:
            pes->trick_mode->trick_mode_control_code = 'Z';
            read_field_id();
            iport->read_reserved_bits(3);
            break;
        case 3:
            pes->trick_mode->trick_mode_control_code = 'R';
            read_field_id();
            intra = iport->read_bit();
            if (intra == ONE)
            {
                pes->trick_mode->intra_slice_refresh_code = 'I';
            }
            else
            {
                pes->trick_mode->intra_slice_refresh_code = '0';
            }
            read_frequency_truncation();
            break;
        default:
            // It must be one of the reserved modes
            iport->read_reserved_bits(5);
            break;
    }
}

void PESConsumer::read_field_id ()
{
    if (pes->trick_mode == NULL) return;
    int code = iport->read_uimbf(2, "reading field_id");
    switch (code)
    {
        case 0:
            pes->trick_mode->field_id_code = '1';
            break;
        case 1:
            pes->trick_mode->field_id_code = '2';
            break;
        case 2:
            pes->trick_mode->field_id_code = 'B';
            break;
    }
}

void PESConsumer::read_frequency_truncation ()

```

```

{
    if (pes->trick_mode == NULL) return;
    int code = iport->read_uimbf(2, "reading frequency_truncation");
    switch (code)
    {
        case 0:
            pes->trick_mode->frequency_truncation_code = 'D';
            break;
        case 1:
            pes->trick_mode->frequency_truncation_code = '3';
            break;
        case 2:
            pes->trick_mode->frequency_truncation_code = '6';
            break;
        case 3:
            pes->trick_mode->frequency_truncation_code = 'A';
            break;
    }
}

void PESConsumer::read_extension ()
{
    pes->extension = new PESExtension();

    // read flags
    char PES_private_data_flag = iport->read_bit();
    char pack_header_field_flag = iport->read_bit();
    char pp_seq_counter_flag = iport->read_bit();
    char P_STD_buffer_flag = iport->read_bit();
    iport->read_reserved_bits(3);
    char PES_extension_field_flag = iport->read_bit();

    if (PES_private_data_flag == ONE)
    {
        pes->extension->PES_private_data = (char*) malloc(16);
        for (int i = 0; i < 16; i++)
            pes->extension->PES_private_data[i] = iport->read_byte();
    }

    if (pack_header_field_flag == ONE)
    {
        pes->extension->pack_field_length =
            iport->read_uimbf(8, "reading pack_field_length");
        for (int j = 0; j < pes->extension->pack_field_length; j++)
            pes->extension->pack_header[j] = iport->read_byte();
    }

    if (pp_seq_counter_flag == ONE)
    {
        pes->extension->pp_seq_counter_code = 'P';
        iport->read_markerbit();
        pes->extension->pp_seq_counter =
            iport->read_uimbf(7, "reading pp_seq_counter");
        iport->read_markerbit();
        pes->extension->original_stuff_length =
            iport->read_uimbf(7, "reading orig. stuff len");
    }
    else
        pes->extension->pp_seq_counter_code = '0';

    if (P_STD_buffer_flag == ONE)
    {
        pes->extension->P_STD_buffer_code = 'B';
        iport->read_pattern("01");
        pes->extension->P_STD_buffer_scale = iport->read_bit();
        pes->extension->P_STD_buffer_size =
            iport->read_uimbf(13, "reading P_STD_buf_size");
    }
    else
        pes->extension->P_STD_buffer_code = '0';

    if (PES_extension_field_flag == ONE)
    {
        iport->read_markerbit();
        pes->extension->PES_extension_field_length =
            iport->read_uimbf(7, "reading PES_ext_field_length");
        for (int k = 0; k < pes->extension->PES_extension_field_length; k++)
            (void) iport->read_reserved_bits(8);
    }
}

```

3.3.27 PESConsumer.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

// PESConsumer class */
#ifndef PESCONSUMER_H
#define PESCONSUMER_H

#include "Consumer.H"
#include "PES.H"
#include "OutputPort.H"

// PESConsumer CState values
#define PCHEAD1 1
#define PCHEAD2 2
#define PCPAYLOAD 3

```

```

class PESConsumer : public Consumer
{
public:
    PESConsumer (Decoder*, PES*);
    PES* pes;
    virtual int read_partial (int);
    void read_pes_header ();
    int pes_out_flag;
    OutputPort* esport;
    virtual void flush ();
private:
    char* buf;
    int buf_index;
    int header_length;
    void read_trick_mode ();
    void read_field_id ();
    void read_frequency_truncation ();
    void read_extension ();
    void write_elementary_stream_chunk (char*, int);
};

/*
DOCUMENTATION

PESConsumers will read PES packets from InputPorts. The member
variable, iport, is inherited from the base class Consumer.

Payload from the PES packets is written to the OutputPort, esport.
The member esport can either be set by the application or it will be
set internally. The actual payload bytes are ignored. Entire PES
packets are written to esport if the flag pes_out_flag is TRUE.

Public members are:

PESConsumer (Decoder*, PES*);
    The only constructor.

PES* pes;
    pes is set during the parse of the pes packet to reflect the
    structure of the packet. Thus an application can call
    read_pes_header and then examine the pes member to determine the
    structure of the packet.

virtual int read_partial (int);
    Will read a specified number of bytes into the buffer buf. When
    the end of the PES header is reached the header is parsed and a
    PESHeaderParsed event is triggered.

void read_pes_header ();
    This function can be called directly by an application but is also
    used internally by the member function read_partial to parse the
    PES header when the entire header has arrived.

int pes_out_flag;
    When set to TRUE the PES header is output with the elementary
    stream onto the port esport.

OutputPort* esport;
    Payload from the PES packet is output to esport. If it has a null
    value when it is needed, then it is set with the call:
    esport = new OPortToFile(fileName, 0);
    where fileName is set to "stream<n>" where <n> is set to the
    decoder's current pid value.

virtual void flush ();
    This will flush the esport.
*/
#endif

```

3.3.28 PESProducer.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties of merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* PES class implementation */

#include "PESProducer.H"
#include "OPortToRam.H"
#include "Encoder.H"
#include "IPortFromFile.H"

extern "C"
{
#include <malloc.h>
}

PESProducer::PESProducer (Encoder* e, PES* p) : Producer(e)
{
    pes = p;
    timestamp_lock = FALSE;
    esport = NULL; // defer initialization of esport until first usage
}

int PESProducer::fill_buffer ()
{
    if (buf) delete buf;
    length_stored = pes->PES_packet_length + 6;
    buf = (char*) malloc (length_stored);
}

```

```

OutputPort* save_port = oport;
oport = new OPortToRam(buf, length_stored, NULL);
send_pes_packet();
delete oport;
oport = save_port;
buf_index = 0;
return length_stored;
}

void PESProducer::send_pes_packet ()
{
    oport->write_pattern("00000000000000000000000000000001");

    oport->write_uimshf(pes->stream_id, 8);

    oport->write_uimshf(pes->PES_packet_length, 16);

    if ((pes->stream_id != id_private_stream_2) &&
        (pes->stream_id != id_padding_stream))
    {
        oport->write_pattern("10");

        // write PES_scrambling_control; currently uses only "00" or "11"
        if (pes->scrambling_code == 'S') oport->write_pattern("11");
        else oport->write_pattern("00");

        // write PES_priority
        if (pes->priority_code == 'H') oport->write_bit('1');
        else oport->write_bit('0');

        // write data_alignment_indicator
        if (pes->alignment_indicator_code == 'A') oport->write_bit('1');
        else oport->write_bit('0');

        // write copyright
        if (pes->copyright_code == 'C') oport->write_bit('1');
        else oport->write_bit('0');

        // write copy_or_original
        if (pes->original_code == 'O') oport->write_bit('1');
        else oport->write_bit('0');

        // write PTS_DTS_flags
        if (pes->PTS) oport->write_bit('1'); else oport->write_bit('0');
        if (pes->DTS) oport->write_bit('1'); else oport->write_bit('0');

        // write ESCR flag
        if (pes->ESCR) oport->write_bit('1'); else oport->write_bit('0');

        // write ES_rate_flag
        if (pes->ES_rate) oport->write_bit('1'); else oport->write_bit('0');

        // write DSM_trick_mode_flag
        if (pes->trick_mode) oport->write_bit('1'); else oport->write_bit('0');

        // write additional_copy_info_flag
        if (pes->additional_copy_info) oport->write_bit('1'); else oport->
        write_bit('0');

        // write PES_CRC_flag
        if (pes->previous_CRC) oport->write_bit('1'); else oport->
        write_bit('0');

        // write PES_extension_flag
        if (pes->extension) oport->write_bit('1'); else oport->write_bit('0');

        // write PES_header_data_length
        oport->write_uimshf(pes->get_header_length() - 9, 8);

        // output all optional fields
        // output PTS, DTS
        if (pes->PTS)
        {
            if (timestamp_lock == FALSE)
            {
                TimeStamp90 PTS = encoder->sys_time + encoder->sys_const;
                pes->PTS = &PTS;
                // FIX -- distinguish between PTS and DTS semantics
                if (pes->DTS) pes->DTS = pes->PTS;
            }
            if (pes->PTS && !pes->DTS)
            {
                oport->write_pattern("0010");
                oport->write_timestamp90(pes->PTS);
            }
            if (pes->PTS && pes->DTS)
            {
                oport->write_pattern("0011");
                oport->write_timestamp90(pes->PTS);
                oport->write_timestamp90(pes->DTS);
                oport->write_timestamp90(pes->DTS);
            }
        }
        // output ESCR
        if (pes->ESCR)
        {
            oport->write_reserved_bits(2);
            oport->write_timestamp27_pes_format(pes->ESCR);
        }

        // output ES_rate
        if (pes->ES_rate)
        {
            oport->write_markerbit();
            oport->write_uimshf(pes->ES_rate, 22);
            oport->write_markerbit();
        }

        // output DSM_trick_mode field
        if (pes->trick_mode)
            send_trick_mode();

        // output additional_copy_info
        if (pes->additional_copy_info)
        {
            oport->write_markerbit();
            char mask = (char) 1;
            for (int i = 0; i < 7; i++)
            {
                if (pes->additional_copy_info & mask)
                    oport->write_bit('1'); else oport->write_bit('0');
                mask = mask << 1;
            }
        }
    }
}

```

```

// output previous_PES_packet_CRC
if (pes->previous_CRC)
    oport->write_uimbf(pes->previous_CRC, 16);

// output PES extension fields
if (pes->extension)
    send_extension();

// output stuffing bytes
for (int i = 0; i < pes->number_stuffing_bytes; i++)
    oport->write_byte(0xff);

// output payload
send_payload();
}

void PESProducer::send_trick_mode ()
{
    if (pes->trick_mode == NULL) return;
    switch (pes->trick_mode->trick_mode_control_code)
    {
        case 'F':
            oport->write_pattern("000");
            send_field_id();
            if (pes->trick_mode->intra_slice_refresh_code == '1')
                oport->write_bit('1');
            else
                oport->write_bit('0');
            send_frequency_truncation();
            break;
        case 'S':
            oport->write_pattern("001");
            oport->write_uimbf(pes->trick_mode->field_rep_control, 5);
            break;
        case 'Z':
            oport->write_pattern("010");
            send_field_id();
            oport->write_reserved_bits(3);
            break;
        case 'R':
            oport->write_pattern("011");
            send_field_id();
            if (pes->trick_mode->intra_slice_refresh_code == '1')
                oport->write_bit('1');
            else
                oport->write_bit('0');
            send_frequency_truncation();
            break;
    }
}

void PESProducer::send_field_id ()
{
    if (pes->trick_mode == NULL) return;
    switch (pes->trick_mode->field_id_code)
    {
        case '1':
            oport->write_pattern("00");
            break;
        case '2':
            oport->write_pattern("01");
            break;
        case 'B':
            oport->write_pattern("10");
            break;
    }
}

void PESProducer::send_frequency_truncation ()
{
    if (pes->trick_mode == NULL) return;
    switch (pes->trick_mode->frequency_truncation_code)
    {
        case 'D':
            oport->write_pattern("00");
            break;
        case '3':
            oport->write_pattern("01");
            break;
        case '6':
            oport->write_pattern("10");
            break;
        case 'A':
            oport->write_pattern("11");
            break;
    }
}

void PESProducer::send_extension ()
{
    if (pes->extension == NULL) return;

    // output flags
    if (pes->extension->PES_private_data) oport->write_bit('1');
    else oport->write_bit('0');

    if (pes->extension->pack_header) oport->write_bit('1');
    else oport->write_bit('0');

    if (pes->extension->pp_seq_counter_code == '1') oport->write_bit('1');
    else oport->write_bit('0');

    if (pes->extension->P_STD_buffer_code == '1') oport->write_bit('1');
    else oport->write_bit('0');

    oport->write_reserved_bits(3);

    if (pes->extension->PES_extension_field_length) oport->write_bit('1');
    else oport->write_bit('0');

    // output PES_private_data
    if (pes->extension->PES_private_data)
    {
        for (int i = 0; i < 16; i++)
        {
            oport->write_byte(pes->extension->PES_private_data[i]);
        }
    }

    // output pack_header
    if (pes->extension->pack_header)
    {
        oport->write_uimbf(pes->extension->pack_field_length, 8);
        for (int j = 0; j < pes->extension->pack_field_length; j++)

```

```

        {
            oport->write_byte(pes->extension->pack_header[j]);
        }
    }

    // output pp_seq_counter
    if (pes->extension->pp_seq_counter_code == ONE)
    {
        oport->write_markerbit();
        oport->write_uimbf(pes->extension->pp_seq_counter, 7);
        oport->write_markerbit();
        oport->write_uimbf(pes->extension->original_stuff_length, 7);
    }

    // output P_STD_buffer
    if (pes->extension->P_STD_buffer_code == ONE)
    {
        oport->write_pattern("01");
        oport->write_bit(pes->extension->P_STD_buffer_scale);
        oport->write_uimbf(pes->extension->P_STD_buffer_size, 13);
    }

    // output PES_extension_field
    if (pes->extension->PES_extension_field_length)
    {
        oport->write_markerbit();
        oport->write_uimbf(pes->extension->PES_extension_field_length, 7);
        for (int k = 0; k < pes->extension->PES_extension_field_length; k++)
        {
            oport->write_byte(0xff);
        }
    }
}

void PESProducer::send_payload ()
{
    // calculate payload length
    int payload_length = pes->get_length() - pes->get_header_length();
    if (payload_length <= 0) return;

    // prepare esport
    if (!esport)
    {
        esport = new IPortFromFile(pes->file_name, CYCLE);
    }

    // read/write loop
    char byte;
    for (int i = 0; i < payload_length; i++)
    {
        byte = esport->read_byte();
        oport->write_byte(byte);
    }
}

```

3.3.29 PESProducer.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

```

```

/* PESProducer class */
#ifndef pesproducer_h
#define pesproducer_h

#include "Producer.H"
#include "PES.H"
#include "InputPort.H"
#include "Utilities.H"

class PESProducer : public Producer
{
public:
    PESProducer (Encoder*, PES*);
    PES* pes;
    void send_pes_packet ();
    Boolean timestamp_lock;
    InputPort* esport;
protected:
    virtual int fill_buffer ();
private:
    void send_trick_mode ();
    void send_field_id ();
    void send_frequency_truncation ();
    void send_extension ();
    void send_payload ();
};

```

DOCUMENTATION

PESProducers will write PES packets to OutputPorts. The member variable, oport, is inherited from the base class Producer.

Payload for the PES packets is read from the InputPort, esport. The member esport can be either set by the application or it will be set internally. The actual payload bytes are ignored.

Public members are:

```

PESProducer (Encoder*, PES*);
    The only constructor.

PES* pes;
    pes is used as a template for PES packets. Some members of pes
    may be modified as the packet is generated (e.g. PTS and DTS, see
    the description of timestamp_lock below).

void send_pes_packet ();
    This function can be called directly by an application but is also
    used internally by the member function fill_buffer to fill the
    internal Producer buffer, buf, with a PES packet.

TState timestamp_lock;
    When set to TRUE prohibits the restamping of the PTS and DTS
    fields in the pes structure - whatever is in the pes structure is
    output regardless of the encoder's clock values. When set to
    FALSE the encoder is queried for the current system time
    (sys_time) and system end-to-end constant (sys_const) the PTS is
    set to their sum. The default is FALSE.

InputPort* esport;
    Payload for the PES packet comes from here. If it has a null
    value when it is needed, then it is set with the call:
    esport = new IPortFromFile(pes->file_name, CYCLE);
*/

#endif

```

3.3.30 Poly.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Poly implementation */

#include "Poly.H"

extern "C"
{
#include <stdio.h>
}

static unsigned short int g[] =
{0,1,1,0,1,1,0,1,
 1,0,1,1,1,0,0,0,
 1,0,0,0,0,0,1,1,
 0,0,1,0,0,0,0,0};

Poly::Poly ()
{
    for (int i=0; i<32; i++)
        shift_reg[i] = 1;
}

void Poly::push_bit (char data_bit) // data_bit range = '0', '1'
{
    int i;
    char obit;

    obit = data_bit ^ '0';

    data_bit ^= shift_reg[31];
    i = 31;
    while (i != 0)
    {
        if (g[i])
            shift_reg[i] = shift_reg[i-1] ^ data_bit;
        else
            shift_reg[i] = shift_reg[i-1];
        i--;
    }
    shift_reg[0] = data_bit;
}

int Poly::poly_cmp (char* bstr)
{
    char bit;
    int i;
    unsigned long vali, valg;

    // bit 31 is MSB
    vali = valg = 0;
    for (i=31; i >= 0; i--)
    {
        vali <<= 1;
        valg <<= 1;
        vali += bstr[i] - '0';
        valg += shift_reg[i];
    }
    if (vali == valg) return 1;
    else if (vali == -valg) return 2;
    else return 0;
}

void Poly::print_bits ()
{
    for (int i = 31; i >= 0; --i)
        puts('0'+shift_reg[i], stdout);
}

```

3.3.31 Poly.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Poly classes */
#ifndef poly_h
#define poly_h

typedef unsigned long CRC32;

class Poly
{
public:
    Poly ();
    void push_bit (char);
    int poly_cmp (char*);
    void print_bits ();
    unsigned char shift_reg[32];
private:
    CRC32 crc;
};

/*
DOCUMENTATION

Poly is used by InputPorts and OutputPorts for computing CRCs. The
32 bit polynomial specified in 13818-1 is "hardwired" into the code.
*/

#endif

```

3.3.32 Producer.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Producer class implementation */

#include "Utilities.H"
#include "Producer.H"
#include "OutputPort.H"

extern "C"
{
#include <stdio.h>
}

Producer::Producer (Encoder* e)
{
    encoder = e;
    length_stored = 0;
    buf_index = 1;
}

void Producer::connect (OutputPort* p)
{
    oport = p;
}

int Producer::fill_buffer ()
{
    // do nothing in the base class
    return 0;
}

int Producer::get_nready ()
{
    if (length_stored < buf_index) return 0;
    return length_stored - buf_index;
}

```

```

)
int Producer::send_partial (int n)
{
    if (buf_index >= length_stored)
    {
        // fill buffer
        length_stored = fill_buffer();
    }
    // now send bytes
    for (int i = 0; i < n; i++)
    {
        if (buf_index < length_stored)
        {
            oport->write_byte(buf[buf_index]);
            buf_index++;
        }
        else
        {
            oport->write_byte(0xff);
            sys_message("0xff padding inserted by Producer::send_partial");
        }
    }
    if (buf_index < length_stored)
        return 0;
    else
        return 1;
}

```

3.3.33 Producer.H

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

/* Producer class */
#ifndef producer_h
#define producer_h

class OutputPort;
class Encoder;

class Producer
{
public:
    Producer (Encoder*);
    void connect (OutputPort*);
    virtual int fill_buffer ();
    virtual int get_nready ();
    int send_partial (int);
protected:
    Encoder* encoder;
    OutputPort* oport;
    char* buf;
    int buf_index;
    int length_stored;
};

/*
DOCUMENTATION

Producers write things to OutputPorts. Typical subclasses are
TSProducer, PESProducer, PATProducer, MapProducer, etc. Although
some of these classes define methods for writing directly on the
OutputPort (e.g. TSProducer), Producers can also be set up to first
fill an internal buffer (via the fill_buffer method). Subsequent
calls to send_partial will send bytes from the buffer to the OutputPort.

Public members are:

Producer (Encoder*);
The only constructor. All Producers have an Encoder object
available for such things as reading a encoder clock for the
generation of timestamps.

void connect (OutputPort*);
To connect to an OutputPort.

virtual int fill_buffer ();
Defined in subclasses to fill the buffer, buf.

virtual int get_nready ();
Returns the number of bytes ready for output (e.g. what's left in
the buffer).

int send_partial (int);
Will send a specified number of bytes to the OutputPort. If the
number requested exceeds the number available in the buffer, then
the output is padded with 0xff bytes.
*/
#endif

```

3.3.34 Section.C

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
```

```

 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license
 * fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
 * all warranties, whether express, implied, or statutory, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose. In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and
 * user's customers, employees, agents, transferees, successors, and
 * assigns.
 *
 * The Tektronix Inc. does not represent or warrant that the programs
 * furnished hereunder are free of infringement of any third-party
 * patents.
 */

```

```
/* Section class implementation */
```

```
#include "Section.H"
```

```
extern "C"
```

```
{
#include <stdio.h>
}
```

```
Section::Section ()
```

```
{
}
```

```
void Section::print_a ()
```

```
{
    printf("pointer: %d\n", pointer);
    printf("table_id: %d\n", table_id);
    printf("section_syntax_indicator: %c\n", section_syntax_code);
    printf("section_length: %d\n", section_length);
}
```

```
void Section::print_b ()
```

```
{
    printf("version_number: %d\n", version_number);
    printf("current_next_indicator: %c\n", current_next_code);
    printf("section_number: %d\n", section_number);
    printf("last_section_number: %d\n", last_section_number);
}
```

```
PATSection::PATSection () : Section ()
```

```
{
}
```

```
void PATSection::print ()
```

```
{
    printf("PAT Section\n");
    print_a();

    printf("transport_stream_id = %d\n", transport_stream_id);
    print_b();
}
```

```
MapSection::MapSection (Program* p) : Section ()
```

```
{
    program = p;
    if (p->pcr_estream) pcr_pid = p->pcr_estream->pid;
    else pcr_pid = UNASSIGNED;
    program_number = p->program_number;
}
```

```
void MapSection::print ()
```

```
{
    printf("Map Section\n");
    print_a();

    printf("program_number = %d\n", program_number);
    print_b();

    printf("pcr_pid = %d\n", pcr_pid);
    printf("program_info_length = %d\n", program_info_length);
}
```

```
CASection::CASection () : Section ()
```

```
{
}
```

```
void CASection::print ()
```

```
{
    printf("CA Section\n");
    print_a();
    print_b();

    // FIX -- print Section loops
    printf("print CA Section loop\n");
}
```

```
PriSection::PriSection () : Section ()
```

```
{
}
```

3.3.35 Section.H

```
/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
```

```

 *
 * Usage Restrictions
 *
 * License is granted to copy, to use, and to make and to use derivative
 * works for research and evaluation purposes only.
 *
 * Disclaimer of Warranty
 *

```

```

* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

/* Section class */
#ifdef section_h
#define section_h

class Section
{
public:
    Section ();
    int pointer; // offset into TP payload
    int table_id; // table_id
    char section_syntax_code; // (S,L) - Short or Long
    int section_length; // 0..1024
    int version_number; // 0..31
    char current_next_code; // (C,N) - Current table or Next table
    int section_number; // 0..255
    int last_section_number; // 0..255
    void print_a ();
    void print_b ();
};

/* PATSection class */
class PATSection : public Section
{
public:
    PATSection ();
    int transport_stream_id; // 16 bits, app specific
    void print ();
};

/* MapSection class */
#include "Directory.H"
class MapSection : public Section
{
public:
    MapSection (Program*);
    Program* program; // from directory
    int program_number; // 16 bits
    int pcr_pid; // 13 bits
    int program_info_length; // 12 bits
    void print ();
};

class CAsSection : public Section
{
public:
    CAsSection ();
    void print ();
};

/* PriSection class */
class PriSection : public Section
{
public:
    PriSection ();
    int table_id_extension; // 16 bits, user defined
};

/* DOCUMENTATION

The Section class and its subclasses are syntax classes; their
objects encapsulate the structure of the various PSI sections.
When generating packets (say in an encoder application) these
objects are used as templates. Changing the template will change the
structure of the sections being generated. When parsing sections (say
in a decoder application) these objects are set to reflect the structure
of the packet being parsed and can then be examined by the
application.

Mnemonic chars are used to represent the various values of the fields
(e.g. the current_next_code can be 'C' or 'N'). These are listed in
the above comments.

All the fields are exposed as public members.
*/

#endif

```

3.3.36 SectionConsumers.C

```

/* Copyright (C) 1995, Tektronix Inc. All Rights Reserved.
*
* Usage Restrictions
*
* License is granted to copy, to use, and to make and to use derivative
* works for research and evaluation purposes only.
*
* Disclaimer of Warranty
*
* These software programs are available to the user without any license
* fee or royalty on an "as is" basis. Tektronix Inc. disclaims any and
* all warranties, whether express, implied, or statutory, including any
* implied warranties or merchantability or of fitness for a particular
* purpose. In no event shall the copyright-holder be liable for any
* incidental, punitive, or consequential damages of any kind whatsoever
* arising from the use of these programs.
*
* This disclaimer of warranty extends to the user of these programs and
* user's customers, employees, agents, transferees, successors, and
* assigns.
*
*/

```

```

* The Tektronix Inc. does not represent or warrant that the programs
* furnished hereunder are free of infringement of any third-party
* patents.
*/

/* SectionConsumer class implementation */

#include "Utilities.H"
#include "IPortFromRam.H"
#include "SectionConsumers.H"
#include "Decoder.H"
#include "Events.H"
#include "PES.H"
#include "PESConsumer.H"

extern "C"
{
#include <malloc.h>
}

SectionConsumer::SectionConsumer (Decoder* d, Section* s) : Consumer(d)
{
    sec = s;
    buf_index = -1;
    length_expected = -1;
}

static int char2int (char c)
{
    int result = 0;
    for (int i = 7; i >= 0; i--)
    {
        if ((c & (1 << i)) == 0) result = result * 2;
        else result = result * 2 + 1;
    }
    return result;
}

/* This function is called to read TP payloads. */
int SectionConsumer::read_partial (int n)
{
    int i = 0;
    while (i < n)
    {
        switch (cstate)
        {
            case CSTART:
                buf = (char*) malloc (1024); // FIX -- private sections can be 4096
                buf_index = 0;
                cstate = SCHEAD1;
                break;
            case SCHEAD1:
                if (buf_index < 4)
                {
                    buf[buf_index] = iport->read_byte();
                    buf_index++; i++;
                }
                if (buf_index == 4)
                {
                    // FIX -- this case assumes ptr == 0
                    if (buf[0] != 0)
                    {
                        sys_error("non zero pointer offset in SectionConsumer");
                    }
                    length_expected = (char2int(buf[2]) & 0xF) << 8;
                    length_expected += char2int(buf[3]) + 4;
                    cstate = SCHEAD2;
                }
                break;
            case SCHEAD2:
                if (buf_index < length_expected)
                {
                    buf[buf_index] = iport->read_byte();
                    buf_index++; i++;
                }
                if (buf_index == length_expected)
                {
                    InputPort* save_port = iport;
                    iport = new IPortFromRam(buf, length_expected);
                    read_section();
                    delete iport;
                    iport = save_port;
                    delete buf; buf_index = 0; // done with buf
                    cstate = SCFLUSH;
                }
                break;
            case SCFLUSH:
                char c = iport->read_byte();
                assert(c == (char) 0xff);
                i++;
                break;
            default:
                sys_error("unknown SectionConsumer state");
        }
    }
    return n;
}

void SectionConsumer::read_header_a ()
{
    // read pointer field
    // FIX -- this assumes that there is always a pointer field
    sec->pointer = iport->read_uimbsf(8, "reading pointer field");

    // skip to start of section
    for (int k = 0; k < sec->pointer; k++)
        (void) iport->read_byte();

    // start CRC
    iport->start_crc();

    // read table_id (should be 0)
    sec->table_id = iport->read_uimbsf(8, "reading table_id");

    // read section_syntax_code
    char syntax = iport->read_bit();
    if (syntax == ONE)
        sec->section_syntax_code = 'S';
    else
        sec->section_syntax_code = 'L';

    // read '0' and reserved bits
    iport->read_bit();
    iport->read_reserved_bits(2);
}

```

```

// read section_length
sec->section_length = ipt->read_uimbsf(12, "reading section_length");
}

void SectionConsumer::read_header_b ()
{
    // read reserved bits
    ipt->read_reserved_bits(2);

    // read version_number
    sec->version_number = ipt->read_uimbsf(5, "reading version_number");

    // read current_next_indicator
    char cur_next = ipt->read_bit();
    if (cur_next == ONE)
        sec->current_next_code = 'C';
    else
        sec->current_next_code = 'N';

    // read section_number
    sec->section_number = ipt->read_uimbsf(8, "reading section_number");

    // read last_section_number
    sec->last_section_number = ipt->read_uimbsf(8, "reading
last_section_number");
}

void SectionConsumer::read_CRC ()
{
    // read CRC
    ipt->stop_crc();
    if (!ipt->check_crc())
        decoder->get_manager()->Trigger(BadCRC, sec);
}

PATConsumer::PATConsumer (Decoder* d, PATSection* s)
: SectionConsumer(d, s)
{
    patsec = s;
}

void PATConsumer::read_section ()
{
    // read header_a
    read_header_a();

    // read transport_stream_id
    patsec->transport_stream_id = ipt->read_uimbsf(16,
"reading trans_strm_id");

    // read header_b
    read_header_b();

    // create new directory; get old one
    Directory* newdir = new Directory();
    Directory* olddir = decoder->get_dir();

    int network_pid = decoder->get_networkpid();

    // read loop
    for (int i = 0; i < patsec->section_length - 9; i += 4)
    {
        int prog_number = ipt->read_uimbsf(16, "reading program number");
        ipt->read_reserved_bits(3);
        if (prog_number == 0)
        {
            network_pid = ipt->read_uimbsf(13, "reading network_pid");
        }
        else
        {
            int pid = ipt->read_uimbsf(13, "reading pid");
            // update Directory
            Program* program = olddir->get_program(prog_number);
            if (program)
            {
                // not a new program
                newdir->add_program(program);
            }
            else
            {
                // a new program
                newdir->add_program(new Program(prog_number, pid));
            }
        }
    }

    // setup decoder using new directory
    decoder->install_netpid(network_pid);
    decoder->install_dir(newdir);

    // read CRC
    read_CRC();

    // call PATParsed callback
    decoder->get_manager()->Trigger(PATParsed, patsec);
}

MapConsumer::MapConsumer (Decoder* d, MapSection* s)
: SectionConsumer(d, s)
{
    mapsec = s;
}

void MapConsumer::read_section ()
{
    // read header_a
    read_header_a();

    // read program_number
    mapsec->program_number = ipt->read_uimbsf(16, "reading program_number");

    // read header_b
    read_header_b();

    // read reserved bits
    ipt->read_reserved_bits(3);

    // read pcr_pid
    mapsec->pcr_pid = ipt->read_uimbsf(13, "reading pcr pid");

    // read reserved bits
    ipt->read_reserved_bits(4);

    // create new program with same program number
    Program* newprog = new Program(mapsec->program_number,

decoder->get_ts()->pid);

// read program_info_length and info into newprog
mapsec->program_info_length =
    ipt->read_uimbsf(12, "reading program info length");
// FIX - implement descriptors for program info
for (int i = 0; i < mapsec->program_info_length; i++)
{
    (void) ipt->read_byte();
}

// read loop
int bytes_in_loop =
    mapsec->section_length - mapsec->program_info_length - 13;
for (int j = 0; j < bytes_in_loop; )
{
    // read fields
    int stream_type = ipt->read_uimbsf(8, "reading stream_type");
    ipt->read_reserved_bits(3);
    int elem_pid = ipt->read_uimbsf(13, "reading elem_pid");

    EStream* estream = new EStream((StreamType) stream_type, elem_pid,
NULL);
    newprog->add_estream(estream);

    // read estream info
    ipt->read_reserved_bits(4);
    int info_length = ipt->read_uimbsf(12, "reading es_info_length");
    // FIX - implement descriptors for estream info
    for (int k = 0; k < info_length; k++)
        (void) ipt->read_byte();

    j += 5 + info_length;
}

// get dir and install new program
decoder->install_prog(newprog);

// read CRC
read_CRC();

// call MapParsed callback
decoder->get_manager()->Trigger(MapParsed, mapsec);
}

CAConsumer::CAConsumer (Decoder* d, CASection* s)
: SectionConsumer(d, s)
{
    casec = s;
}

void CAConsumer::read_section ()
{
    // read header_a
    read_header_a();

    // read 16/18 reserved bits, read_header_b() will read 2 more bits.
    ipt->read_reserved_bits(16);

    // read header_b
    read_header_b();

    // read loop
    int bytes_in_loop = casec->section_length - 9;
    for (int j = 0; j < bytes_in_loop; )
    {
        // read descriptor
        int descriptor_tag = ipt->read_uimbsf(8, "reading des_tag");
        if (descriptor_tag != 9)
            sys_message("bad descriptor in CASection");

        int descriptor_length = ipt->read_uimbsf(8, "reading des_length");
        int CA_system_id = ipt->read_uimbsf(16, "reading CA_system_id");

        ipt->read_reserved_bits(3);

        int ca_pid = ipt->read_uimbsf(13, "reading des_tag");
        // FIX -- eventually add a CAConsumer to consume ca_pid; ignore for now
        for (int k = 0; k < descriptor_length - 4; k++)
            (void) ipt->read_byte();

        j += descriptor_length + 2;
    }

    // read CRC
    read_CRC();

    // call CAParsed callback
    decoder->get_manager()->Trigger(CAParsed, casec);
}

PriConsumer::PriConsumer (Decoder* d, PriSection* s)
: SectionConsumer(d, s)
{
    prisec = s;
}

void PriConsumer::read_section ()
{
    // read header_a
    read_header_a();

    // check syntax indicator
    if (prisec->section_syntax_code == 'S')
    {
        for (int i = 0; i < prisec->section_length; i++)
            (void) ipt->read_byte();
    }
    else
    {
        // read table_id_extension
        prisec->table_id_extension =
            ipt->read_uimbsf(16, "reading table_id_ext");

        // read header_b
        read_header_b();

        // read loop
        int bytes_in_loop = prisec->section_length - 9;
        for (int j = 0; j < bytes_in_loop; j++)
        {
            (void) ipt->read_byte();
        }
    }
}

```