

AN ALTERNATIVE WORKFLOW PROPOSAL FOR TV  
GRAPHICS GENERATION USING THE ANCILLARY DATA  
SPACE ON MEDIA STREAM

Guilherme Dantas Couto

Projeto de Graduação apresentado ao Curso  
de Engenharia Eletrônica e de Computação  
da Escola Politécnica, Universidade Federal  
do Rio de Janeiro, como parte dos requisitos  
necessários à obtenção do título de Enge-  
nheiro.

Orientador: Eduardo A. B. da Silva

Rio de Janeiro

Agosto de 2023

AN ALTERNATIVE WORKFLOW PROPOSAL FOR TV  
GRAPHICS GENERATION USING THE ANCILLARY DATA  
SPACE ON MEDIA STREAM

Guilherme Dantas Couto

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO  
CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA  
POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO  
PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU  
DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

  
\_\_\_\_\_  
Guilherme Dantas Couto

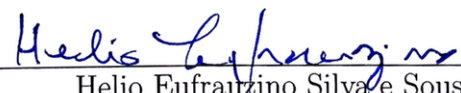
Orientador:

  
\_\_\_\_\_  
Prof. Eduardo Antônio Barros da Silva, Ph. D.

Examinador:

  
\_\_\_\_\_  
Prof. Jose Gabriel Rodriguez Carneiro Gomes, Ph. D.

Examinador:

  
\_\_\_\_\_  
Helio Eufrauzino Silva e Sousa, Eng.

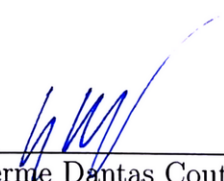
Rio de Janeiro

Agosto de 2023

## Declaração de Autoria e de Direitos

Eu, *Guilherme Dantas Couto*, CPF 135.657.147-60, autor da monografia *An alternative workflow proposal for TV graphics generation using the ancillary data space on media stream*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e ideias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.



---

Guilherme Dantas Couto

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica - Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro - RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmear ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es).



## AGRADECIMENTOS

À minha mãe, Josiane, e meu pai, Emilson, pela dedicação, amor e suporte incondicional que pavimentaram a estrada trilhada até aqui.

Aos meus irmãos, Henrique e Wallace, pelo alívio cômico, sarcasmo barato e a doce acidez que permeia nossas relações.

À *Olympic Broadcasting Services* e, em especial, a Stefano Frattini pela confiança e oportunidades que inspiraram a elaboração deste projeto.

Aos docentes que marcaram minha passagem pela UFRJ: Professores Luiz Wagner, Paulo Diniz e Eduardo pela devoção ao ensino, sugestões e orientação.

Aos muitos amigos que fiz durante minhas passagens pela TV Globo e aos integrantes da banca examinadora.

Aos bons brasileiros que, assim como eu, acreditam neste país e contribuíram indiretamente com seu suor para a realização desta monografia.

## RESUMO

Quando se observa o modelo mais comum de produção de mídia e televisão, é possível destacar algumas etapas técnicas fundamentais pelas quais o conteúdo normalmente é submetido: a captação, a edição, o arquivamento e a distribuição. Durante a edição — que pode ser realizada *ao vivo* ou não —, geralmente são sobrepostas às imagens tarjas, caracteres e outros componentes visuais com intuito de enriquecer a produção. Em transmissões esportivas ou programas jornalísticos, esta ferramenta pode desempenhar papel central na comunicação com o telespectador. Entretanto, nos fluxos habituais de trabalho, o grafismo é renderizado diretamente sobre o material em vídeo. Em outras palavras, elementos e informações estampadas nas imagens não poderão ser modificados ou removidos nas etapas subsequentes da cadeia de produção — um aspecto desvantajoso. Este projeto apresenta um modelo alternativo, explorando uma região de dados auxiliares encapsulados junto ao fluxo de mídia. Para isso, se propõe que um conjunto de metadados seja utilizado para caracterizar peças gráficas e seu conteúdo, de forma que a renderização possa ser efetuada de modo mais flexível ao longo da rede de distribuição. Uma aplicação prática será apresentada, demonstrando como o novo modelo poderia ser adotado para a geração síncrona e automatizada de grafismo em mais de um idioma — o que pode ser especialmente útil em grandes eventos esportivos internacionais.

Palavras-Chave: TV, grafismo, multilíngua, metadados.

## ABSTRACT

When observing the most common model of media and television production, it is possible to highlight some fundamental technical steps in which content typically has to go through: capture, editing, archiving, and distribution. During the editing process, whether conducted *live* or not, overlays such as banners, characters, and other visual components are often superimposed onto the footage to enhance the production. In sports broadcasts or news programs, this tool might play a central role in communicating with the audience. However, in typical workflows, graphics are directly rendered onto the video material, which may be a disadvantageous aspect in certain circumstances. In other words, elements and information displayed onscreen cannot be modified or removed in subsequent stages of the production chain. This project discusses an alternative model, by exploring a region of *auxiliary data* encapsulated within the media stream. To accomplish this, it is proposed that a set of metadata be used to characterize graphic elements and their content, allowing the rendering process to be performed in a more flexible manner across the distribution network. A practical application will be presented, demonstrating how the novel workflow could be adopted for synchronous and automated generation of graphics in multiple languages, which can be particularly useful in large scale sports events.

Key-words: TV, graphics, multi-language, metadata.

## ACRONYMS

A/D — Analog to Digital  
ANC or ANCI — Ancillary Data  
ANSI — American National Standards Institute  
API — Application Programming Interface  
ASI — Asynchronous Serial Interface  
AVC — Advanced Video Coding  
BNC — Bayonet Neill-Concelman  
CBR — Constant Bit Rate  
CC — Closed Captions  
CG — Computer Graphics or Character Generator  
CRC — Cyclic Redundancy Check  
CRT — Cathode Ray Tube  
CSS — Cascading Style Sheets  
CSV — Comma-separated Values  
DAI — Dynamic Ad Insertion  
DASH — Dynamic Adaptive Streaming over HTTP  
DSK — Downstream Keyer  
DTMF — Dual-Tone Multi-Frequency  
EAV — End of Active Video  
FPGA — Field Programmable Gate Array  
GFX — Graphics  
GOP — Group of Pictures  
GPI — General Purpose Input  
GPU — Graphics Processing Unit  
GUI — Graphical User Interface  
HANC — Horizontal Ancillary Data  
HD — High Definition  
HEVC — High Efficiency Video Coding  
HLS — HTTP Live Streaming  
HTML — HyperText Markup Language

HTTP — Hypertext Transfer Protocol  
I/O — Input/Output  
IEC — International Electrotechnical Commission  
IETF — Internet Engineering Task Force  
IGMP — Internet Group Management Protocol  
IOC — International Olympic Committee  
IP — Internet Protocol  
ISO — International Organization for Standardization  
ITU — International Telecommunication Union  
JPEG — Joint Photographic Experts Group  
JSON — JavaScript Object Notation  
MAM — Media Assets Management  
MPEG — Moving Picture Experts Group  
MRH — Media Rights Holder  
MUX — Multiplex or Multiplexer  
MXF — Media Exchange Format  
NDI — Network Device Interface  
NRZ — Non-Return to Zero  
NRZI — Non-Return to Zero Inverted  
NTSC — National Television Systems Committee  
OBS — Olympic Broadcasting Services or Open Broadcaster Software  
OS — Operational System  
OTT — Over-The-Top  
PAT — Program Association Table  
PCM — Pulse Code Modulation  
PMT — Program Map Table  
PTP — Precision Time Protocol  
PTS — Presentation Time Stamp  
RGB — Red, Green, Blue  
RHB — Rights Holder Broadcaster  
RTP — Real-time Transport Protocol  
RX — Receiver

SAV — Start of Active Video  
SCTE — Society of Cable Telecommunications Engineers  
SDI — Serial Digital Interface  
SDTV — Standard Definition Television  
SMPTE — Society of Motion Picture and Television Engineers  
SQL — Structured Query Language  
SRT — Secure Reliable Transport  
TCP — Transmission Control Protocol  
TX — Transmitter  
UDP — User Datagram Protocol  
UFRJ — Universidade Federal do Rio de Janeiro  
UHD — Ultra High Definition  
VANC — Vertical Ancillary Data  
VBI — Vertical Blanking Interval  
VBR — Variable Bit Rate  
VOD — Video-On-Demand  
VPID — Video Payload Identifier  
XML — Extensible Markup Language

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Organization . . . . .	4
<b>2</b>	<b>Production Workflow</b>	<b>5</b>
2.1	Broadcast stages . . . . .	5
2.1.1	Real-time infrastructure . . . . .	6
2.1.2	File-based systems . . . . .	7
2.1.3	Transport layer . . . . .	8
2.1.4	Overview . . . . .	9
2.2	Graphics . . . . .	10
2.2.1	Types of graphics . . . . .	10
2.2.2	Graphics engine . . . . .	11
2.3	Case study . . . . .	14
2.3.1	Major multi-feed events . . . . .	14
2.3.2	Topology drawbacks . . . . .	16
2.3.3	Overview . . . . .	19
<b>3</b>	<b>Standards Review</b>	<b>20</b>
3.1	SDI interface . . . . .	20
3.1.1	Video components . . . . .	22
3.1.2	Data format . . . . .	23
3.1.3	SDI evolution . . . . .	25
3.1.4	Migration to IP . . . . .	26
3.1.5	Ancillary data structure . . . . .	28

3.1.6	Overview . . . . .	29
3.2	MPEG-2 Transport Stream . . . . .	30
3.2.1	Stream structure . . . . .	30
3.2.2	Packet composition . . . . .	31
3.3	SCTE-35/104 Standards . . . . .	32
3.3.1	Splice points and events . . . . .	35
3.3.2	Normative schema . . . . .	35
3.3.3	Splice commands . . . . .	36
3.3.4	Injection principles . . . . .	37
3.3.5	Overview . . . . .	38
<b>4</b>	<b>Novel Proposal</b>	<b>39</b>
4.1	Introduction . . . . .	39
4.2	Engines redefined . . . . .	40
4.3	Domain . . . . .	42
4.4	Template library . . . . .	43
4.5	Notification format . . . . .	43
4.5.1	GFX command ID . . . . .	44
4.5.2	Standalone mode . . . . .	45
4.6	Ancillary manipulators . . . . .	46
4.7	Scenarios . . . . .	47
4.7.1	Multi-language graphics . . . . .	47
4.7.2	Adapted video formats . . . . .	50
<b>5</b>	<b>Experimental Implementation</b>	<b>51</b>
5.1	Introduction . . . . .	51
5.2	Methodology . . . . .	52
5.3	Tools and material . . . . .	53
5.3.1	Hardware . . . . .	53
5.3.2	TSDuck utilities . . . . .	54
5.3.3	NodeCG platform . . . . .	54
5.3.4	MongoDB . . . . .	55
5.3.5	Open Broadcaster Software . . . . .	55



5.3.6	<i>FFmpeg</i> . . . . .	56
5.4	Environment setup . . . . .	56
5.4.1	Relevant aspects . . . . .	56
5.4.2	Network . . . . .	57
5.4.3	Media preparation . . . . .	58
5.4.4	TS encapsulation . . . . .	60
5.5	Code development . . . . .	62
5.5.1	Bundles architecture . . . . .	62
5.5.2	Operational modes . . . . .	63
5.5.3	Configuration file . . . . .	65
5.5.4	Credentials file . . . . .	66
5.6	Results . . . . .	66
5.6.1	System organization . . . . .	66
5.6.2	Interface . . . . .	67
5.6.3	Database entries . . . . .	68
5.6.4	Graphics messages . . . . .	70
<b>6</b>	<b>Conclusions</b>	<b>74</b>
	<b>Bibliography</b>	<b>76</b>
<b>A</b>	<b>SCTE-35 adapted XML schema</b>	<b>80</b>
<b>B</b>	<b>Cobalt Insertion Overview</b>	<b>82</b>

# List of Figures

1.1	Graphics elements printed on card title . . . . .	2
1.2	Winner graphics stamped on footage . . . . .	3
2.1	Overview of main areas within a traditional media facility . . . . .	5
2.2	Production gallery overview, highlight to operators . . . . .	6
2.3	Major technology pillars of production . . . . .	9
2.4	Illustrative graphics examples . . . . .	10
2.5	Virtual graphics examples . . . . .	10
2.6	Overlays examples . . . . .	11
2.7	Highlight of graphics generation in a gallery . . . . .	12
2.8	Illustration of linear keying process . . . . .	12
2.9	Illustration of data fields mapping . . . . .	13
2.10	First stage of distribution at the venue . . . . .	15
2.11	Distribution from venues to IBC and MRHs . . . . .	15
2.12	Cropped graphics elements on smartphones . . . . .	17
2.13	Most TV graphics are not adapted for mobiles . . . . .	17
2.14	Graphics with defective data, as recorded . . . . .	18
2.15	Translated graphic overlay from korean broadcaster MBC . . . . .	19
3.1	Coaxial SDI cable terminated with BNC connector . . . . .	20
3.2	Color space change and digitizing process of a video source . . . . .	22
3.3	Serializing process of parallel stream . . . . .	23
3.4	Television horizontal line data by SMPTE . . . . .	24
3.5	Simplified spatial layout of a digital frame . . . . .	25
3.6	Hypothetical SDI source mapped onto ST 2022 vs ST 2110 flows . . . . .	27
3.7	Ancillary data packet semantics . . . . .	29

3.8	Illustration of MPEG-TS with two hypothetical services . . . . .	31
3.9	MPEG-TS packet structure . . . . .	32
3.10	Typical advertisement break example . . . . .	33
3.11	SCTE-35 vs SCTE-104 schematics . . . . .	34
3.12	Semantics of <i>splice_insert()</i> command . . . . .	37
4.1	Illustration of separate media essences . . . . .	39
4.2	Illustration of host engine and ANCI injection system . . . . .	41
4.3	Illustration of replica engine and ANCI reading system . . . . .	41
4.4	Illustration of a domain and its resources . . . . .	42
4.5	Illustration of steps until injection . . . . .	45
4.6	Ancillary embedder/de-embedder electronic card . . . . .	46
4.7	Injection workflow in one of the multilateral feeds . . . . .	48
4.8	Reception of feeds package at rights holder premises . . . . .	49
4.9	Proposal of simultaneous graphics generations for distinct channels . . . . .	50
5.1	Experimental transmission workflow for both feeds . . . . .	52
5.2	Experimental receiver workflow and dirty feed output . . . . .	53
5.3	Representation of TS packets encapsulated in IP/UDP flow . . . . .	57
5.4	Generic illustration of a typical GOP structure . . . . .	59
5.5	Adopted GOP structure for experimental streams . . . . .	59
5.6	Stream analysis showing no significant spikes on video bitrate . . . . .	61
5.7	NodeCG bundles architecture and code organization . . . . .	62
5.8	Picture showing two stream generators and the GFX recovery system . . . . .	66
5.9	Graphic interface of the engine, showing control panels and output . . . . .	67
5.10	Engine triggering an indicator corresponding to the command received . . . . .	68
5.11	Tab showing templates loaded and which one is currently being displayed . . . . .	68
5.12	Database entry, accessed directly in the MongoDB platform . . . . .	69
5.13	Startlist graphic, displayed in widescreen and English . . . . .	70
5.14	Startlist graphic, displayed in widescreen and Japanese . . . . .	70
5.15	Startlist graphic, displayed in adapted format for mobiles and Japanese . . . . .	71
5.16	Athlete ID graphic, displayed in widescreen and English . . . . .	71
5.17	Athlete ID graphic, displayed in widescreen and Japanese . . . . .	72

5.18 Athlete ID graphic, displayed in adapted format for mobiles and Japanese	72
5.19 Schedule graphic, displayed in widescreen and English . . . . .	73
5.20 Schedule graphic, displayed in adapted format for mobiles and Japanese .	73

# List of Tables

3.1	Scanning systems formats adapted from SMPTE 274M [1]. . . . .	21
3.2	SDI interfaces, formats and historical context. . . . .	26

# Chapter 1

## Introduction

### 1.1 Motivation

Back on the early ages of modern mass communication, TV emerged as a groundbreaking evolution of the radio by delivering not only audio but black and white pictures to its audience. Despite the novelty, broadcast systems were still quite complex and primitive, relying entirely on analog technology to produce and deliver creative content.

Visual experience on television was poor and limited, suffering of a lack of quality and definition for many years. This might explain why it took a while for the industry to realize the full potential TV images could offer. One of this underestimated possibilities was the usage of onscreen information and graphics elements to communicate with the public, which came out later.

First graphics for television had to contemplate several restrictions. Up to 20% of the screen was considered unusable due to the lack of focus around the screen's border, coupled with the fact that different television sets would cut the picture off at different points. In order to create a simple lettering, large and white printed characters had to be placed onto a black card, copying techniques seen in cinema. Credit rolls were made with special devices which used long strips of black material onto which the letters were stuck and manually rolled. One or more cameras had to be allocated to record these visual elements in a very archaic manner [2].



Figure 1.1: In the early ages, graphics elements had to be printed and filmed.

Technology evolved over the years, expanding the horizons of coverage opportunities and boosting new techniques. Colors were added to transmissions, electronic devices got more advanced and computers were later being incorporated into the production chain. Changes happened on the way people would consume TV content, driving the rise of news and live broadcasting popularity.

The audience's tolerance and desire for a broader visual language and greater amounts of onscreen information has also increased. Graphics evolved to the point where they would play a central communication role. Think of modern broadcasting of sports events: in many occasions, on-screen elements are the main tool for pointing scores, winners and statistics. Dedicated teams of designers and operators might be allocated for building templates and integrating them to complex data feeds.

Another aspect to highlight is the importance of characters for people with any degree of hearing impairment. According to recent estimates, over 1.5 billion people globally live with hearing loss [3]. Along with captioning, the resource offers the basics for comprehending onscreen action, improving the experience of this audience. Same principle applies for the audience following TV shows at restaurants or similar public spaces, where the equipment is usually found muted. Brian Douglas said: "As a director, I always assume that the audience is watching from a noisy bar and ultimately unable to hear the commentators" [4].



Figure 1.2: 1991 Grand Prix of Interlagos, Brazil. Historical race of Aryton Senna, broadcasted by TV Globo. Winner graphics stamped on footage.

When we look at TV productions today, there is very little that does not require some amount of reading from the screen [4]. Graphics generation turned more sophisticated, following the rise of computational power and design tools. Some aspects, however, were kept untouched from legacy systems. One of these premises is that most characters generated live are rendered and superimposed directly onto the background footage. This process is often performed at the initial stages of the broadcast chain, implying that the material will be “dirty” at every subsequent step. Once inserted, these visual elements cannot be removed or easily modified, emphasizing the lack of flexibility of this topology.

The project will propose an alternative method for representing data and visual elements, turning graphics into a set of metadata to be carried along with the clean background video content. This give broadcasters more control on how the process of rendering the art pieces will be performed, allowing also data content to be modified. Special attention will be paid to live sports events on TV.

The model to be presented applies to both modern and traditional TV production environments, with internal distribution infrastructure relying mainly on the SDI video interfaces and its latest versions. No specific equipment or suppliers will be recommended, although they might get mentioned across the text.



A software prototype will be developed for testing purposes, using offline media files to emulate a transmission. These tests are limited to a partial and controlled representation of a broadcast environment, simulating the behavior of key devices indicated in the project. Topics like performance and code optimization are out of scope. Databases and additional cloud resources employed are generic, not sticking to any particular technology or protocol. Aspects such as security, access control or concurrency will not be addressed.

## **1.2 Organization**

Chapter 2 focus on the basics of a TV production workflow, allowing the reader to understand some key aspects of the proposal and how graphics are generated.

Chapter 3 introduces technical details and specifications of broadcast systems and protocols used by the industry.

Chapter 4 is dedicated to the novel proposal. The structure of ancillary data packets and production chain is formulated.

Chapter 5 describes the software simulation, giving a better understanding of the system functionality.

Chapter 6 presents the conclusions and proposals for future works.

# Chapter 2

## Production Workflow

### 2.1 Broadcast stages

Before jumping into the actual graphics generation workflow, some aspects of the underlying technical infrastructure of a broadcast network must be highlighted, specially how signals are carried throughout the facilities. In general, media can be generated, processed and moved across three main areas: a real-time infrastructure, file-based systems and, finally, a transport & distribution layer. An intermediary routing stage interconnects these environments, as illustrated in figure 2.1.

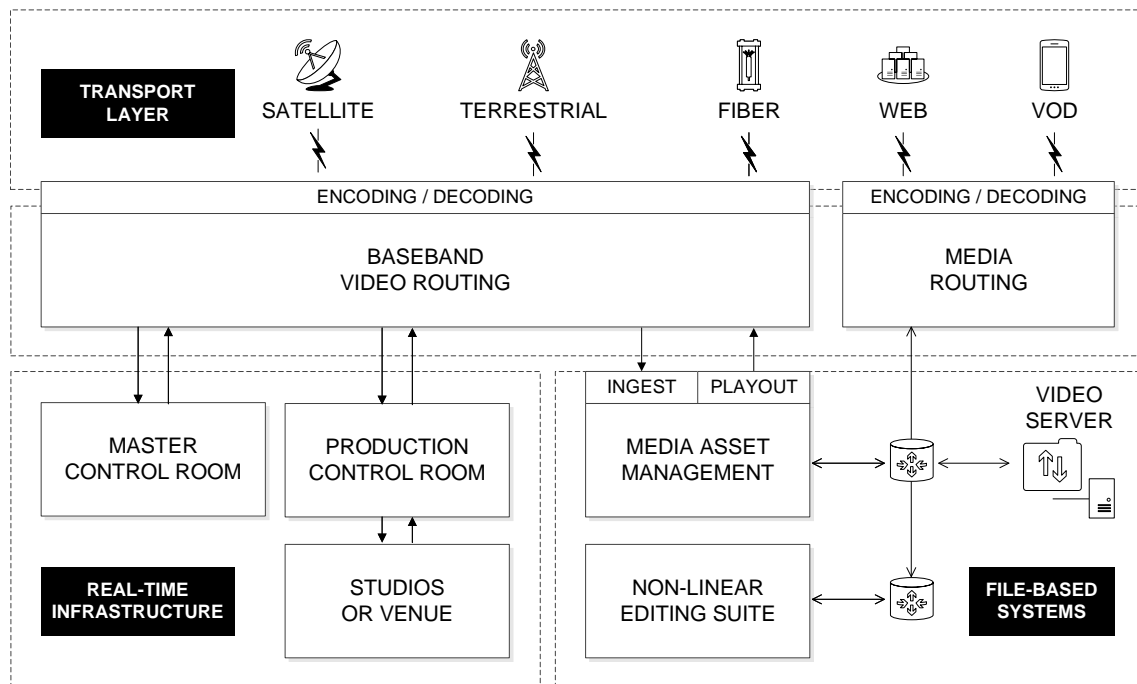


Figure 2.1: Overview of main areas within a traditional media facility.

### 2.1.1 Real-time infrastructure

Within a professional broadcast complex, the real-time infrastructure is the base for transmission of uncompressed, unencrypted digital audio and video signals. These flows have particular requirements when it comes to quality, performance, latency and tolerance for outages.

For decades, the real-time infrastructure has been relying on the SDI interface, created back in the late 80's. It uses coaxial cabling to deliver signals in a point-to-point, one-way direction. With the evolution of picture formats and higher resolutions, SDI standards have been updated over the time to accommodate the increasing bandwidths, but are rapidly becoming legacy and giving space to modern IP-based interfaces. Most broadcasters are currently operating hybrid systems, with characteristics inherited from the SDI era — as detailed on chapter 3.

Going back to the production stages, one of the most important steps in the workflow is capturing the action: either happening in a studio or a venue. These sites are connected to a *production control room* — a gallery where several input and output feeds are exchanged through the real-time infrastructure, as shown below.



Figure 2.2: Production gallery. Highlight to graphics operator (first row), assistant, vision mixer and TV director in the back.

The control room is the heart of production, specially when live content is being aired. Material is edited and monitored *on-the-fly* by a set of special broadcast equipment. The vision mixer is used for cutting between many different sources such as cameras, servers and external feeds, creating the main video feed of the show. Additional effects may be added — including graphics integrated with rich data streams.

The output signal of a control room, usually known as the *Program Feed* (PGM), is routed through the real-time distribution infrastructure for the next step in production chain, which can be a master control room, a recording server or a encoder for transmission. Notice that this video feed usually contain graphics superimposed to the images, being sometimes referred to as the *dirty feed*.

## 2.1.2 File-based systems

In a not-so-recent past, media management inside a broadcast center used to be entirely dependent on the recording of video tapes and discs. With the evolution of technology and larger storage capacity, this topology has been suppressed and shifted to modern file-based systems. This digital platform serves as a hub for content ingestion and distribution across the production pipeline, often sorted in centralized databases by *Media Asset Management* tools (MAM).

Within this workflow, media is shared as files over the traditional IT network, easily carried in data packets throughout multiple servers — boosting efficiency. Inside editing suites, for example, previously recorded footage can be captured, reviewed, and treated by a wide range of software tools. Graphics may also be inserted at this stage, mixed to the video. The output is usually a new rendered file ingested back into the storage servers.

Material may be originated from many different sources, including external recordings, satellite receivers or any other feed routed through the traditional real-time contribution infrastructure. In that case, hardware and software tools serve a compatibility layer, integrating media carried over electro-optical interfaces into the file-base domain and vice-versa.

Ingested media is usually transcoded to a common file format, in accordance to broadcast parameters set by the station, assuring proper interoperability amid equipment. Codecs and containers may differ greatly and the urge for smoothest workflows has lead the invention of formats to promote efficiency [5].

Back in the 1990s, engineers representing a wide number of users and manufacturers united to develop an open file format to facilitate the interchange of audio, video, and associated metadata within file-based workflows. This initiative eventually led to development of the SMPTE-approved *Media Exchange Format*, in 2004 [6].

When initially designed, MXF had a number of fundamental design requirements: it had to be open, standardized and compression-format independent. Most of all: simple and flexible, ideally applicable to a large variety of workflows, carrying faithfully metadata and media essences throughout the life cycle of a program, movie or clip [6]. *De facto*, it became one of the most popular formats for media assets.

### 2.1.3 Transport layer

Once material has been treated and properly edited — in a control room or editing station —, next natural step is preparing it for transmission, either to another broadcast center, an affiliate or the final audience. Ways of distributing feeds may involve usage of fiber, satellite, microwave or internet links.

Media to be transferred is normally compressed in an efficient way, bearing in mind possible bandwidth restrictions. This process is usually performed by FPGA-based hardware, in conformity with well established specifications. Common video compression standards adopted by the broadcast industry are H.264 (AVC), H.265 (HEVC) and, lately, JPEG XS.

Along with the video, audio and any additional metadata must be finally encoded. A *container* encapsulates elementary streams in a digital structure, applying proper synchronization patterns and error correction codes to segmented data. This ensures a certain degree of robustness and reliability, specially when the communication channel carrying the stream is unstable or degraded.

One of the most popular containers adopted is the *MPEG Transport Stream* (MPEG-TS), carried over an ASI interface, IP ASI, or TCP/UDP. Chapter 3 will present the basics of the standard, focusing on its inner structure. Other distribution protocols may include a SRT link on open internet, MPEG-DASH or HLS for streaming platforms.

On receiving side, each sub-streams should be unwrapped into their audio, video and data elements. Processing is often performed to each essence independently, and content is later reassembled and synchronized based on timing stamps sent along with the media.

## 2.1.4 Overview

As demonstrated, the real-time infrastructure, file-based and transport systems form the technical pillars supporting any regular media production. Some broadcast business models may be content-oriented or constrained to particular technologies, demanding more from one of these platforms than the others.

As this work will be fundamentally focused on *live* sports productions, emphasis will be given to aspects of the traffic between real-time and transport infrastructures. These processes are supported by dedicated broadcast hardware and software, as illustrated below.

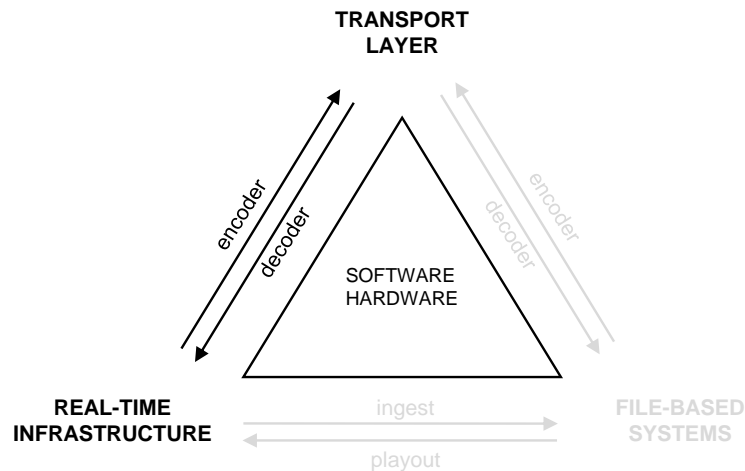


Figure 2.3: Major technology pillars of production.

## 2.2 Graphics

### 2.2.1 Types of graphics

First aspect to observe in the graphics production workflow is the difference between their types and main purposes. They are often generated in different ways, despite sharing many elements and visual characteristics. Three basic categories can be mentioned: *illustrative graphics*, *virtual enhancements* and *overlays*.

- **Illustrative Graphics** — Created by designers and animators based on editorial requests, usually covering specific needs. May include static or dynamic backgrounds, logos, maps, introduction clips, promotional material and similar. Are often rendered *in advance* by an editing station, recorded, and later displayed on over-the-shoulder screens at the studio or on-air.



Figure 2.4: Illustrative graphics examples. Tokyo 2020 coverage, TV Globo.

- **Virtual Enhancements** — These compositions are directly associated to specific camera shots, usually integrated with tracking systems. A 3D mapping of the scene allows the graphics to follow camera movements, creating an immersive perspective. Very common at sports transmissions.

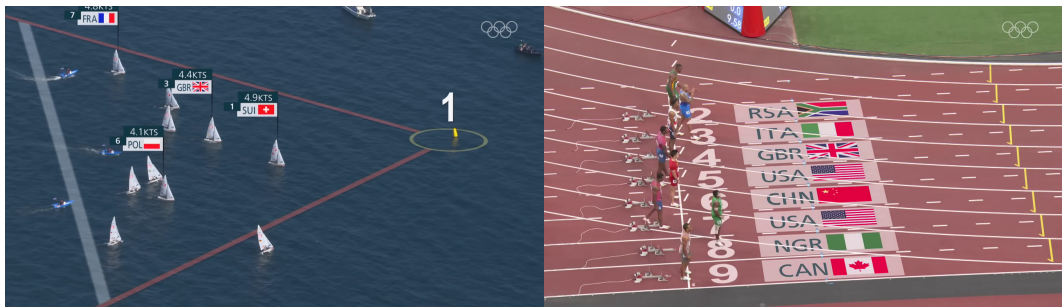


Figure 2.5: Virtual graphics examples. Tokyo 2020 and Rio 2016, IOC/OBS.

- **Overlays** — Also called *text graphics* [4], the main purpose is transmitting information. May contain 2D or 3D visual elements associated with data content. They are superimposed to background image and some examples includes: scoreboards, lower-thirds with names, crawls and results graphics.

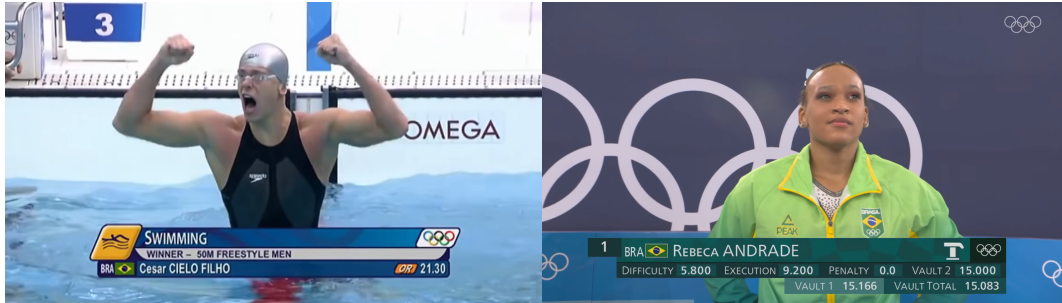


Figure 2.6: Overlays examples. Beijing 2008 and Tokyo 2020, IOC/OBS.

This project will be fundamentally focusing on graphics *overlays*, specifically those generated live in a control room. An overview of the process will be presented below.

## 2.2.2 Graphics engine

*Character Generators* (CG's) are responsible for creating the animated text and graphics content to be mixed onto the footage. They can be hardware or software based, but this distinction is becoming less and less obvious since all modern graphics engines are build on top of computerized platforms.

Hardware character generators, such those found in any regular control room, encode the visual composition into a high-quality video signal with the assistance of interface cards. Originally, these signals were not capable of carrying *opacity* information along with luminance and crominance components. In order to overcome this limitation, the engine traditionally supplies a secondary output with the alpha channel — a gray-scale alike mask indicating transparency levels.

Both CG outputs, *fill* and *key* are delivered to a *Downstream Keyer* (DSK), the system that finally blends the graphics to the background video, usually through a linear operation. Figures 2.7 and 2.8 exemplifies the process. The keyer may also be incorporated in the vision mixer.



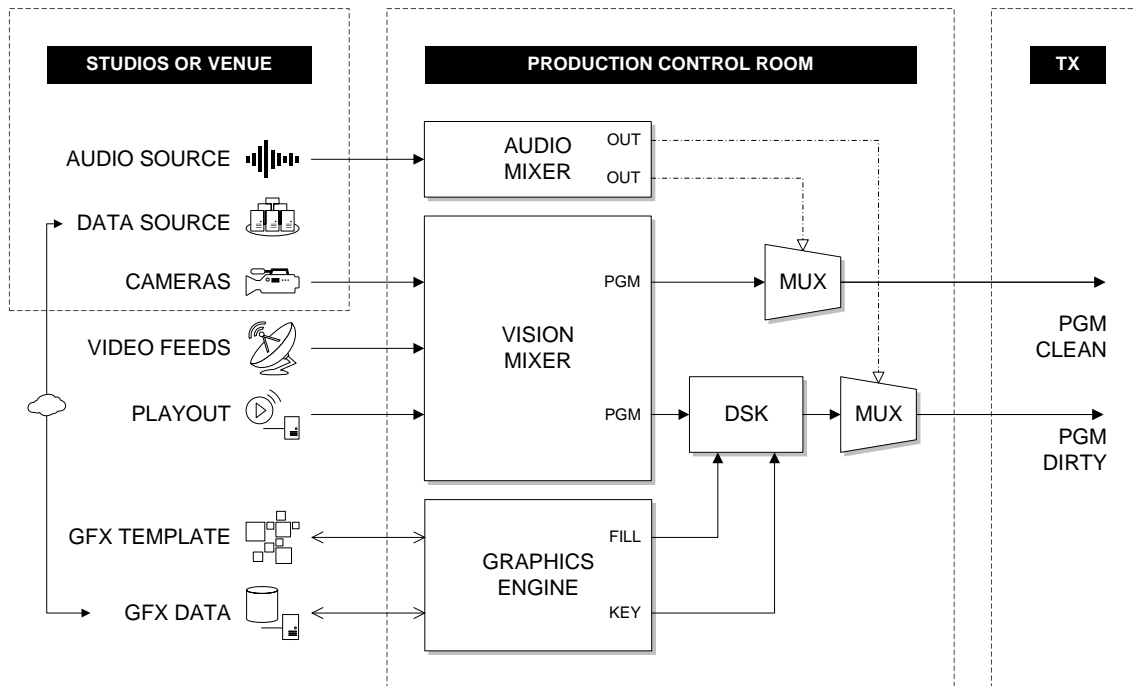


Figure 2.7: Highlight of graphics generation in a gallery for linear TV.

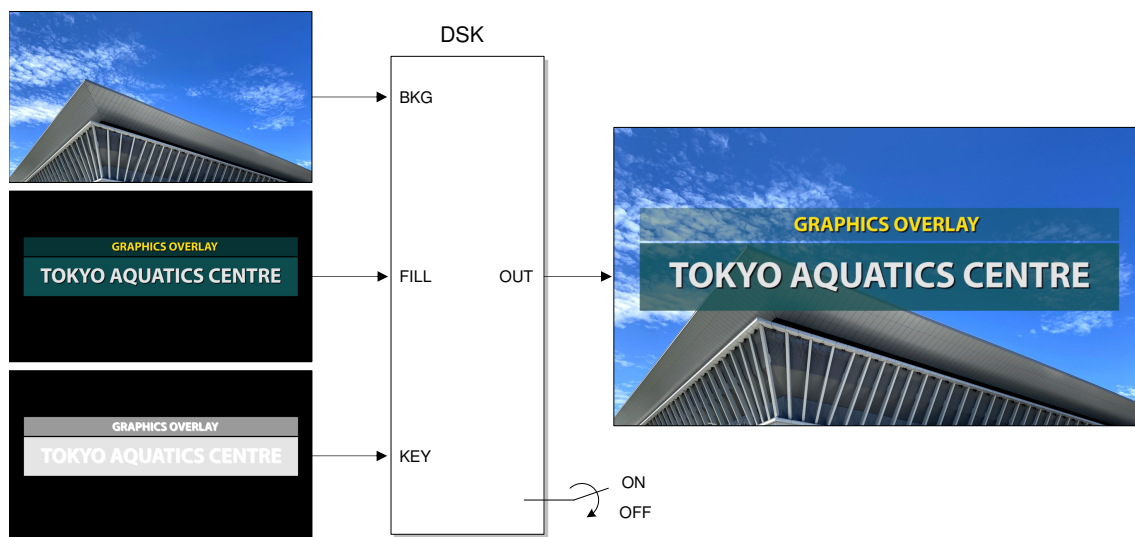


Figure 2.8: Illustration of linear keying process.

Virtually, any data source can be interfaced with the engine, either through serial ports or the network. This information can be used to populate the templates and may come from web API's, databases, timers, sensors, scoring devices, among others.

Common delivery formats for data are CSV, JSON or XML files. The graphics application is built with the appropriate formulas to map those into a schematic and the final rendered message, as shown in figure 2.9.

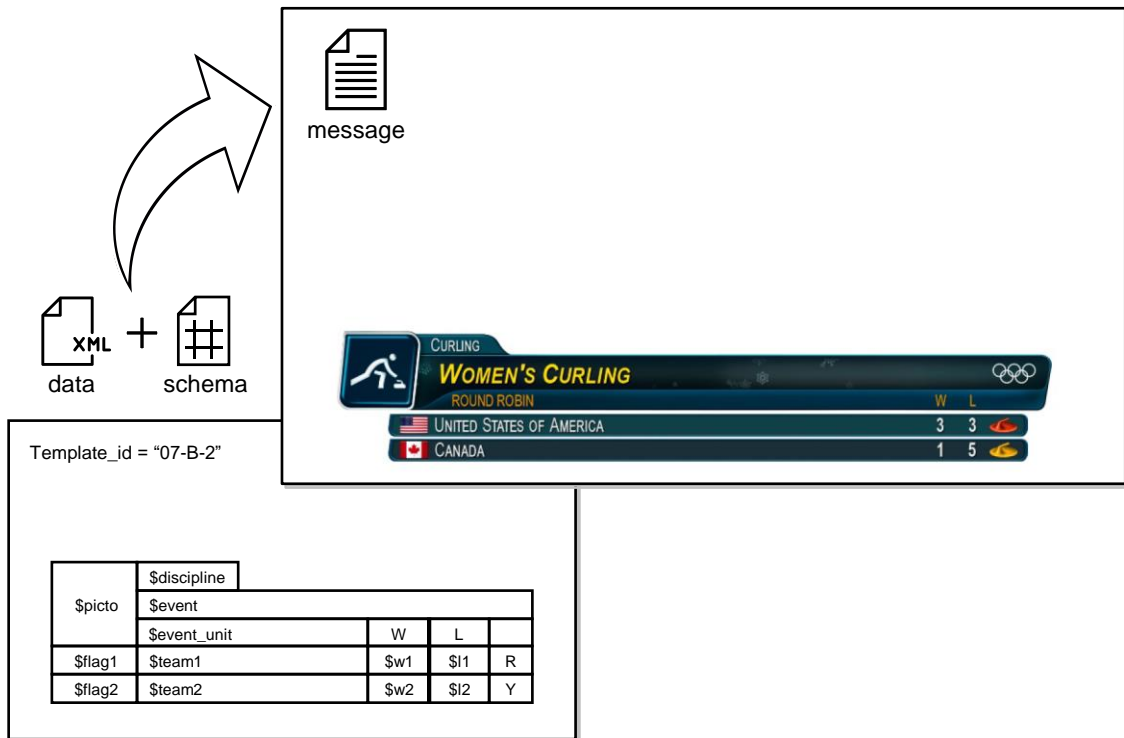


Figure 2.9: Illustration of data fields being mapped into a graphic template.

Another aspect to highlight is the capacity graphics assets may have of accepting *commands* carried in a control channel. These are scripted instructions for performing certain tasks, usually *in* and *out* animations, but can include other sophisticated functions.

The system can run with a certain degree of automation, dispatching commands and animations when triggered externally (through network, GPI, etc) or manually. An operator is usually supervising and properly inserting the arts when commanded by the director, following on-screen action and the desired storytelling. Some template fields may also require manual typing or a few corrections before going on-air.

One of the trending technologies in broadcasting is the usage of web-based applications for graphics rendering. Versatile and sophisticated packages can be build on top of HTML5, CSS and JavaScript — tools easily interpreted by any browser. Controlling can be performed in a page with web-sockets and plenty of tools can be used to overlay the content onto the video. This sort of system is becoming extremely popular in streaming platforms.

## 2.3 Case study

### 2.3.1 Major multi-feed events

Consider now the scenario of an international transmission event — as the *Olympic Games* or the *Football World Cup*. Given its size and operational complexity, these sports competitions emerge as exceptional study objects in the context of this project. Deliberations to be made here can later be extended to similar or less intricate broadcast productions.

Large scale events like these usually count with the figure of a *Host Broadcaster* — an organization responsible for capturing, producing and distributing high quality audiovisual content to *Media Rights Holders* (MRHs) around the world. Without this entity, each network interested in broadcasting such event would have to mobilize its own resources for the transmission — which would not be viable.

These competitions usually take place at multiple fields-of-play, geographically distributed in a region of the world. The most common operational agreement usually states that the Host Broadcaster must prepare a local broadcast infrastructure at each venue, in order to guarantee the best coverage for the global audience. Hundreds of technological resources are deployed and usually operated at the *local production unit*, where are also concentrated the signals for transmission.

The main goal of the local production unit is generating the international program, also called *multi-lateral feed* — ready to be distributed to broadcasters. Due to its nature, this feed usually is dirty with graphics, fed by real-time data coming from the-field-of play. Content is commonly rendered in English — with the intention of making it as much universal as possible.

Figure 2.10 exemplifies the process. First stage of the transmission chain is a local distribution of the dirty feed to rights holders with presence at the venue. This is usually the case when there is a particular interest of a network on specific events, specially when they have intention of enriching their transmission with *unilateral* contributions — exclusive cameras or other feeds of that broadcaster.

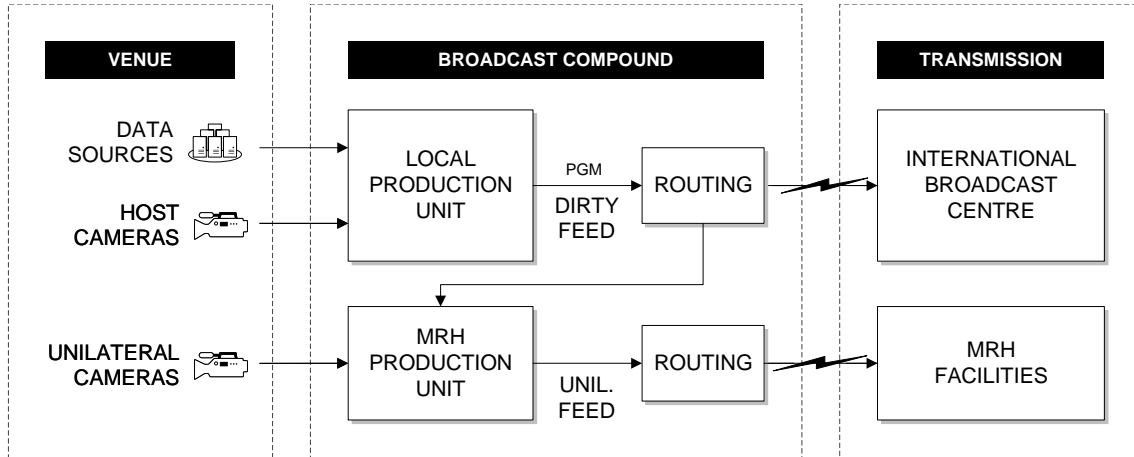


Figure 2.10: First stage of distribution at the venue.

Next step of the process consists of carrying the multi-lateral signals to a central facility, which acts like a hub where further processing will take place. At the *International Broadcast Center* (IBC), the feed of all competition venues is routed and distributed across several systems attending many different purposes, including digital archiving.

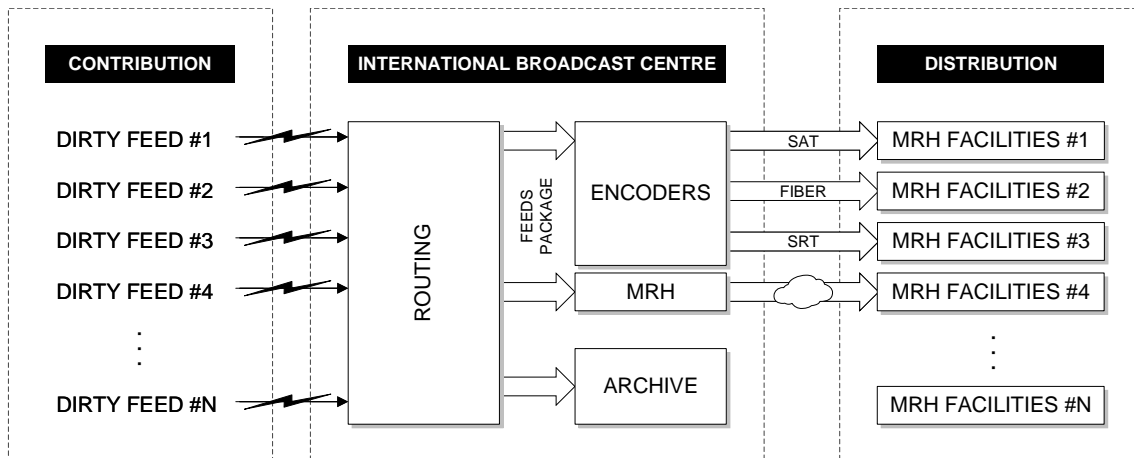


Figure 2.11: Distribution from venues to IBC and MRHs.

Vast majority of right holders chooses to have presence at the IBC, where is possible to have direct access to all multi-lateral feeds. They can build and operate their own smaller broadcast infrastructure, mainly to carry content from the IBC to their headquarters in the way they might judge appropriate. Networks with no presence at the IBC may still have access to feeds through fiber or satellite, usually made available at points of presence spread across the globe.

Once material is under MRHs domains, is up to them to perform the local distribution on their platforms. In the past, traditional TV was the major (or some times, the only) distribution channel of these contents. Transmissions were exclusively planned for this format, which used to dictate all major technical and editorial aspects. This scenario is drastically shifting with the consolidation of mobiles and other devices for media consumption.

### 2.3.2 Topology drawbacks

As detailed in previous section, the distribution chain of such events works from one end to another with the dirty feed, composed of audio and video essences in sync to each-other. Both can be processed separately by their own specialized broadcast systems and tools, but later regrouped. Observe that graphics are directly mixed in the program feed, and, therefore, follows the video path across the network.

A preliminary reflection can be made on this topic: evaluating a graphic and the associated background video content, it becomes clear that they have distinct natures, despite the visual appeal. Graphics are digitally build, made of static or dynamic elements and gradients that can easily make use of the entire color palette. On the other hand, the image of a camera, for example, is captured with organic movements and may have a slightly restricted luminance and crominance ranges.

Most of video codecs currently in use are optimized to explore characteristics and redundancies found on camera footage. This means that a codec may perform poorly with graphics components, resulting in digital artifacts or distortions when the video content is compressed — which certainly will happen at a certain point, possibly more than once. This drawback gets in evidence during the archiving process, when feeds have to be transcoded and stored in limited-size media.

Another negative aspect to highlight had already been mentioned, which is the fact that archived and distributed material will be indefinitely *dirty* with rendered graphics elements. In several occasions, access to *clean* video content is highly desired, given the flexibility it admits for usage in novel applications.

Figure 2.12 exemplifies such process, showing content originally broadcasted being adapted to social media posts, consumed in smartphones. Images displayed on TV usually have an *aspect ratio* (proportion between its width to its height) of 16:9. In contrast, mobile devices may operate with all sorts of screen sizes and proportions, either in portrait or landscape modes. In order to adapt to 4:5 aspect of the publication, the original video material had to be cropped, leading to unreadable and dysfunctional graphics on-screen.

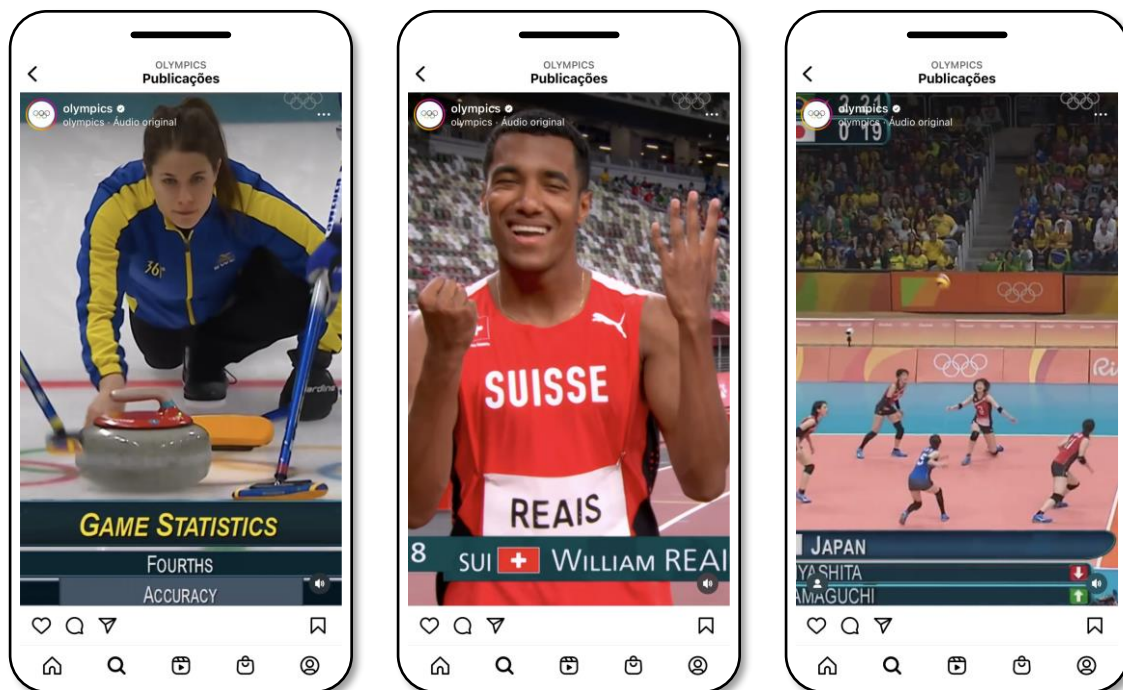


Figure 2.12: Cropped graphic elements on smartphones. Images from IOC/OBS.



Figure 2.13: Most TV graphics are not adapted for mobiles. Fonts may look small and elements covered by user's hand. Image from Pyeongchang 2018, IOC/OBS.

Now consider the scenario of some wrong data being incorrectly inserted on a graphics template and going on-air, either because of system faults or inadvertent human action. In the case of a sports competition, in which precision with names, timing and numeric values are critical, the mistake might have a considerable impact on the audience. Besides, it will be inadequately stamped in recorded material, unless a non-linear video edition is later performed to amend or remove the content.

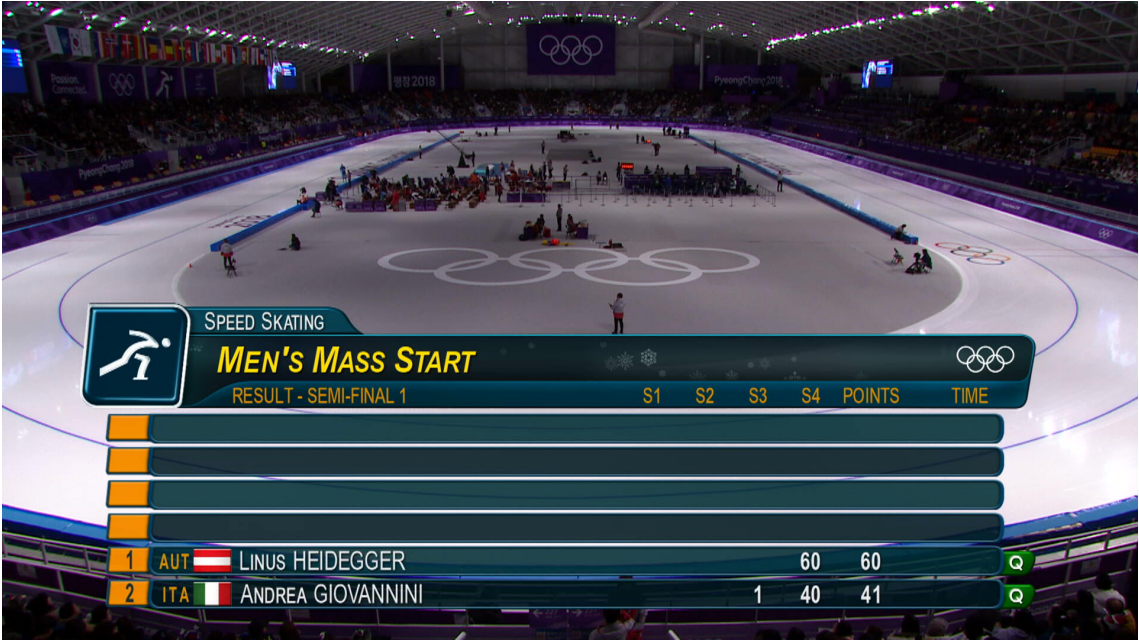


Figure 2.14: Graphics with defective data, as recorded. Pyeongchang 2018, IOC/OBS.

Previous example emphasizes another major deficiency: it is not possible to re-process or rebuild graphics templates and contents *on-the-fly* over the distribution path. Back to large events scenario, imagine that rights holders are willing to offer translated graphics for their audiences. Current alternatives to make this process viable are costly and scarce, often involving access to clean-feeds and a prohibitive replication of the graphics infrastructure for each desired new language.

A solution adopted by some broadcasters to overcome this limitation consists of superimposing translated graphics onto the dirty feed, generally using a similar visual package. An operator has to manually dispatch these new messages every time a graphic is displayed. The delay of this non-automated action is quite visible and the result prone to errors.



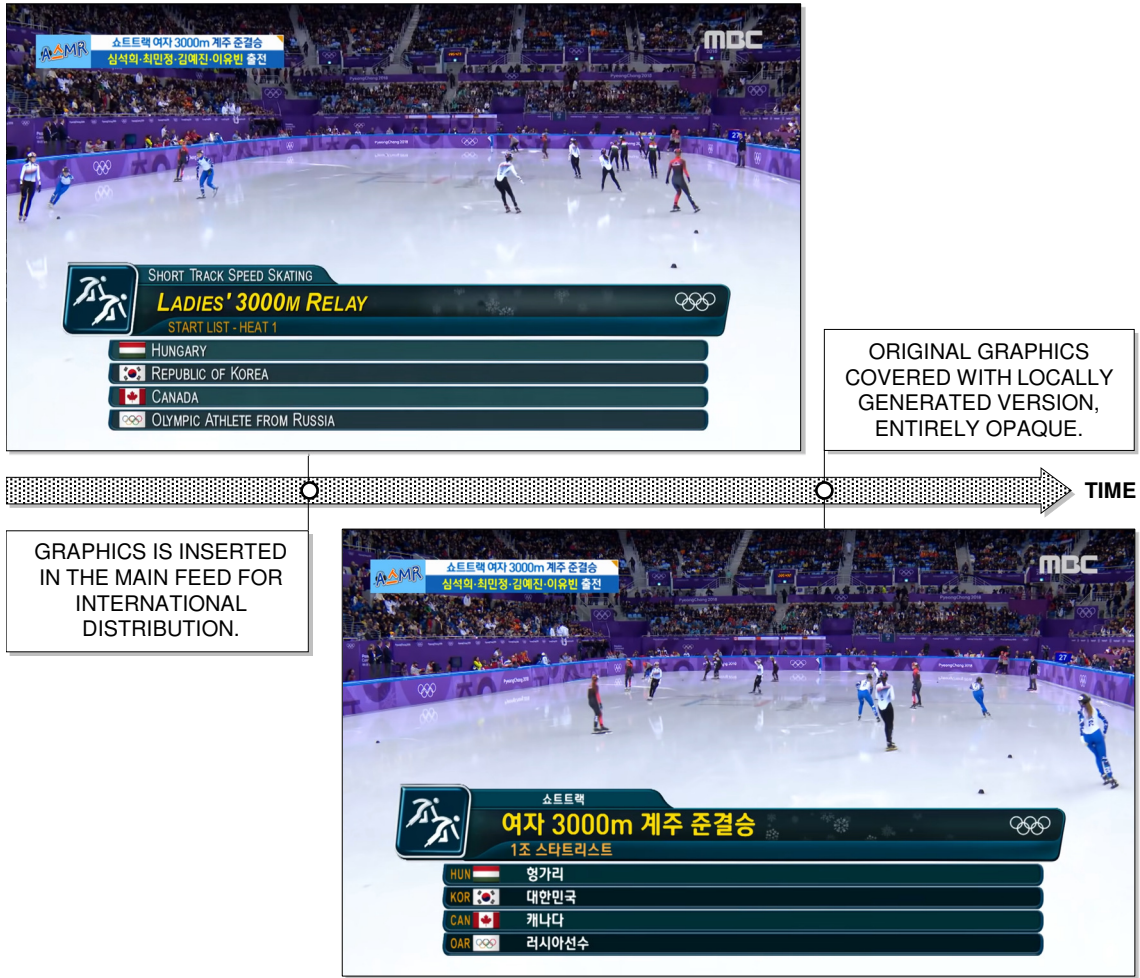


Figure 2.15: Translated graphic overlay from korean broadcaster MBC.

### 2.3.3 Overview

Above mentioned disadvantages could be mitigated or solved avoiding overlays to be directly rendered onto video essence. Instead, if graphics could be represented as a set of raw metadata, one potential solution would be carrying these information across the distribution path for subsequent reprocessing in a flexible manner. It is essential that this additional metadata flow is synchronized with audio and video.

A feasible way of guaranteeing integrity and synchronicity of this data is encapsulating it directly with media tracks, to be transported within digital interfaces. This is when the *ancillary data space* comes up as a plausible alternative to be considered, well established and popular in the industry. Next sections are going to dive into details of ancillary data carriage, focusing on how it relates with broadcast systems and protocols currently in use.



# Chapter 3

## Standards Review

### 3.1 SDI interface

The *Serial Digital Interface* is a professional-grade family of specifications for audio, video and data transmission over coaxial cables or fiber. Created by the SMPTE back on the late 80's, it marked a revolutionary transition from analog to digital era. SDI has a point-to-point topology, used to connect all sorts of broadcast equipment such as cameras, routers, playouts, video mixers, processors, encoders, among others.



Figure 3.1: Coaxial SDI cable terminated with BNC connector.

Carrying uncompressed base-band video, SDI has proven to be incredibly reliable, being widely adopted and upgraded over the years, embracing the modernization of picture formats and ever increasing bandwidths. First large technology jump occurred in the transition from SD to HD, with transfer-rates multiplied by five.

At later stages, support for progressive scan modes have been introduced and, ultimately, UHD formats are pushing the standards to the edge.

One important aspect to observe is the fact that even the most modern updates of SDI kept legacy attributes from previous releases, given the industry needs for backwards compatibility. In fact, multiple features present in the standard are actually inherited from analog television times, as will become clearer later on.

For the purpose of summarizing its architecture, next sections are going to focus on key elements of the HD-SDI specification, as defined by SMPTE ST-292 [7]. This is an interface with nominal rate up to 1.5 Gbps and support for several video source formats, including popular high definition television systems such those referenced in ANSI/SMPTE 274M [1] and presented below.

Table 3.1: Scanning systems formats adapted from SMPTE 274M [1].

	<b>System Nomenclature</b>	<b>Active / total samples per line</b>	<b>Active / total lines per frame</b>	<b>Frame rate (Hz)</b>	<b>Interface sampling frequency (MHz)</b>
1	1920×1080/60i	1920 / 2200	1080 / 1125	30	74.25
2	1920×1080/59.94i	1920 / 2200	1080 / 1125	$\frac{30}{1.001}$	$\frac{74.25}{1.001}$
3	1920×1080/50i	1920 / 2640	1080 / 1125	25	74.25
4	1920×1080/30p	1920 / 2200	1080 / 1125	30	74.25
5	1920×1080/29.97p	1920 / 2200	1080 / 1125	$\frac{30}{1.001}$	$\frac{74.25}{1.001}$
6	1920×1080/25p	1920 / 2640	1080 / 1125	25	74.25
7	1920×1080/24p	1920 / 2750	1080 / 1125	24	74.25
8	1920×1080/23.98p	1920 / 2750	1080 / 1125	$\frac{24}{1.001}$	$\frac{74.25}{1.001}$

Letter “i” in system nomenclature indicates an *interlaced* format, while letter “p” stands for *progressive* scan mode. The factor 1.001 was kept as a legacy from NTSC system, which required this adjustment to match analog audio and video carriers.

Source format shown on first line has the highest bandwidth consumption, and therefore will be used to illustrate next sections. A 1920x1080/60i system has video resolution of 1920 horizontal samples and 1080 active lines, aspect ratio of 16:9, square pixels, interlaced scan and frame rate of 30 Hz.

Not only aspects of the data structure are detailed by the standard, but also opto-electrical parameters for signal exchange, such as peak-to-peak amplitudes and jitter — which are not going to be covered here. The basics of inner video structure and the serializing process are demonstrated below.

### 3.1.1 Video components

Imagine a video source, like a camera, generating its *gamma-corrected* components red, green and blue (represented as  $R'$   $G'$   $B'$ ), each occupying a separate transmission channel. Despite the simplicity, this color representation is not the most efficient in terms of bandwidth optimization, often being replaced by an alternative professional method.

Due the fact that the human vision is more sensitive to changes in brightness than to changes in color, the original video components can be derived into *luminance* and *chrominance* portions, and distinct transmission rates for each can be adopted. The new color space is defined by a mathematical coordinate transformation, as depicted in figure 3.2.

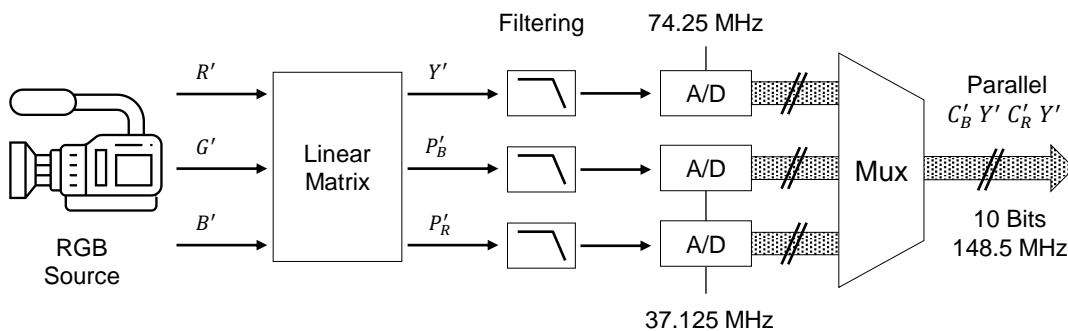


Figure 3.2: Color-space change and digitizing process of a source. Adapted from [8].

By having one full bandwidth luminance channel ( $Y'$ ) to represent brightness and details of the signal, the two other color-difference components ( $P'_B$  and  $P'_R$ ) can be limited to about half the luma channel bandwidth and still provide sufficient color information [8] — a process referred to as *chroma subsampling*.

Luminance and chrominance components are then filtered, sampled and digitized, respecting parameters detailed by the SDI normative. For 1920x1080/60i system, luma is sampled at 74.25 MHz and chrominance at half the rate, 37.125 MHz, resulting in  $Y'$ ,  $C'_B$  and  $C'_R$  channels. Each word has a 10-bit resolution and gets multiplexed in a parallel stream of 148.5 Mwords/s.

Data enters a coprocessor, which may insert *ancillary data* at designated portions of stream not used by the active video. Then, a cyclic error checking mechanism adds another set of information for each video line and, finally, the signal is serialized for transmission. The channel coding ensures that there are sufficient edges for reliable clock recovery and scrambling minimize low frequency content. The SDI output has NRZI formatting and a total of 1.485 Gbps for this source format.

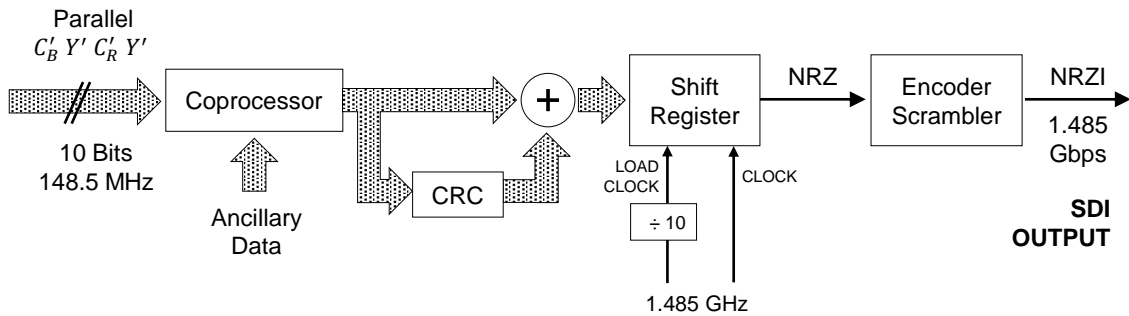


Figure 3.3: Serializing process of parallel stream. Adapted from [8].

### 3.1.2 Data format

In accordance to the SMPTE specifications, each video line should be structurally divided into four areas: *Start of Active Video* (SAV), *End of Active Video* (EAV), *Digital Line Blanking* and *Digital Active Line*.

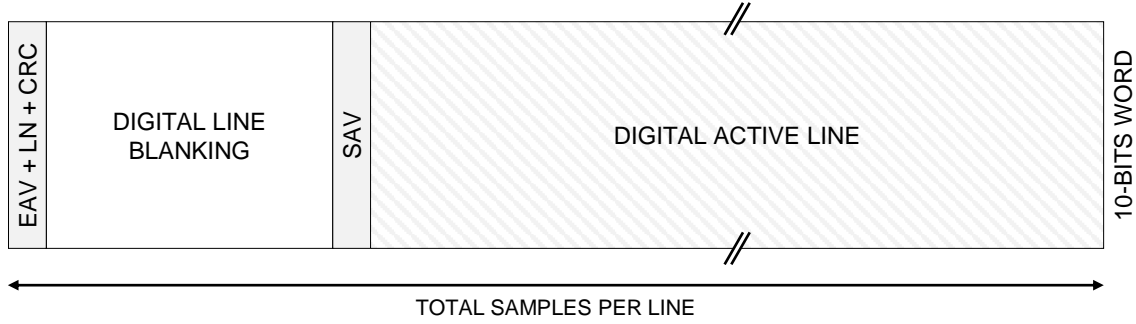


Figure 3.4: Television horizontal line data by SMPTE (not in scale). Adapted from [8].

- **EAV** — Space reserved for synchronization and other special words, including a line number and error checking. As the name suggests, it indicates the end of active picture region and has a total of 16 words for this source format.
- **Digital Line Blanking** — Back to the analog TV period, image was built on CRT displays by scanning an electron beam from left to right across the screen. An interval was needed to move the beam for the next line to raster, originating this legacy blank region.
- **SAV** — Area that also contains reserved synchronization packets used to demarcate the start of the digital active line. Composed of 8 words.
- **Active Picture** — Space for the actual transmission of luminance and chrominance code-words. In a high-definition video, the total 1920 horizontal pixels are translated into 1920 samples of luminance and 1920 samples of chrominance ( $C'_B + C'_R$ ), a total of 3840 words.

The standard for the 1920x1080/60i format specifies that a total of 1125 lines as described above should be present to form the signal. The additional lines form another blanking region over the active picture, as shown in figure 3.5. Another aspect inherited from analog era, this interval was necessary to accommodate the slow return of the electron beam from the bottom to the top of CRT screens. In a digital domain, both blanking intervals form a non-picture area and can be used for transmitting supplementary data. These regions are denominated as the *horizontal ancillary data space* (HANC) and the *vertical ancillary data space* (VANC).

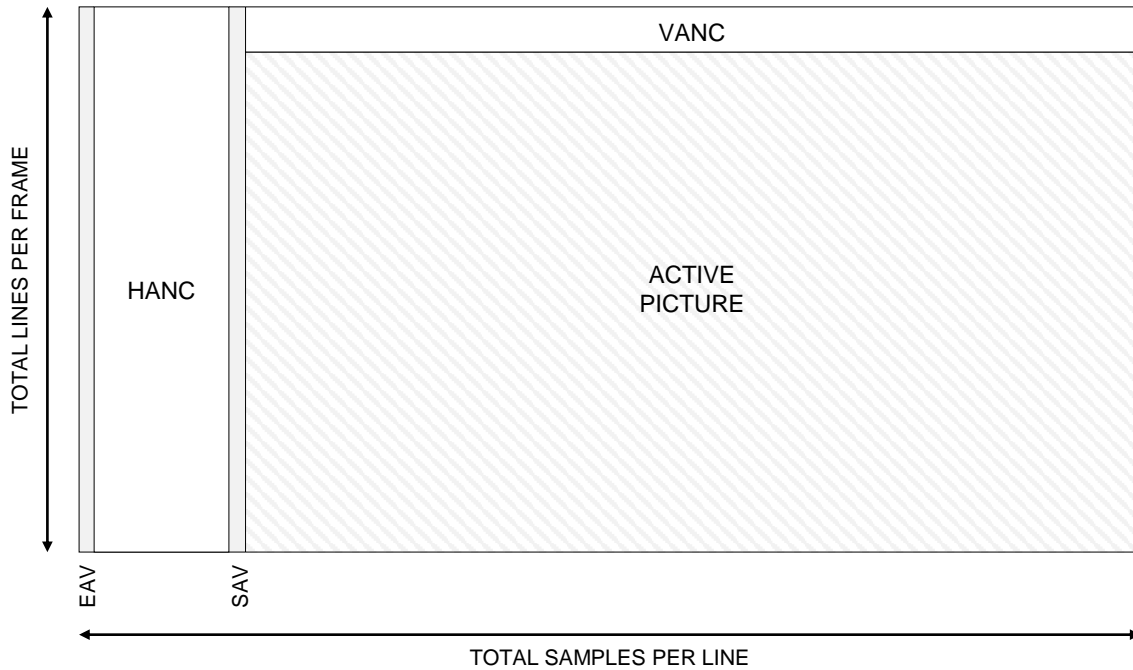


Figure 3.5: Simplified spatial layout of a digital frame (not in scale). Adapted from [8].

### 3.1.3 SDI evolution

In order to support higher resolutions, new color spaces and frame rates, SDI has been adapted to meet the needs of the industry and fast-advancing consumer technology. Moving to progressive 1080p video meant a doubling of the bit rate from 1.485 Gbps to 2.97 Gbps, for example. Back then, chipsets and interface specifications were not available to achieve distribution over a single link of such data rates.

To address these limitations, a standard was released in 2002 admitting transport of a 1080p60 video signal over two HD-SDI links [9]. By 2006, electronics had improved enough in general, allowing a single cable to reliably deliver a 2.97 Gbps payload — originating the 3G-SDI normative. Formats continued to mature and SMPTE released their ST-2081 suite of standards to ease progression to higher frame rates, making available 6G-SDI and carriage in three versions: over single link, dual links, and quad links. Lastly, 12G-SDI was released in 2015, smoothing delivery of 4K signals over a single cable.

A summary of above mentioned standards is presented below. They all are capable of supporting seamless carriage of ancillary data within their interfaces.

Table 3.2: SDI interfaces, formats and historical context.

<b>Nomenclature</b>	<b>Standard</b>	<b>Video Resolution</b>	<b>Bitrate</b>	<b>Year</b>
HD-SDI	SMPTE 292M	1080i60	1.485 Gbps	1998
Dual Link HD-SDI	SMPTE 372M	1080p60	2.970 Gbps	2002
3G-SDI	SMPTE 424M	1080p60	2.970 Gbps	2006
6G-SDI	SMPTE ST-2081	2160p30	5.940 Gbps	2015
12G-SDI	SMPTE ST-2082	2160p60	11.88 Gbps	2015

### 3.1.4 Migration to IP

The need for increased bandwidth for transmission of ultra high definition content, better connectivity, combined with distribution of content on growing over-the-top platforms is simply outweighing the capabilities of SDI. As a consequence, there is a fast ongoing process of shifting all real-time distribution to IP-based networks. Most media facilities are currently operating hybrid systems.

These networks can run on the traditional IT ecosystems, taking advantage of internet protocols that may provide agility and flexibility for broadcasters. In terms of infrastructure, usage of IP routers and switches, for example, allows multiple bidirectional flows to be transported over a reduced number of links — unlike SDI. In the long run, this means decreased costs, improved bandwidth consumption and boosted overall efficiency [10].

One of the first normatives on this matter was introduced back in 2007. The SMPTE ST 2022 [11] suite of standards describes a reliable way of using IP to transmit signals that were traditionally sent over serial interfaces, including uncompressed video, audio, synchronization patterns and — last but not least — ancillary data packets.

A SMPTE ST 2022-6 stream can be seen as an IP version of the SDI flow. The entire SDI content is packetized for routing, including both blanking intervals — which is why ancillary data gets transparently transported along with all media content. This approach has one evident drawback, however: on receiving end, devices must read, process and de-embed the whole packet flow in order to access the individual tracks of that stream, even if only interested in one of these particular essences.

At a later stage, a global team of engineers and broadcasters gathered along with SMPTE to develop a new suite of IP standards. The collaboration efforts were focused on the definitive transition from circuit-switching SDI to a flexible, manageable IP workflow, suitable for multi-platform and mixed-consumption scenarios.

The resulting SMPTE ST 2110 standards specifies the carriage, synchronization, and description of *separate* elementary essence streams over IP for real-time production. Each stream is individually timed by the ST 2110 system and can take different routes over the network to arrive via unicast or multicast at one or more receivers [12]. This allows each component to be processed, transported, and stored separately from the others, as illustrated in figure 3.6.

The standards are split into four main documents: 2110-10 for timing and synchronization, 2110-20 for uncompressed active video, 2110-30 for PCM audio, and 2110-40 for ancillary data [13][14][15][16]. SMPTE ST 2110-40 standardizes how metadata is encapsulated, defining that contents of ancillary packets shall be directly mapped into RTP packets as specified in IETF RFC 8331 [17].

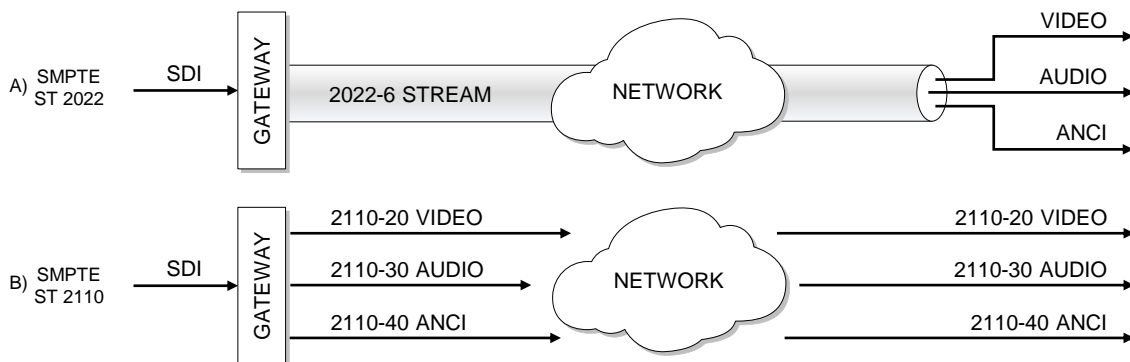


Figure 3.6: Hypothetical SDI source mapped onto ST 2022 and ST 2110 flows.



### 3.1.5 Ancillary data structure

This topic is standardized by SMPTE ST-291 [18], which defines the basic formatting structure of the ancillary data space in digital video data streams in the form of 10-bit words. Application of this standard includes, but is not limited to, high definition digital television interfaces, as the one described previously.

Space available for ancillary data packets is defined in documents detailing the connecting interfaces. Ancillary data packet payload definitions for a given application may be specified by a SMPTE *Standard*, *Recommended Practice*, *Registered Disclosure Document*, or by a document generated by a third party individual. When a payload format is registered with SMPTE, an application document describing its content is required and the data packet is identified by a registered *Data Identification* word [18].

Ancillary data packets are divided into into *Type 1* and *Type 2*, with small differences between each, as indicated in figure 5.6. The basic semantics comprises:

- **Ancillary Data Flag (ADF)** — A three-word start sequence to enable packets to be detected, having values 0x000, 0x3FF and 0x3FF.
- **Data Identifier (DID)** — Used to distinguish packets carrying a particular type of ancillary signal.
- **Secondary Data Identifier (SDID)** — Defined as a combination of data ID and a secondary data ID. Used only in Type 2, allowing a wider range of identification values.
- **Data Block Number (DBN)** — Distinguishes successive ancillary data packets with a common data identifier. Used in Type 1 only.
- **Data Count (DC)** — Number which indicates the quantity of user data words to follow.
- **User Data Words (UDW)** — The payload of the packet, with a maximum of 255 words. Defined according to the packet type.
- **Checksum (CS)** — Word to permit error detection.

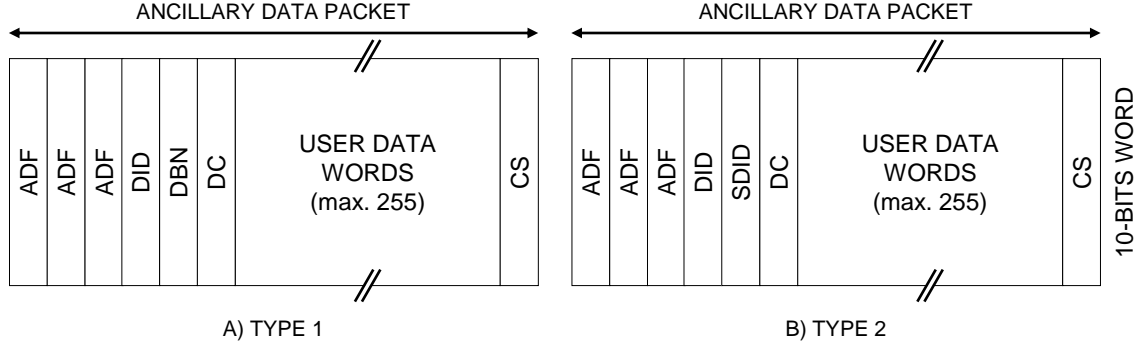


Figure 3.7: Ancillary data semantics. Type 2 packets are composed of the same elements as Type 1, except for the DBN, which is replaced by SDID. Adapted from [19].

Provision is made for ancillary data exceeding 255 words to be carried in two or more linked packets, not necessarily contiguous with each other [19]. Many different applications explore the ancillary spaces. The HANC region accommodates high-bandwidth data or services that need to be synchronized to a particular line. Examples are *embedded audio* and VPID (Video Payload Identifier). On the other hand, VANC is commonly used for low-bandwidth data, or for resources that only need be updated on a per-field or per-frame rate.

Some of these applications are intended for internal use within broadcast workflow, like *time-code*. Some are intended to propagate to end user, like *closed captions*. Some ancillary data come and go through the production pipeline, while others simply get lost. Some are time-aware, while others are temporarily bound to video essence and selectively picked by receivers.

### 3.1.6 Overview

In terms of intended usage for this project, ancillary data updated with a certain degree of frame accuracy seems enough for our proposed goal, therefore suggesting that these could be properly carried in VANC space. The semantics of user data words filling the ancillary data packets can be entirely personalized to fulfill the application needs. Metadata could, for example, be written in KLV (Key-Length-Value) format — a data encoding protocol specified in SMPTE ST 336 [20]. This level of flexibility, however, imposes a series of short and mid-term restrictions.

A custom-built implementation may require SMPTE registration and may not be supported by current installed base of broadcast devices and systems — eventually requiring additional development of firmware or software for interpreting these ancillary packets. An alternative for overcoming this constraint relies on the employment of a well-consolidated protocol that could be adjusted to the project needs. Detailed in section 3.3, ANSI/SCTE 35-104 messages arise as a feasible option.

## 3.2 MPEG-2 Transport Stream

The *MPEG Transport Stream* is a standard defined by ISO/IEC 13818-1 [21] and ITU-T Recommendation H.222.0, describing a digital container for media and data elements to be stored or transmitted live. It is widely adopted by the industry, specially when sending encoded and compressed streams between facilities, over internet, satellite, or terrestrial broadcasting — scenarios where errors in transmission channel are likely to occur. A brief summary is presented below.

### 3.2.1 Stream structure

The standard defines an *Elementary Stream* (ES) as the result of the encoding process of each audio, video and data essences of a media source. In order to facilitate manipulation of this continuous flow of data, elementary streams are divided into packets, originating a flow denominated as *Packetized Elementary Stream* (PES).

Each PES has a unique identifier, called *Packet Identifier* or PID. This value is used to indicate to which stream a packet belongs, and can be any in the range of 32 to 8190 [22]. A combination of PES packets sharing a common time base establishes a *Program* (or *Service*).

Since PES flows do not have any indication to which services they are associated with, a mechanism is needed to create this bounding. This is achieved with the usage of a *Program Map Table* (PMT) — a structure that contains a list of PIDs of each packetized elementary stream related to a program.

Finally, a *Program Association Table* (PAT) stores the PIDs linked to PMTs of all services in a transport stream. This table is always identified with PID 0 and must be transmitted frequently, allowing a decoder to recognize the structure of the TS. Figure 3.9 illustrates this semantics with an example.

On receiving end, reconstruction process starts with searching for PAT sections and PMT tables. Each of these provides the identifiers for elementary streams, allowing data to be cached and finally sent to the appropriate decoding module.

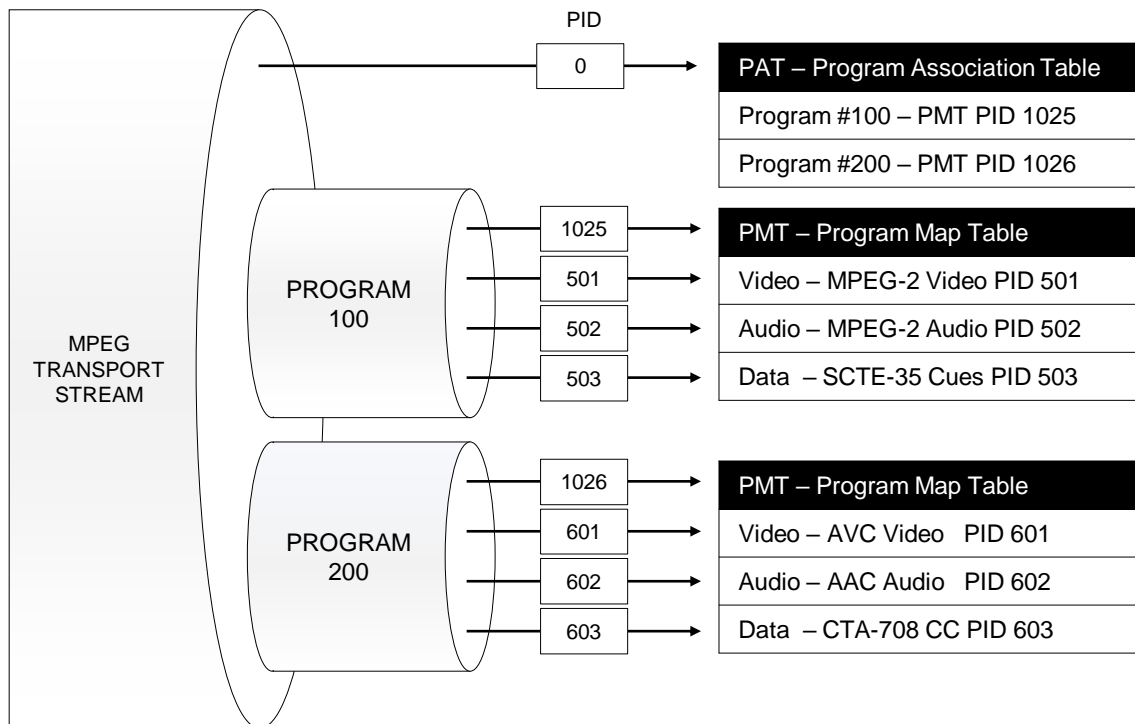


Figure 3.8: Illustration of MPEG-TS with two hypothetical services.

### 3.2.2 Packet composition

A standard TS flow is comprised of packets, which form the basic unit and has a total length of 188 bytes. First four bytes are dedicated to the header while the rest form the packet payload. All TS packets shall start with the MPEG-defined *synchronization byte* 0x47. Any packet not starting with this synchronization byte is considered invalid and rejected [23]. Other fields in the header are used for error checking, continuity, prioritization, among other utilities which are not going to be covered in this text.

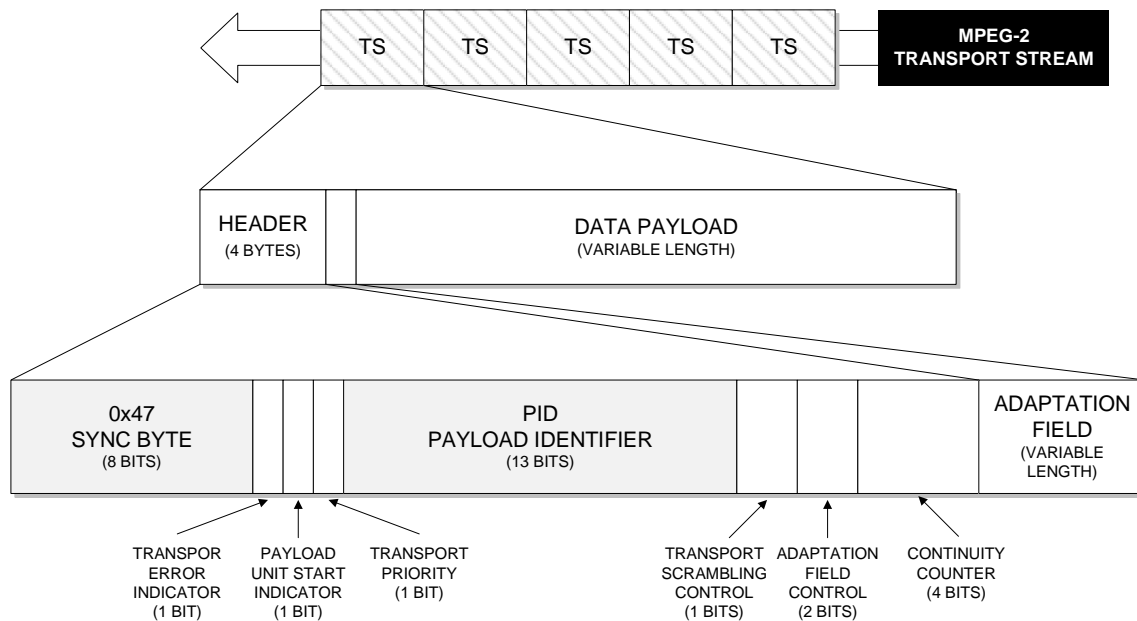


Figure 3.9: MPEG-TS packet structure.

When it comes to TV and media metadata, SMPTE ST 2038:2021 [24] defines the syntax for carriage of ancillary data packets in MPEG-2 Transport Stream PES packets. These guidelines should be carefully observed by registered applications and compliant hardware, specially professional encoders and decoders performing this sort of operation.

The MPEG-TS encapsulation will be particularly useful for the experimental implementation of the novel proposal, as presented in chapter 4. Many software tools can manipulate TS media, and some are able to read and write ancillary packets as described above.

### 3.3 SCTE-35/104 Standards

Created by ANSI and the *Society of Cable and Telecommunications Engineers*, SCTE-35 and SCTE-104 standards have been adopted by the industry for many years. Both work in companion as signaling protocols to inject markers on video stream, supporting delivery of events — frame accurate or non-frame accurate — and associated descriptive data [25].

These protocols are mainly employed for services such as *Dynamic Ad Insertion* (DAI) and *Digital Program Insertion* (DPI), allowing TV networks to inject triggers into their distribution feed. Upon reception, SCTE messages can be used to delimit break intervals and indicate to affiliates time slots in the stream where to automatically insert local advertisements. Some other applications have been proposed and are currently in use, such as prompting remote recordings, enforcing content restrictions, indexing, among others.

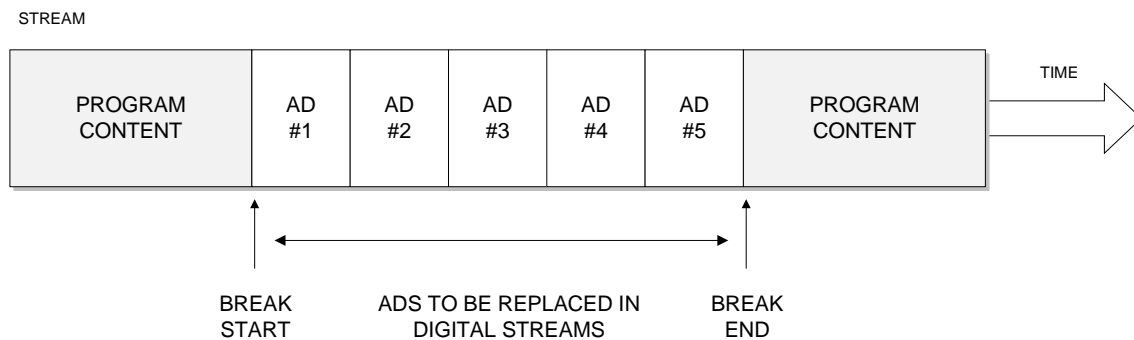


Figure 3.10: Typical advertisement break example, with slots that could be dynamically replaced using SCTE standards. Adapted from [25].

SCTE-35 and 104 are tightly intertwined, complementary to each other [26]. First published in 2001, SCTE-35 is the core signaling standard, originally contemplating events to be delivered through MPEG-2 Transport Streams. Later, in 2004, SCTE-104 emerged describing API messages that could be carried over TCP/IP data links or through the VANC space of unidirectional SDI base-band video [27][28]. These triggers are typically used as a precursor to eventual creation of SCTE-35 messages on outgoing compressed streams.

Another way of visualizing this mechanism is assuming that SCTE-104 cues in VANC space are requests from an operator or automation system to create splice markers in the video signal. Compliant MPEG-TS encoders processing this feed would translate these requests to SCTE-35, including all descriptive data within. This data is going to be interpreted by a dedicated device and appropriate action shall be taken upon live reception of the feed, as illustrated in figure 3.11.

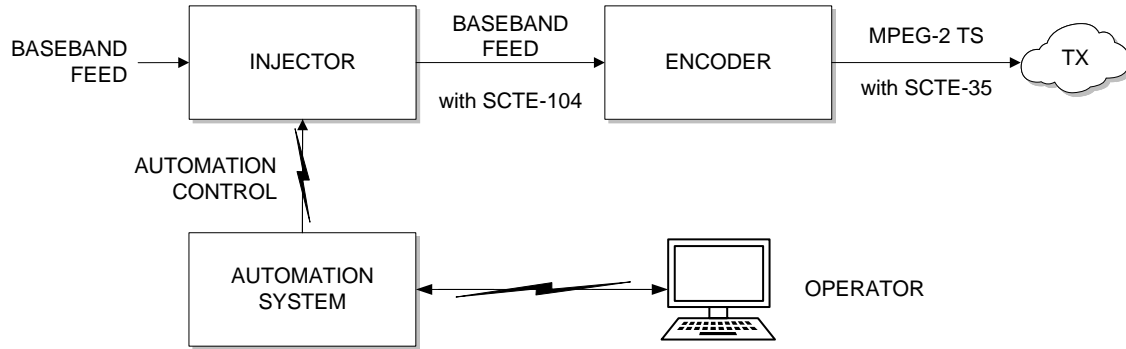


Figure 3.11: SCTE-35 vs SCTE-104 schematics, adapted from [25].

The system processing the messages on receiver side may be an ad server, a set-top box, a video switcher, or any other equipment. Based on the unique identifiers and flags injected, the final operation may be insertion of an advertisement, another piece of video, overwriting of content with a locally generated slate [29], or any other remotely triggered operation.

For the purpose of illustrating the semantics of this digital cueing protocol, next sections are going to focus on SCTE-35 signaling commands, as described in its normative. SCTE-35 can be more frame accurate than the legacy *audio cue tone system* (DTMF), also adopted by broadcasters. Furthermore, for outputs formats such SRT, ASI and IP ASI streams, no additional translation steps are required for SCTE-35 markers [30]. With the broadcast infrastructure moving deeper into OTT domains, SCTE-35 data can also be mapped into MPEG-DASH and HLS, very common WEB streaming protocols, which may be a great advantage.

Finally, it is important to note that while SCTE-35 is an international standard, its implementation can incorporate vendor or user specific set of rules. This level of flexibility gives rise to diverse interpretations of its contents and the manner in which operations shall be conducted. Supplementary documents later published tried to address this aspect, but it is essential to exercise caution and pay close attention to this characteristic when working with the protocol.

### 3.3.1 Splice points and events

To enable the splicing of compressed bit streams — which is the prime goal of the standard —, SCTE-35 normative defines a *splice point*. A splice point is a marker indicating a place in the bit-stream where a switch can be made. Splicing at such points may or may not result in good visual and audio quality, and that depends on the performance of the splicing device and adopted parameters.

The employment of advanced video compression, such as H.264 (AVC) or H.265 (HEVC) typically requires advance notice of splice points in order to allow the creation of spliceable markers in the video elementary stream. In most systems, this is provided by the use of SCTE-104, which offers a communication path from the base-band injection system to the encoder.

There are two types of splice points: *In points* are locations in the bit stream where it is acceptable to “enter” from a splicing perspective. *Out points*, on the other hand, are locations where it is acceptable to “exit” the bit stream. Both are hypothetical positions and can be co-located. This document will not delve into splicing streams and the complexities associated with performing such operations on compressed video. As a result, further details will be omitted.

According to SCTE-35 standard, a *splice event* is signaled through a *splice command*. The command identifies which point within a stream to use for that event and provides additional details for its interpretation. On receiving end, once stream has been captured and decoded, the cue messages are sent to the *downstream device*. This equipment may choose to act or not act upon a signaled event, and its behavior is not specified or constrained in any way by the standard.

### 3.3.2 Normative schema

Elements composing the semantics of SCTE messages are detailed by the normative text in combination with a XML schema document [31], comprising the full specification. Unless otherwise specified, the normative text and values assigned to attributes in this specification are constrained by the bit stream equivalent field.



No XML documents representing the structures defined in the schema are considered conformant unless they are valid according to the schema document [25]. For the purpose of illustrating how parameters are distributed and organized, a slightly adapted version of a compliant SCTE-35 message is included in Appendix A.

### 3.3.3 Splice commands

Commands are sent to the injection system with the appropriate flags and data. Each command has its own set of attributes to be filled and a summary adapted from the baseline normative text is shown below.

- **splice\_null()** — Allows a *splice\_info\_table* to be sent without issuing one of the other defined commands. The cue injection equipment may send these messages at intervals to be used as heartbeat messages, ensuring the proper operation of the system.
- **splice\_schedule()** — Command provided to allow a schedule of splice events to be conveyed in advance.
- **splice\_insert()** — The most important command, shall be sent at least once before each splice point. Provides ways of adjusting PTS of injection, *break duration* and a flag for indicating a network *in* or *out* point.
- **time\_signal()** — A time synchronized data delivery mechanism. Provided for extensibility by adding descriptors while preserving the precise timing allowed in the *splice\_insert()* command.
- **bandwidth\_reservation()** — Provided for reserving bandwidth in a multiplex. A typical usage would be in a satellite delivery system that requires packets of a certain PID to always be present at the intended repetition rate to guarantee a certain bandwidth.
- **private\_command()** — A structure that provides means to distribute user-defined commands using the SCTE-35 protocol. The first field in each user-defined command is a 32-bit identifier, unique for each participating vendor. Receiving equipment should skip messages containing *private\_command()* structures with unknown identifiers.

### 3.3.4 Injection principles

The main point about splice information is guaranteeing synchronization with video. Observing above mentioned commands and their composition, it is possible to roughly define two different classes of splice events: those that contain a specific PTS value (which refers to a *time stamp* in the video and audio PID's of the service), and all the others.

Commands that do not contain a time stamp are injected as soon as possible after reception [23]. Looking specifically at *splice\_insert()* semantics, a flag is available for indicating if the command should be working on *immediate* or *non-immediate* mode, as shown in figure 3.12. *Splice immediate* indicates that the splicing device shall choose the nearest opportunity in the stream, relative to the *splice\_info\_table*, to splice. When not in this mode, the message gives a *splice\_time* for the intended cue moment.

Syntax	Bits	Mnemonic
<code>splice_insert() {</code>		
<b>splice_event_id</b>	32	uimsbf
<b>splice_event_cancel_indicator</b>	1	bslbf
<b>reserved</b>	7	bslbf
if(splice_event_cancel_indicator == '0') {		
<b>out_of_network_indicator</b>	1	bslbf
<b>program_splice_flag</b>	1	bslbf
<b>duration_flag</b>	1	bslbf
<b>splice_immediate_flag</b>	1	bslbf
<b>reserved</b>	4	bslbf
if((program_splice_flag == '1') && (splice_immediate_flag == '0'))		
splice_time()		
if(program_splice_flag == '0') {		
<b>component_count</b>	8	uimsbf
for(i=0;i<component_count;i++) {		
<b>component_tag</b>	8	uimsbf
if(splice_immediate_flag == '0')		
splice_time()		
}		
}		
if(duration_flag == '1')		
break_duration()		
<b>unique_program_id</b>	16	uimsbf
<b>avail_num</b>	8	uimsbf
<b>avails_expected</b>	8	uimsbf
}		
}		

Figure 3.12: Semantics of *splice\_insert()* command, adapted from [25].

Due to the nature of novel proposal — which will be mostly handling the injection of graphics metadata into live feeds —, seems adequate to exclusively use *splice immediate* mode when decorating the stream. Additional details are elucidated in the subsequent sections.

Other fields of interest present in the command are the *splice\_event\_id* and the *out\_of\_network\_indicator*. First one is a 32-bits unique splice identifier, used to stamp the interval defined between splice points, while the second is the actual flag indicating an *out* point (when set to “1”) or a *in* point (when set to “0”).

According to the normative, *splice\_event\_id* values do not need to be sent in an incrementing order in subsequent messages nor must they increment chronologically, and may be chosen at random. Additionally, a *splice\_event\_id* value may be re-used when its associated splice time has passed, or when the logic behind the notified event has expired.

### 3.3.5 Overview

In the previous sections, we delved into the technical details of the protocols that will support our proposal, with emphasis on the essential components necessary for its implementation. More specifically, SCTE messages might help establishing a well-formed signaling model, enabling downstream systems to be implemented in a cost effective, consistent fashion. This topic is addressed in the following chapter.

# Chapter 4

## Novel Proposal

### 4.1 Introduction

As previously pointed, the practice of hardcoding graphics onto the video introduces a series of undesirable limitations, such as the inability to modify the data contents or adjust the composition for a new rendering. To enable greater flexibility throughout the production and distribution chains, graphics should be treated as an *additional layer of content*. This approach allows broadcast systems and/or end-user devices to handle content essences independently.

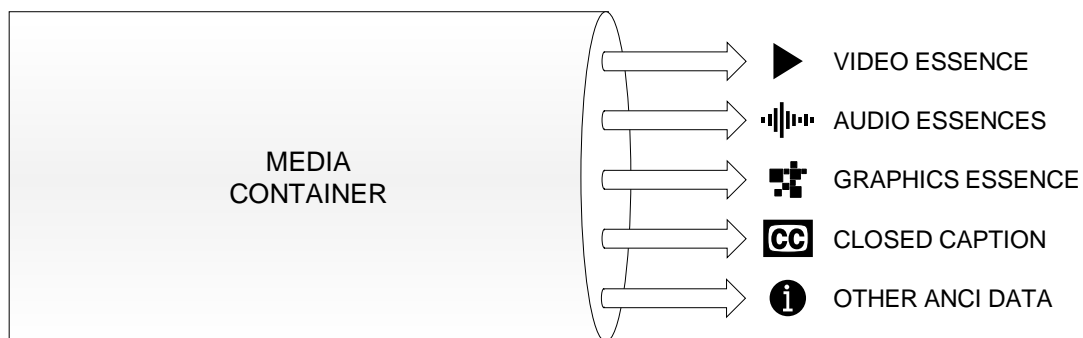


Figure 4.1: Illustration of separate media essences.

One effective approach to achieve this objective is exploring the ancillary space, which is encapsulated within media interfaces and distribution protocols. More specifically, usage of SCTE-35/104 cues may offer additional advantages, provided that adjustments to the message format are implemented appropriately.

One of these advantages lies in the fact that, being a well-known protocol, the current installed base of broadcast systems could potentially accommodate such messages seamlessly, opening up opportunities for automated GFX tools to interact with triggers embedded in SCTE-decorated feeds.

Several crucial steps need to be observed in order to support the intended proposal. These steps include specifying the nature of the data to be carried in the ancillary space and in the SCTE marker, determining how and when to activate its injection, establishing methods for decoding it at endpoints, and implementing control mechanisms to support all above mentioned operations.

## 4.2 Engines redefined

When delving into the proposed topology, the first important aspect to consider is the new role that graphic engines will play in this novel process. In the current workflow, actions taken by users operating the system are in fact translated into *commands*, dispatched either to graphic templates or to the engine via control channels. These commands are subsequently interpreted and processed to generate a visual composition, which is blended with the clean background video.

Since the goal is allowing graphics to be processed or rendered at distributed points along the transmission path, background video must be kept clean at this initial stage. On the other hand, commands issued to the engine, whether by users or externally, must be appropriately *adapted* into a symbolic representation that can be carried by the the chosen ancillary data protocols, namely SCTE-104 and SCTE-35.

Modern character generators, whether software or hardware-based, should provide sufficient scripting tools to seamlessly execute this task, possibly taking advantage of APIs utilized for linking the user interface with graphic assets. The adaption of commands shall result in *SCTE notifications* that are ready to be forwarded to the ancillary injector. The engine sending notifications will be referred to as the *host*.

Note that the host generator shall be connected to the ANCI embedder through the network, serial port, or any similar. This dedicated tool shall be the one actually decorating the real-time feed, following the normatives and user-defined instructions.

Furthermore, observe also that a host engine may still generate a video signal output, although this resource should be exclusively utilized for monitoring purposes. When an operator is dispatching overlays following director's instructions or in response to video actions, an auxiliary keyed output with the expected graphics result should be internally routed to a preview monitor within the production facility.

At the receiving end, a single or multiple graphic engines must be capable of converting the notifications captured from the ancillary data protocol back into the necessary commands for generating the visual composition, mimetizing the actions performed by the host CG. With the appropriate data sources and libraries, the graphic messages can be rendered seamlessly and automatically, finally to be blended onto the clean video feed. Engines receiving notifications will be entitled *replicas*.

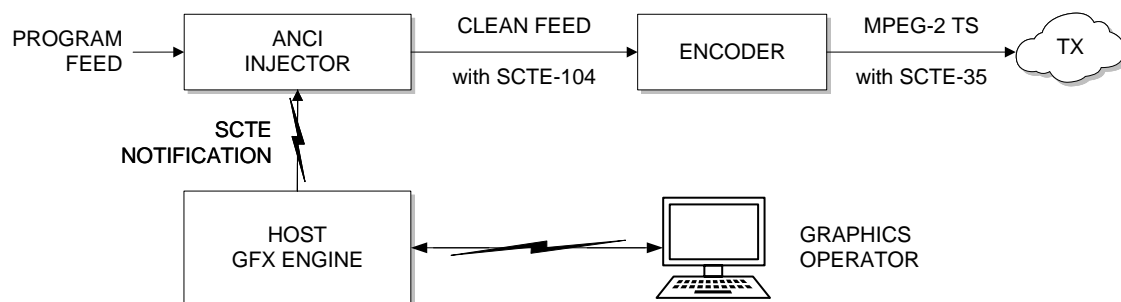


Figure 4.2: Illustration of host engine and ANCI injection system.

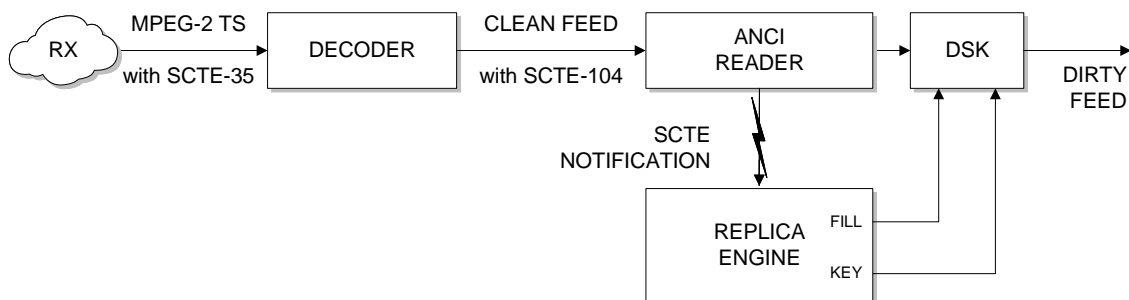


Figure 4.3: Illustration of replica engine and ANCI reading system.

## 4.3 Domain

Another point to observe is the establishment of a logic delimitation, baptized in this project as a *domain*. Inside a domain, systems and devices share a common set of rules and parameters, meaning that they can “comprehend” each other mutually.

In the context of this novel proposal, a domain consists of a group of graphics engines (hosts and replicas) capable of processing commands in a predefined format for a library of templates. In other words, messages generated by host CGs shall be interpreted and decoded by any replica CGs within the domain.

The extension of the domain depends on business models and each particular application. In theory, it could be as wide as wanted, encompassing the whole park of graphics generators serving a broadcaster, for example. However, this approach may face technical limitations, particularly when dealing with multiple live transmissions and extensive asset libraries.

In that case, a narrowed version seems more appropriate, attending specific demands or certain divisions of a broadcaster’s production site, for instance. Another example could be the establishment of a domain for large events with multiple feeds, like the ones discussed in section 2.3.1. This scenario is presented in section 4.7.1.

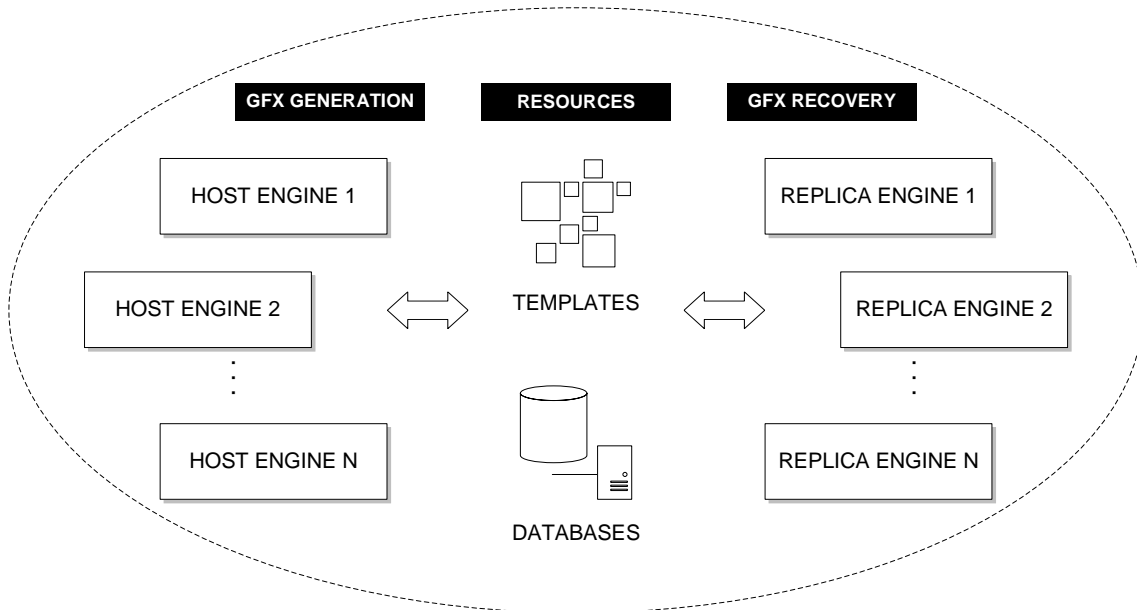


Figure 4.4: Illustration of a domain and its resources.

## 4.4 Template library

One of the crucial resources for the operation is the *template library* and its availability. Observe that graphics assets are created using static or dynamic data fields, including vector elements, images, or animations. These components can either be embedded within the internal data structure or referenced symbolically to external media files or similar resources.

The preferred and commonly adopted approach is to have all these assets available *online* in a shared network location, ensuring that eventual updates are instantly reflected in all systems. It should be noted that a graphic template may have versions tailored for TV, mobile devices, or other screen sizes. If that is the case, the replica engine must select the appropriate format based on its designated parameters.

In this project, it is also assumed that replica engines can efficiently switch and load any template from the library within a reasonable time, in a seamless way upon reception of the notification triggers, given the proper scripting to allow such operations. However, depending on the technology employed, there may be limitations imposed by the engine hardware or the control channel, which may affect retrieval times, memory size, and versioning – and thus should be carefully considered.

## 4.5 Notification format

As previously stated, host engines do not handle the direct placement of markers in the feed. Instead, they are responsible for adapting graphics metadata into SCTE notifications, which are then sent to the injection system. These notifications must include sufficient information for a replica generator to accurately decipher templates used, controls, and, most importantly, the data written in the graphics fields.

Evaluating SCTE-35/104 semantics and considering the constraints for ancillary message carriage, roughly two distinct strategies could be considered for performing the aforementioned task: writing the whole graphics metadata into the ancillary space or embedding an unique identifier to characterize GFX contents. Both methods are discussed below, highlighting the advantages and drawbacks of each.



### 4.5.1 GFX command ID

In this mode, every single *command* issued to graphic messages shall be assigned an *unique identifier*, which serves as a key for linking the template with the corresponding set of data populating its fields. These IDs and their associated content shall be stored in an external resource, baptized as the *commands database*.

The uniqueness of these identifiers guarantees that replica engines shall be able to retrieve templates, commands, and data solely by consulting the key stored in the collection. The extension of the domain and the number of host engines generating notifications directly relate with the size of this database. An orchestrator may be appointed to manage distribution of IDs in response to requests within a domain.

Notifications to the SCTE injection system shall be carried in a *splice\_insert* command, and the *splice\_event\_id* field shall correspond to the unique ID designated by the database manager. As shown in Appendix A, this field has a total length of 32 bits, *unsigned int* type. Consequently, the commands database should be limited to a maximum of  $2^{32}$  entries, which corresponds to more than 4 billion commands. Workarounds for this restriction may be reducing the domain size, either on the amount of host engines generating entries or implementing a time-frame constraint.

Furthermore, the notification shall be sent with the *splice\_immediate* flag set to “1”, ensuring that the cue is written into the feed as soon as possible upon its reception by the ancillary embedder. Other flags and fields shall be maintained with their default values. The process of receiving the command to injection should take as little time as possible. Typically, anything below 0.5s is virtually imperceptible.

One advantage of operating with a command database is the ability to modify, update, or remove previously registered contents. For instance, consider a scenario where a feed decorated with GFX metadata is recorded, and the ancillary essence is successfully transferred to the media. Suppose that, later on, changes are made to the database to rectify data that has been wrongly displayed. If the material is retrieved, reading of IDs by replica generators would trigger the creation of a composition with the updated version of data.

Naturally, it is essential to maintain the appropriate access rights for reading and writing in the database, whether it is stored online or offline. Additionally, it is assumed that the total time required for I/O operations in the database is shorter than the time needed for decorated feeds to be transported from the source to the destination. This is typically the case, especially when audio and video have to be compressed and decompressed along the path.

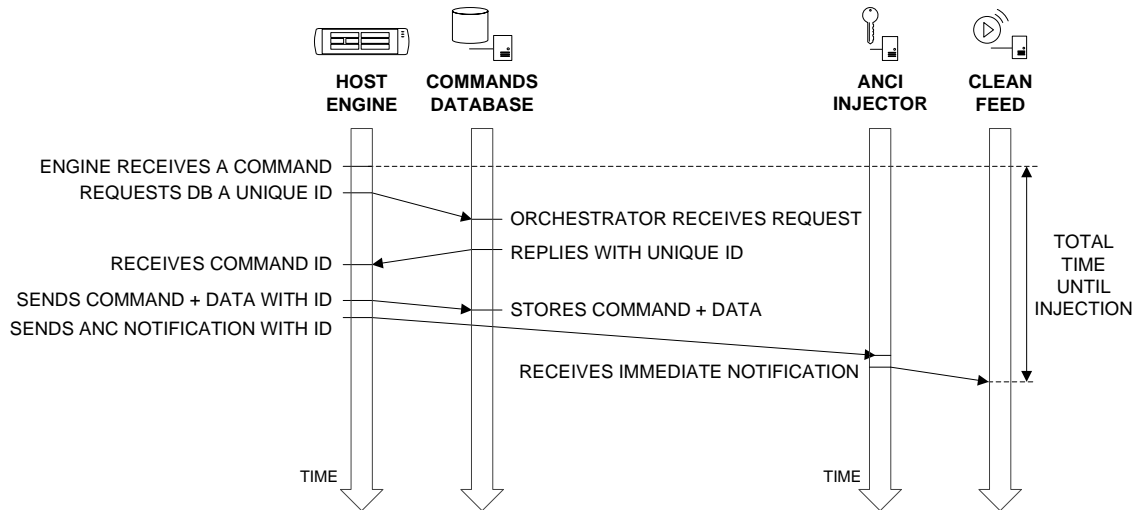


Figure 4.5: Illustration of steps until injection.

### 4.5.2 Standalone mode

This mode allows broadcasters adopting the solution to write their graphics metadata in a customized format, appropriate for their domains. As a requirement, the entirety of data necessary to populate the graphics shall be embedded in the notification itself. Consequently, the receiver side only needs access to the templates library and the clean feed, simplifying the recovery process significantly.

In this scenario, the notification for the SCTE injection system shall be carried in a *private\_command()*, as briefly mentioned in section 3.3.3. Broadcasters interested in utilizing this method must undergo a registration process to obtain a valid user identifier for their SCTE messages. The advantage of this approach is its simplicity, since it eliminates the requirement for connection to an external commands database.

On the other hand, post-editing of data contents becomes virtually impossible once the material is distributed, unless a complex ancillary data amendment is performed at each endpoint. This mode may be particularly suitable for overlays that involve rapidly changing data, generated by real-time sensors, timing systems, or similar sources. In such cases, updates to the displayed values can be directly forwarded to the notification system without requiring any sort of database I/O.

## 4.6 Ancillary manipulators

Once the ancillary message is properly formatted, it needs to be injected into the real-time feed. This task can be accomplished using either software or dedicated hardware. Conversely, extracting the messages from the feed follows an analogous process: an ancillary reader retrieves the SCTE markers and forwards the messages to the replica engine.

For instance, consider the device and schematics depicted in Figure 4.6. It showcases the features of an ANCI embedder/de-embedder electronic card, which receives incoming SCTE-formatted messages through an ethernet interface and injects them into the feed, whether SD, HD, or 3G-SDI. The same card model can also be employed for extracting user data. For more operational details, refer to Annex B.

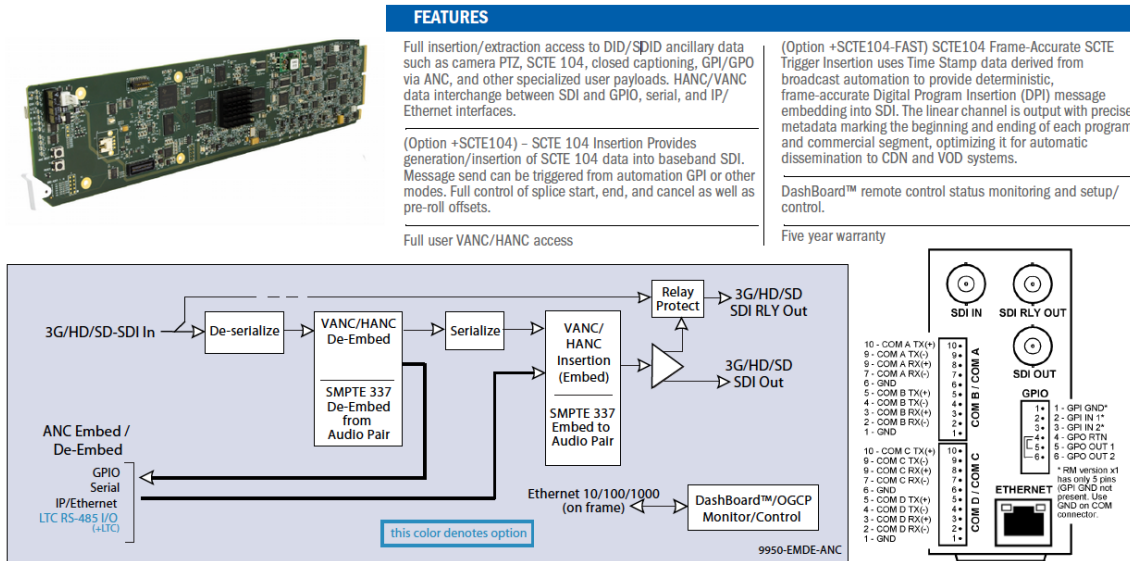


Figure 4.6:

Ancillary embedder/de-embedder, manufactured by Cobalt Digital©. Source [32].

## 4.7 Scenarios

Now that the fundamental concepts for implementing the novel proposal have been introduced, let's explore some situations in which its potential advantages become evident. Below scenarios are based on the case study discussed in section 2.3.1, where we examined the operational complexity and graphic generation in large-scale events under specific circumstances.

### 4.7.1 Multi-language graphics

The new approach, outlined below, focuses on overlays that do not involve dynamic data updated in real-time, such as those derived from sensors. These graphics have unique characteristics that, for the sake of simplicity, are not going to be addressed in this text.

As briefly mentioned, automated and on-the-fly generation of compositions in multiple languages requires a costly infrastructure replication in the current workflow. Moreover, this capability is typically limited to a few selected feeds, contingent upon broadcasters willingness in funding such a solution.

In the proposed workflow, the host broadcaster remains responsible for the graphics generation service, but this time through the establishment and management of a domain, including its resources. Ideally, this domain would have a time-limited scope corresponding to the duration of the event, due to the nature of these productions.

For each local production unit, a graphic engine shall be allocated, following a similar approach to the previous model. However, this device would function solely as a host generator, sending SCTE notifications to an ANCI embedder installed at the end of the PGM chain for each multilateral feed, as shown in Figure 4.7.

In this proposal, a command database would be employed, thus utilizing the *command ID* mode as discussed in section 4.5.1. Additionally, a new generic resource, called the *translation database*, would serve as a repository for translated graphic components such as names, competition venues, schedules, events, and more.

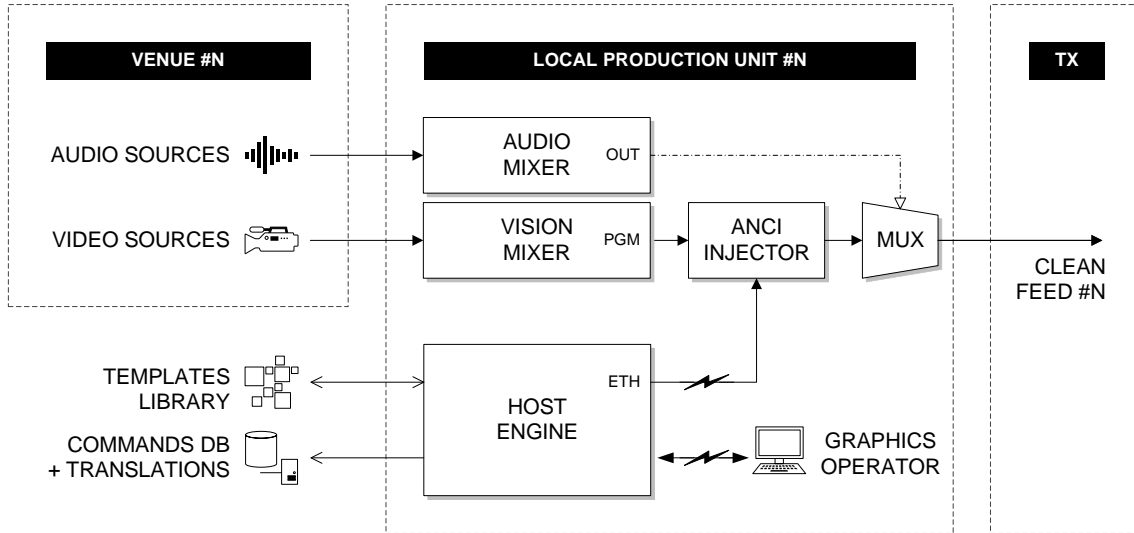


Figure 4.7: Injection workflow in one of the multilateral feeds.

Once the clean feeds are decorated with graphic metadata, they should be routed through the traditional real-time distribution and transport paths, until they reach the point where they will be rendered in a visual composition. This operation shall take place within the premises of the media rights holders, being one crucial aspect of this topology. Access to templates library and databases that are part of the event domain must be granted.

The replica engines, serving rights holders, should receive incoming SCTE notifications from ANCI de-embedders installed at the end of the PGM feed of the broadcaster's production unit. By doing so, graphics could be automatically generated for the signal being aired, based solely on the injected markers captured from the ancillary essence. Even if a switching between different video feeds occurs, the generator would be able to recognize the GFX IDs and create the visual composition accordingly. Having one engine capable of rendering graphics for multiple feeds and packages creates a substantial opportunity for cost savings.

Based on their specific configurations, the replica engines could also translate graphics content into the desired language for the audience. To achieve this, appropriate databases and translation scripts need to be developed. Depending on the structure of the messages, it may even be possible to convert numerical values and units, such as between the *metric* and *imperial* systems, for example.

Finally, the rendered graphics need to be blended with the clean feed through the keying process, as discussed and exemplified earlier. Figure 4.8 illustrates the hypothetical reception of feeds package, internal routing and the graphics recovery process.

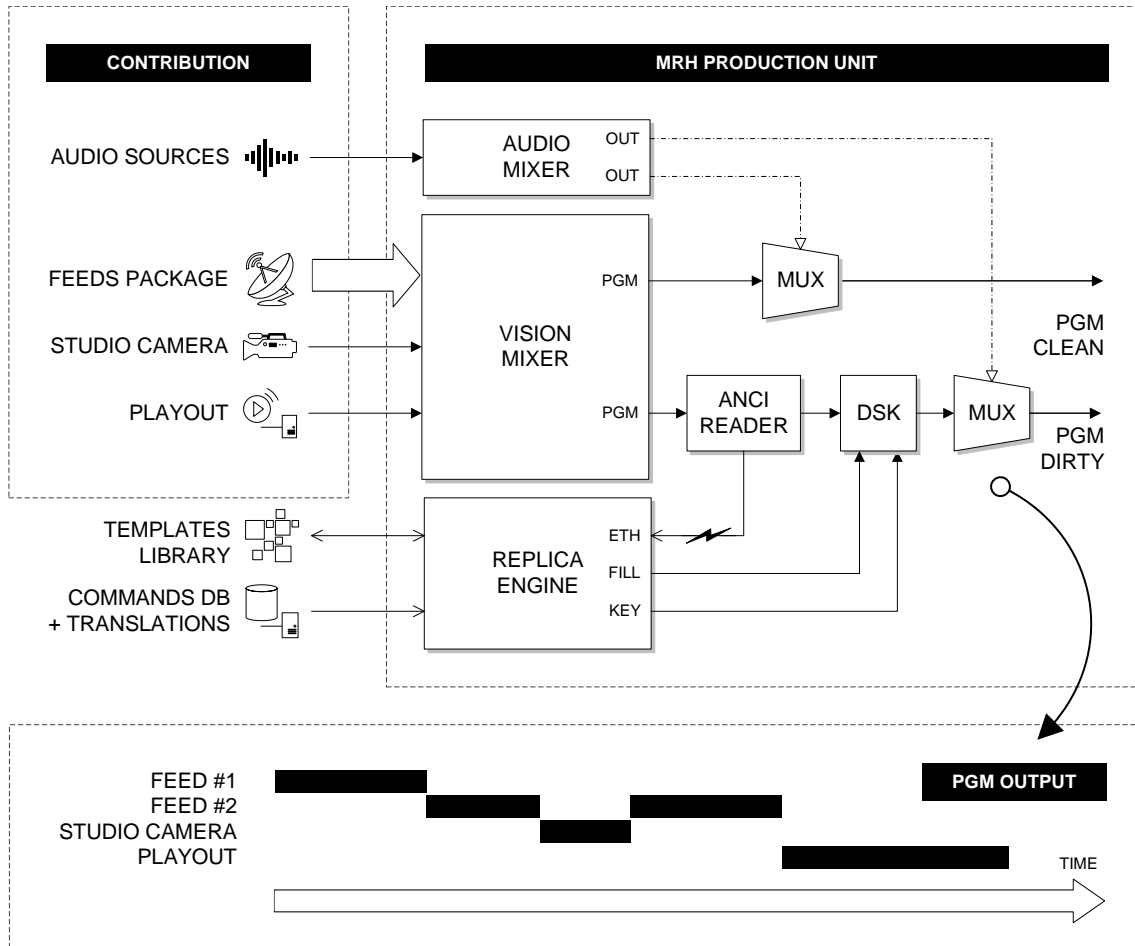


Figure 4.8: Reception of feeds package at rights holder premises.

One critical aspect of the proposal is ensuring that the metadata injected into the ancillary data space is carried throughout the entire broadcast chain. This requires all systems along the transmission path to be capable of effectively transmitting and preserving this metadata. For example, consider the technical specifications of the *Carbonite Black* [33] vision mixer, which can be configured to strip or pass ancillary data from the video output. The amount of data, and how it is stripped, depends on the video format of the video signal. Additionally, reference documentation also indicates that some video manipulations may automatically strip metadata from the feeds.

## 4.7.2 Adapted video formats

Consider now the scenario of distributing content on mobile devices, where screen sizes and aspect ratios vary. To address this, we will retain part of the previously proposed formulation with host engines and ANCI embedders on the clean feed generation side. The main change occurs on the receiving end.

Imagine that a broadcaster intends to generate two different output formats for its feed: one for linear TV and another for digital distribution, with a video format adapted for mobile devices, for example. In this case, two replica engines could be allocated, directly in the master control room serving the output channels for each of these platforms. This arrangement would allow the generation of adapted graphics for both formats, provided that the necessary templates are available.

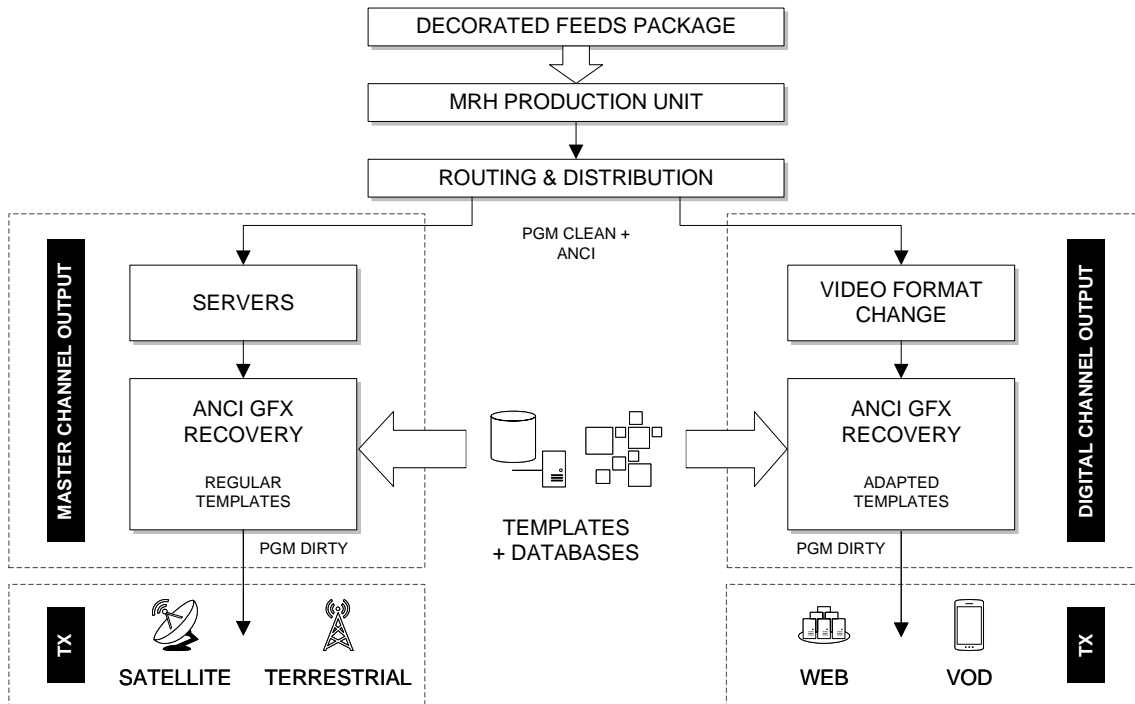


Figure 4.9: Proposal of simultaneous graphics generations for distinct output channels.

A great way of testing the scenarios presented above would be implementing a small-scale model of the production infrastructure. Certain broadcasting tools can be replicated in digital application, even if not directly dealing with real-time video signals. Aspects related to templates and database manipulation could be evaluated, which is one of the goals of the experimental model to be presented next.

# Chapter 5

## Experimental Implementation

### 5.1 Introduction

The following simulation focuses on the usage of SCTE-35 cue messages within our proposed topology. It relies entirely on adapted software solutions capable of handling transport streams, splice injection, and software-based graphics generation. The objective is evaluating the overall behavior of the system, particularly when writing or reading ancillary notifications.

To showcase the flexibility of the proposal, a domain shall be established, and simple graphics generated for two generic and recorded MPEG-TS clean feeds. These feeds will be decorated with SCTE markers, indicating the presence of graphics metadata that will be carried alongside with the video essence. After undergoing a network distribution stage, the feed package shall be captured by an independent graphics recovery system, with its assigned replica engine. This engines shall generate the corresponding visual composition based on the ancillary data received.

In addition, a setting on receiver side shall allow the user to select the desired language for rendering the graphics. Two options shall be offered: *English* and *Japanese*. A mechanism shall be developed to allow switching between the two streams being transmitted in the network. Furthermore, templates shall be shared among both engines, and two output formats should be available: TV and mobile. Commands issued shall be stored online, in a database with the appropriate reading permissions, simulating a large-scale distribution of the feeds package.



## 5.2 Methodology

The implementation of this solution will primarily rely on two broadcast toolkits: *TSDuck* and *NodeCG*. The first will be responsible for the manipulation of transport streams, including the injection and extraction of tables and splice markers. The second is a graphics generator framework, which is entirely based in web technologies. It supports a collection of custom-built templates and control APIs. More detailed information about their characteristics is provided in the following subsections.

For testing purposes, generic video materials are going to be used to create two sports-like feeds: one with *swimming* footage and the other with *athletics* content. These videos have been obtained from public repositories and can be freely distributed. A processing stage will adapt these clean video feeds to an specific output format.

Regarding the graphics generation process, there are two distinct workflows to be considered, both sharing the same adapted infrastructure, as summarized below.

- **Transmission workflow** — Where GFX commands are issued, processed, sent to the database and SCTE messages injected into the clean video feed.

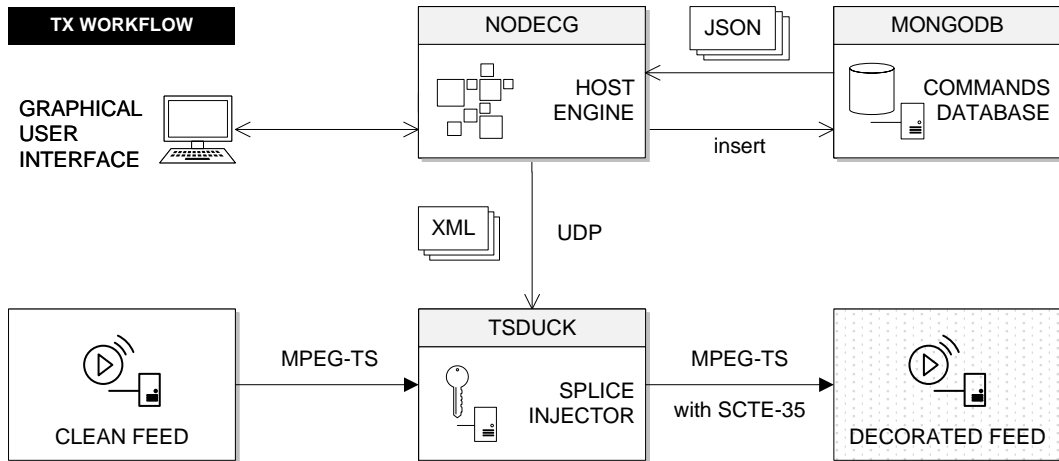


Figure 5.1: Experimental transmission workflow for both feeds.

- **Receiver workflow** — Where both clean feeds are captured and routed, SCTE cues extracted, database queried and visual composition rendered.

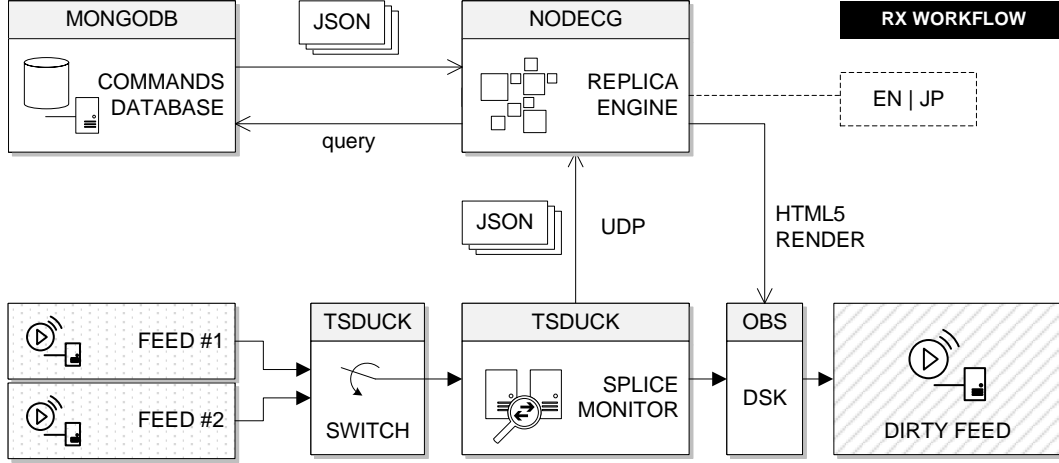


Figure 5.2: Experimental receiver workflow and dirty feed output.

## 5.3 Tools and material

### 5.3.1 Hardware

For this project, two *Raspberry Pi 3* single-board computers are going to be utilized as stream servers and SCTE injectors. These devices will run the standard *Raspbian* operating system with optimized instructions for the architecture. Both TSDuck and NodeCG can be executed on these units without significant issues, as long as the necessary dependencies and libraries are installed.

On the receiver side, a standard computer running a *Linux* distribution will be used to fulfill the roles of TS selector, SCTE monitor, replica engine, and downstream keyer. As it generates the final video output, enough GPU power should be available to handle the manipulation of video streams. It is also possible to add multiple client sets to decode and process the streamed content with ease.

Lastly, a network switch interconnects the devices and will serve as a video feed distributor. In order to support this operational mode, transport stream packets will be forwarded to a *multicast address*, ensuring that all receivers get the same network traffic. Multicasting usually requires *IGMP snooping* to be enabled on the device. For the purposes of the tests, an *EdgeRouter X* unit operating in switch mode will be employed.

### 5.3.2 TSDuck utilities

TSDuck is a large C++ library for transport stream manipulation. Provided services includes low-level features such as manipulating TS packets, intermediate features such as demuxing and packetizing tables and high-level features such as running TS processing pipelines [23].

As an open-source project, TSDuck also includes powerful filtering capabilities. Custom filters can be defined to extract specific components from a TS, such as audio or video streams, tables, or data. These filters can be combined and chained to create complex processing workflows, allowing users to manipulate and modify content according to their requirements.

In addition, a variety of input and output formats are supported, including file-based operations, network streaming, tuners and similar devices. It can receive and transmit transport streams over IP networks, making it a versatile tool for applications.

### 5.3.3 NodeCG platform

NodeCG is an open-source broadcast graphics framework, allowing creation and management of dynamic and custom-built graphics for live streaming, shows, and similar broadcast events. It combines the power of web technologies with the flexibility and real-time capabilities required for live productions. Its core is built with *Node.js*, one of the most popular JavaScript runtime environments. The service runs as server-side manager, acting as a bridge between the broadcast software and the graphics elements, promoting seamless integration and communication.

NodeCG follows a modular architecture, where graphics are defined as individual *bundles*. These resources can be created and customized to suit specific production requirements. The basic structure of bundles consists of a HTML template, including CSS stylesheets and JavaScript code. A control channel can be established between these, the server and a *web dashboard*, constituting a complete management ecosystem for the production.

In addition, NodeCG event-driven architecture allows response of graphics elements to events triggered by external resources. For instance, a graphic could be updated when a specific message is received or when a button is clicked. This characteristic is specially useful for this project, which will be handling the interactive reception and injection of notifications from and to the SCTE-35 automation system.

On server-side source code, modules can be incorporated to NodeCG for additional services and functions, developed by the community. One of these modules allows network *socket* manipulations, which can be used to interface and control TSDuck tools and parameters. Another module will be used to set up communication with the external database service.

#### 5.3.4 MongoDB

The GFX commands database will be entirely hosted in a MongoDB instance, in the cloud. MongoDB is a distributed database at its core, with high availability and scalability. As a non-SQL service, data is stored in flexible, JSON-like format, meaning that fields can vary from document to document and structure can be changed over time [34]. It is possible to map these models to objects in our CG application code, making data easy to work with.

Two authorization profiles will be created in the cloud instance: *host* and *broadcaster*. First profile is allowed to write and modify the database freely, and, therefore, will be used by host engines and administrative tools. Second profile has only permission to retrieve data, designated to be used in replica engines of our prototype.

#### 5.3.5 Open Broadcaster Software

Open Broadcaster Software is an application that provides powerful tools for video recording and live streaming. It is widely used by content creators, and professionals in the broadcasting industry to capture, mix, and broadcast high-quality multimedia content. OBS offers a user-friendly interface with a wide range of features and customization options, making it accessible to both beginners and advanced users.

It is supported by multiple platforms, including *Windows*, *macOS*, and *Linux*, providing also extensive compatibility with various video and audio capture devices, cameras, microphones, and other peripherals.

OBS has a powerful scene composition system that allows creation of complex layouts by combining various media sources, such as images, videos, text, and even HTML5 pages. This last feature is particularly interesting for creating the final graphics output mixed to the clean background stream in our project domain. Therefore, OBS will play the role of a keyer in this experimental implementation.

### **5.3.6 *FFmpeg***

*FFmpeg* is the leading multimedia framework, providing a collection of libraries and command-line tools for handling audio, video, and other sort of streams. One of the key features of *FFmpeg* is its ability to convert, encode, and decode media files, supporting a vast range of formats, from the most obscure to the most popular ones.

Known for its robust media manipulation, *FFmpeg* cross-platform filters and processing tools allows the adjustment of parameters such as resolution, bitrate, and frame rate. Furthermore, tasks like resizing, cropping, and rotating videos can be performed with ease. These features will be handful when adjusting the media files to be transmitted, as described below.

## **5.4 Environment setup**

### **5.4.1 Relevant aspects**

To simulate the real-time transmission of the two feeds, the video material needs to undergo treatment beforehand, in order to adapt it to the network infrastructure and hardware conditions. In addition, the testing environment needs to be calibrated to the packet-routing reality, which imposes a few constraints when compared to a circuit-switching workflow.

The first aspect to be considered relates to *latency*. When using an IP network, even in a small and controlled environment like the proposed one, introduction of delays are inevitable due to the presence of buffers and the digital processing steps. It is possible, however, to optimize certain parameters and minimize latency as much as possible in all stages, from reading of media files, transmission and decoding on receiver side.

Second aspect relates to *video integrity*. Since the idea is allowing the user to switch between different streams in the network, players and tools decoding the feed must be somehow prepared to handle the abrupt cut from one data flow to another.

## 5.4.2 Network

Each video stream will be transported in UDP multicast flows within the network. This way, multiple receiving devices may subscribe to the same content without overloading the transmitter. The UDP protocol is also more suitable due to its simplicity, avoiding TCP overhead and handshaking.

The network is configured to have a MTU of 1500 bytes, which is the maximum size of a data packet for transmission. To avoid unnecessary fragmentation and overflow in network buffers, the data should be sent continuously and in controlled way at each stage. Considering headers and that a MPEG-TS packet has 188 bytes, a maximum of 7 packets should be encapsulated in the UDP payload at each time, totaling 1316 bytes.



Figure 5.3: Representation of TS packets encapsulated in IP/UDP flow (not in scale).

### 5.4.3 Media preparation

In order to create the two files with *swimming* and *athletics* content to be loop-streamed, both raw footages should first be cut in smaller clips with 60 seconds each. A running timecode hard-coded to the video shall be incorporated, to help checking synchronicity and smooth playback later on.

In the next step, video should be re-encoded, using a codec and parameters optimized for live streaming and real-time transmission. Also, a reasonably constant and predictable data flow is desirable, to avoid short-time spikes on bandwidth usage and buffered information. To achieve this, x264 encoder will be adopted, with a relatively consistent target bitrate and zero latency flags. These shall increase reliability for transmitting time-sensitive content and boost encoding speed.

The above mentioned approach, however, may lead to lower overall video quality and increased media sizes, specially when compared to the adoption of a true VBR encoding method. For this experimental implementation and its controlled environment, bandwidth restrictions are not a major obstacle, but manipulation of large media files should be avoided. To reduce operational complexity, video resolution has been set to 1280x720 with progressive scan mode and framerate of 30 fps. Target bitrate is set to be around 10 Mbps and all audio tracks removed.

Jumping from one transport stream to another will not necessarily result in a clean video switch, because of discontinuities on packets contents affecting the decoding process of the the media structure. However, to avoid glitches and big disruptions on monitored video, the GOP structure can be adjusted. In video compression, the concept of GOP (Group of Pictures) plays a crucial role, representing a set of consecutive frames, of different types, that are encoded together as a unit.

*I-frames* (or keyframes) are reference frames that are encoded independently, without relying on any other frame for decoding. They are complete pictures, containing all the necessary information for the rendering process. Therefore, they have the highest quality and size, serving as reference points for decoding subsequent frames in the GOP.

*P-frames* stands for “predicted frame”, meaning they rely on previously encoded frames to forecast the content. *B-frames* are frames that can reference both preceding and subsequent frames within the GOP (B standing for “bi-directional”). Both take advantage of motion estimation and temporal redundancies to achieve higher compression ratios.

The arrangement of these frames within a GOP is typically structured as shown in figure 5.4. The specific configuration of GOP depends on the video codec and its settings, directly affecting the trade-off between video quality and media size. The GOP length (meaning distance between keyframes) may also impact smooth playback of streamed content.

For this experiment, the encoder will be set to create only *I-frames* in each media file, meaning a GOP with length equal to 1. This way, when a switch between streams occurs, decoders on receiver side will take less time to recover from the abrupt break in the video rendering process, as illustrated in figure 5.5.

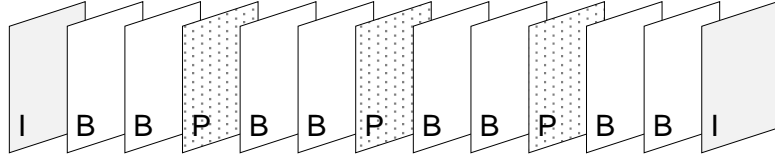


Figure 5.4: Generic illustration of a typical GOP structure.

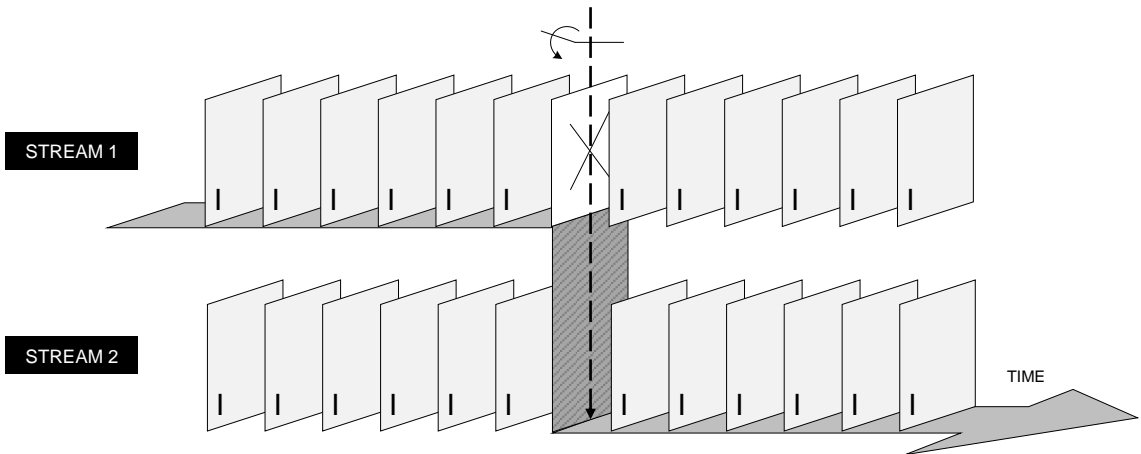


Figure 5.5: Adopted GOP structure for experimental streams, easing switching.



#### 5.4.4 TS encapsulation

After the media is prepared, it must be wrapped into the MPEG-TS. The H.264 video essence shall occupy its own PID inside *service 1*, whilst space shall be reserved in the multiplex for SCTE-35 markers. The additional space, when compared to video bitrate, is to be filled with other TS tables and stuffing packets.

Each time a new TS packet needs to be injected into the stream, *TSDuck* plugins waits for the next null packet and replaces it with the new packet. Same happens when *splice injection* is requested, specially when using *splice immediate* mode, when stuffing packets shall be replaced with the actual SCTE-35 cues as soon as possible.

Consequently, the original amount of stuffing and its distribution in a stream directly influences the insertion profile of new packets, since it is not possible to add more data than the stuffing bitrate. Moreover, precise timing cannot be always achieved. In broadcast streams, where the modulation parameters impose a fixed bitrate, there is always some stuffing [23].

Since there is no requirement for a global bitrate for the experiment, artificial stuffing will be inserted at input level on media files using *TSDuck* plugins. Arbitrarily, encoding script has been set to add one null packet every 9 regular packets, increasing total bitrate by approximately 10%.

Finally, the *splice information table* in the transport stream that carries SCTE-35 signaled events should be created using a common MPEG-TS *stream\_type* (0x86), which identifies all PID streams containing this type of data. PID 600 in *service 1* has been selected to transport our graphics metadata.

The *bash script* below has been developed to streamline the media creation process and will be available in the project repository. It can be used to convert all files within a specific source folder into the corresponding output TS media, ready to be served over the network. Figure 5.6 presents the packet distribution over time for one converted file, evaluated by *DVB Inspector*, a TS analyzer tool. It shows a reasonably constant bitrate for the video essence, as desired.

```

1 #!/bin/bash
2
3 i=1;
4
5 for f in $(pwd)/../streams/source/*
6 do
7     ffmpeg -i $f -c:v libx264 \
8         -b:v 10M -minrate 10M -maxrate 10M -bufsize 5M \
9         -g 1 -s 1280x720 -r 30 -pix_fmt yuv420p -crf 1\
10        -tune zerolatency -movflags faststart+frag_keyframe+empty_moov \
11        -y tmp.ts \
12
13    tsp --verbose --add-input-stuffing 1/9 \
14        -I file tmp.ts \
15        -P pmt --service 1 --add-pid 600/0x86 \
16        -O file $(pwd)/../streams/stream$i.ts
17
18    i=$((i+1))
19    rm -f tmp.ts
20 done

```

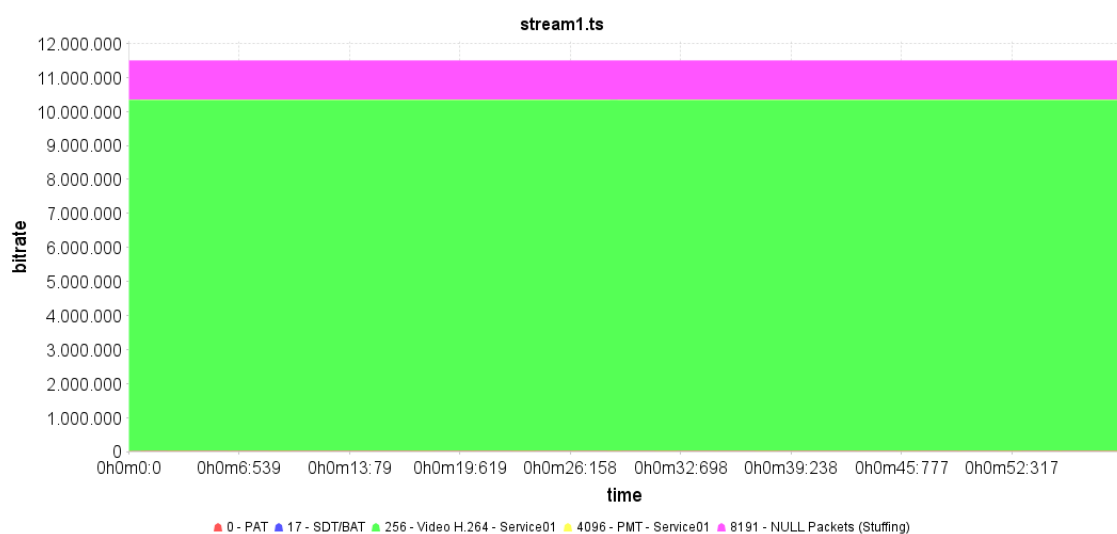


Figure 5.6: Stream analysis showing no significant spikes on video bitrate (green).

## 5.5 Code development

### 5.5.1 Bundles architecture

In order to integrate the graphics assets, control channels, database and scripts, a design pattern must be established, connecting modules inside NodeCG platform. As previously mentioned, each NodeCG bundle is a smaller collection of files, templates and codes, created as a self-contained package encapsulating a set of graphics or functionalities. They are loaded when NodeCG instance is initiated.

For the project, two bundles will be created: *main* and *games*. Main bundle, as the name suggests, incorporates all major services for operating the graphic engine, whether as a host or as replica generator. It will include modules for connecting with the MongoDB database, language selection, channels for data exchange with templates, OS calls for scripts and network sockets for sending SCTE notifications. It also includes a web-page player, with the graphics output.

The other bundle is the actual graphic package, containing all elements for creating a visual composition for both swimming and athletics content. It also includes small dashboards for operational control and *JavaScript* code for data exchange with main bundle. Each package comprises 3 assets, with *in* and *out* animations created using HTML5 and CSS. Figure 5.7 illustrates the basic folder architecture.

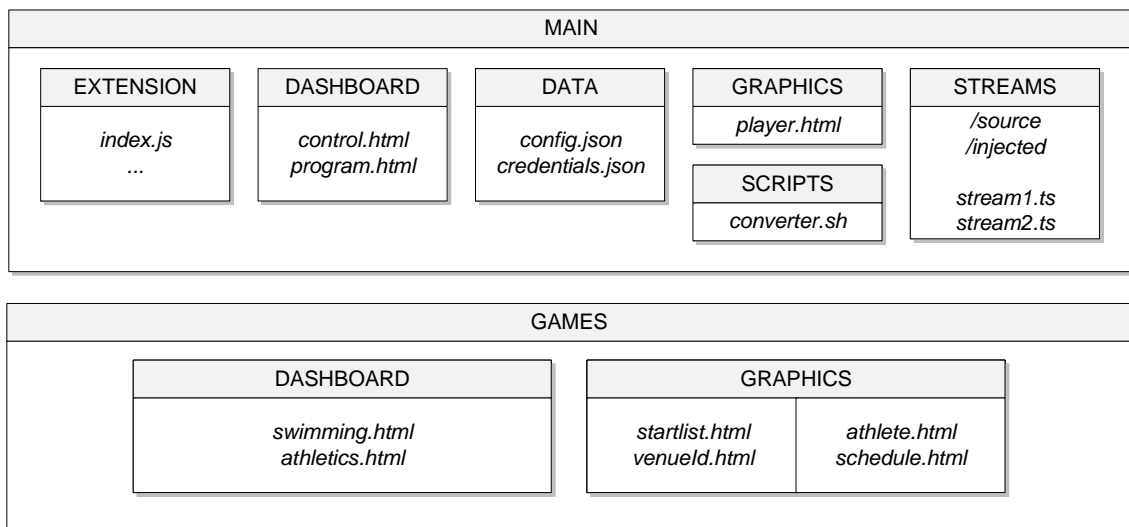


Figure 5.7: NodeCG bundles architecture and code organization.

## 5.5.2 Operational modes

The experimental code shall be developed allowing combination of both operational modes for the graphics engine, which shall act as host, replica or both at the same time. These settings directly affects which bash scripts and modules shall run when NodeCG instance is initiated.

- **Host mode** — If enabled, the engine activates the *splice injector* plugin and shall start listening for SCTE-formatted notifications in a designated network port. Commands dispatched on the GUI shall trigger a process for decorating the input stream accordingly, including assignment of an ID and insertion in the database.

The output stream shall be addressed to an IP, and, optionally, might be recorded back into a new TS file. The clean video can be played with low-latency parameters, for monitoring purposes.

The equivalent script for streaming and injecting both swimming and athletics content is presented below. The variables are to be filled with parameters in the configuration file.

```
1  #!/bin/bash
2
3  echo Input control on IP $1, transmitting file on PATH $2
4  echo Listening SCTE notification on IP $3,
5  echo Recording file PATH $4 (if set), broadcasting on IP $5
6
7  tsswitch --remote $1 --fast-switch --buffer-packets 14
8          --max-input-packets 7 --max-output-packets 7 --verbose
9  -I fork 'tsp --verbose
10         -I file $2
11         -P regulate'
12  -O fork "tsp --verbose --realtime
13         -P spliceinject --service 1 --udp $3
14         -P fork 'tsp -O file $4'
15         -O ip $5"
```

- **Replica mode** — If *replica* mode is enabled, *tsswitch* plugin receives one or more feeds. This is the TSDuck utility that allows jumping between multiple MPEG-TS streams given as input parameters. The output stream is then forwarded to a *splice monitor* plugin, which shall monitor any SCTE service in the data flow and redirect notifications to an user-specified address.

Finally, the video feed is forwarded to an output IP, to be captured by the system acting as a *downstream keyer*, which will finally mix the output graphics with the selected feed.

The equivalent script to run in the replica engine is presented below. The variables are to be filled with parameters in the configuration file.

```

1  #!/bin/bash
2
3  echo Feed switch control on IP $1
4  echo Receiving feeds on IPs $2 and $3, from IPs $4 and $5
5  echo Forwarding SCTE notification to IP $6
6  echo Recording file PATH $7 (if set), playing on IP $8
7
8  tsswitch --remote $1 --fast-switch --buffer-packets 14
9          --max-input-packets 7 --max-output-packets 7 --verbose
10 -I ip $2 --source $4
11 -I ip $3 --source $5
12 -O fork "tsp --verbose --realtime
13         -P splicemonitor --json-udp $6
14         -P fork 'tsp -O file $7'
15         -O ip $8"
```

- **Host + Replica modes** — If both modes are activated, the system will run above mentioned modules in parallel. If there are no conflicts in IP addresses, users shall be able to decorate the input clean feed with SCTE cue markers while monitoring the graphic outcome at the same time. This single-device operational mode should be avoided, unless observed enough computational power.

### 5.5.3 Configuration file

All above mentioned modes and values can be modified in the *config.json* file, located at the data folder inside *main bundle* directory. By default, host and replica modes are enabled, with IPs and ports to avoid conflicts between all services (shown below). Systems running the application should adjust the parameters accordingly.

```
1 {
2   "id": "PGM",
3   "tx": {
4     "mode": true,
5     "ipInjector": "127.0.0.1:5555"
6   },
7   "rx": {
8     "mode": true,
9     "ipMonitor": "127.0.0.1:4444"
10  },
11  "inputs":{
12    "ipSwitch":"127.0.0.1:2222",
13    "paths":[
14      {
15        "type": "file",
16        "path": "bundles/main/streams/stream1.ts"
17      },
18      {
19        "type": "file",
20        "path": "bundles/main/streams/stream2.ts"
21      }
22    ],
23    "output":{
24      "ipPlayer": "127.0.0.1:7777",
25      "recording": false,
26      "fileOutput": "bundles/main/streams/injected/stream1.ts"
27    }
28  }
```

### 5.5.4 Credentials file

To access the commands database, the user must provide the MongoDB credentials as per the structure below. The file should be saved as *credentials.json*, alongside the configuration file. A private file has been created for testing purposes.

```
1 {  
2   "db": "",  
3   "collection": "",  
4   "uri": ""  
5 }
```

## 5.6 Results

### 5.6.1 System organization

The picture below shows the two stream generators (host) and the graphics recovery (replica), connected through the switch. The IPs have been assigned as in the image, while internet link is provided to access the remote database instance.

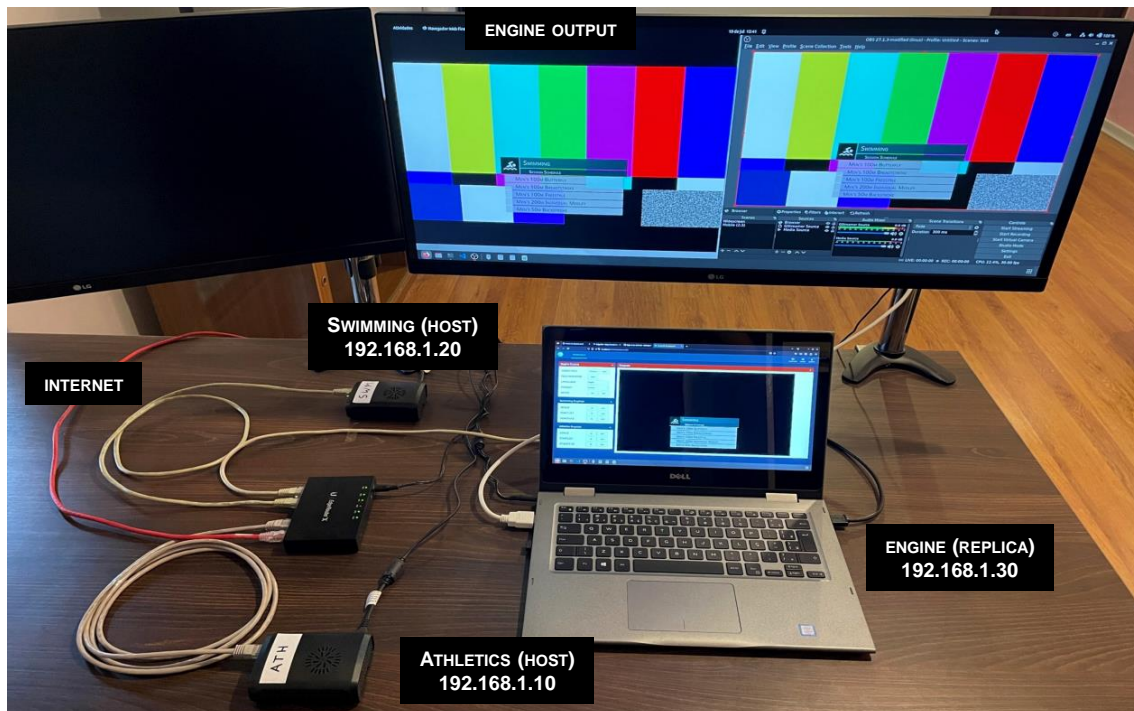


Figure 5.8: Picture showing two stream generators and the GFX recovery system.

## 5.6.2 Interface

The operational interface of the engine, developed as a NodeCG dashboard, is shown in Figure 5.9. On the left side, the engine control panel allows users to switch between streams, select the language for rendering the messages (replica mode), select the format of the graphic output and control the downstream keyer (if on or off). The following panels are loaded along with the bundles and are used to control the messages for each graphic package (swimming and athletics).

If the engine is operating as a host, commands sent using GUI buttons are translated onto SCTE notifications and injected into the clean feed. If the engine is operating as a replica, received messages trigger a luminous notification, indicating which operation is being mimetized (Figure 5.10).

Moreover, the graphics tab can be used to show all available templates built (currently four) and which one is on air, if any (Figure 5.20). The system allows more than one to be displayed at the same time if they are assigned to distinct *layers*. There is virtually no limit to the number of layers that can be used.

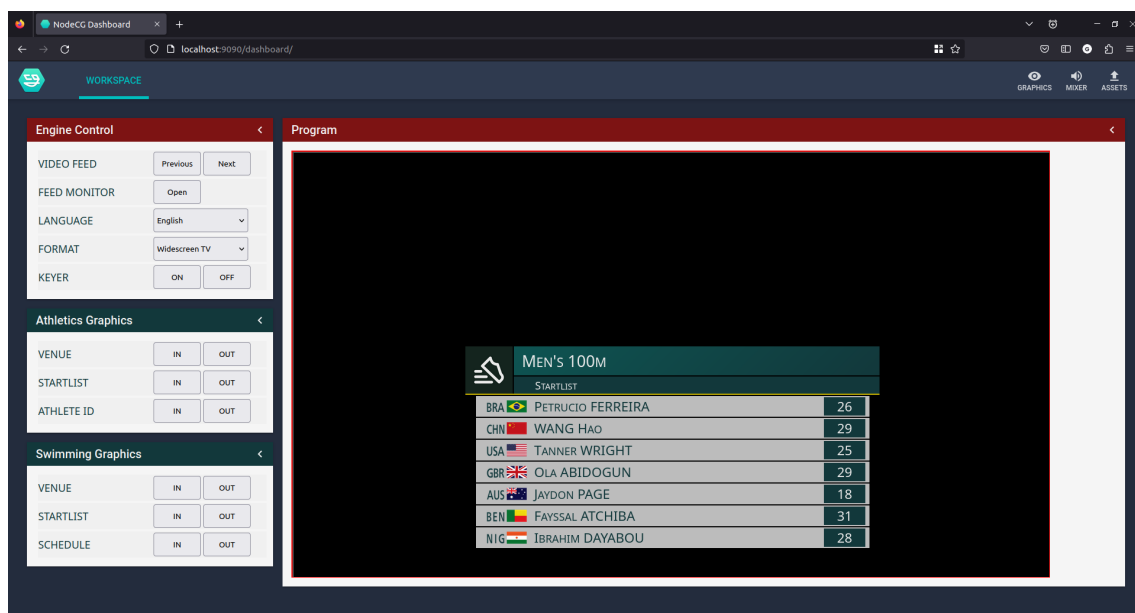


Figure 5.9: Graphic interface of the engine, showing control panels and output.



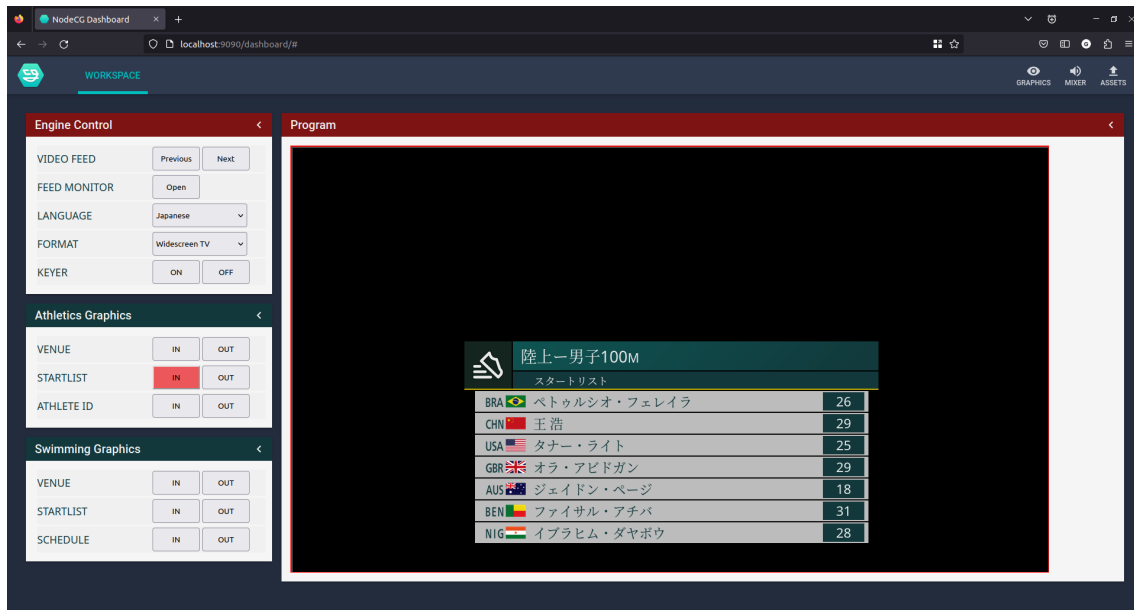


Figure 5.10: Replica engine triggering a luminous indicator corresponding to the command received from the ancillary space.

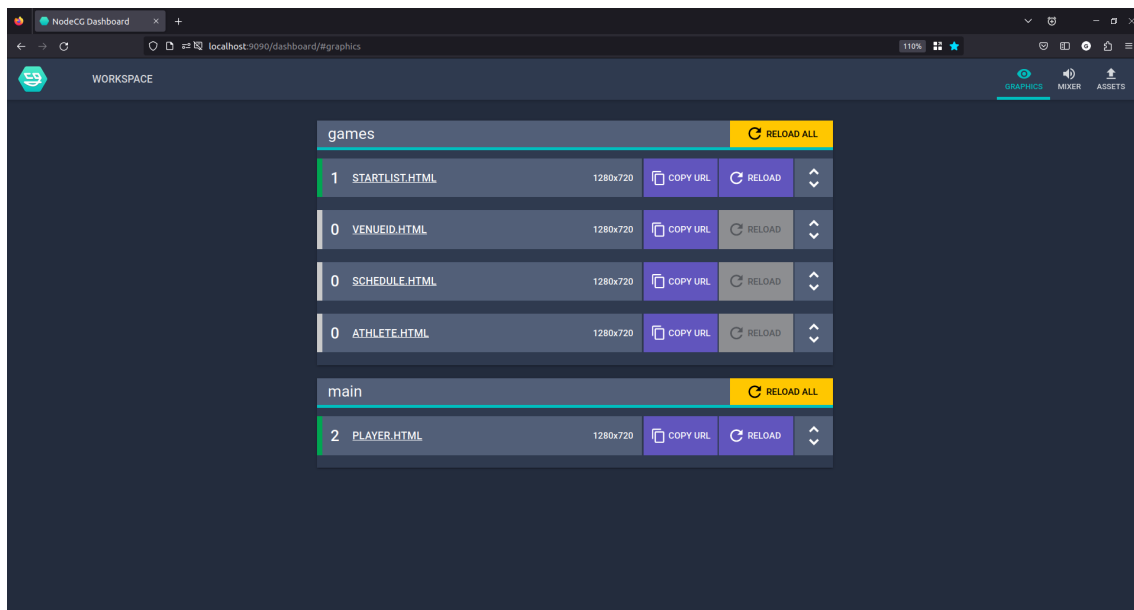


Figure 5.11: Graphics tab showing templates loaded and which one is currently being displayed (in green).

### 5.6.3 Database entries

The graphic messages have been filled with static test data and each template has its corresponding set. The engine prepares both the SCTE notification and the database entry with a randomly assigned ID for injection. The image below shows the semantics of one of these entries.

## db.commands

STORAGE SIZE: 44KB   LOGICAL DATA SIZE: 28.64KB   TOTAL DOCUMENTS: 40   INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes

Filter

Type a query: { field: 'value' }

```
_id: ObjectId('64b5ed9062f181c6143a70c1')
▼ info: Object
  eventId: 2012875654
  timestamp: 1689644432644
▼ template: Object
  domain: "GAMES"
  package: "athletics"
  graphic: "venueId"
  src: "/bundles/games/graphics/venueId.html"
  channel: "PGM"
  layer: 1
  command: 1
▼ data: Array
  ▼ 0: Object
    id: "PICTO"
    src: "pictos/ATH.png"
  ▼ 1: Object
    id: "EVENT"
    ▼ value: Array
      0: "Olympic Stadium"
      1: "国立競技場"
  ▼ 2: Object
    id: "PHASE"
    ▼ value: Array
      0: "Athletics"
      1: "陸上"
```

Figure 5.12: Database entry, accessed directly in the MongoDB platform.

Each message in the database contains information about the template used, the command triggered, layer, and the state of the downstream keyer if this has been operated. Any of these fields can be easily edited by an authorized user.

Furthermore, each graphic element that makes up the visual piece can have equivalent versions in other languages. To achieve this, the data of the component in question is written as an array, where each index corresponds to a language. In the project, it has been conventionally established that index 0 corresponds to English (also used as fallback) and index 1 corresponds to Japanese.

## 5.6.4 Graphics messages

The images below showcase some of the rendered graphic pieces overlaid on the video feed. This process is done in OBS scenes, which allows the overlay of the graphical output (*player.html*) onto the video stream. The simulation also demonstrates how the output would appear on mobiles with an aspect ratio of 2:3.



Figure 5.13: Startlist graphic, displayed in widescreen and English.



Figure 5.14: Startlist graphic, displayed in widescreen and Japanese.

By adjusting the parameters of the replica engine, multiple combinations of languages and output formats can be obtained. This demonstrates the system’s flexibility, while synchronicity is maintained with a relatively simple apparatus on the decoding side — based entirely on the video stream itself for triggering the arts.



Figure 5.15: Startlist graphic, displayed in adapted format for mobiles and Japanese.



Figure 5.16: Athlete ID graphic, displayed in widescreen and English.



Figure 5.17: Athlete ID graphic, displayed in widescreen and Japanese.

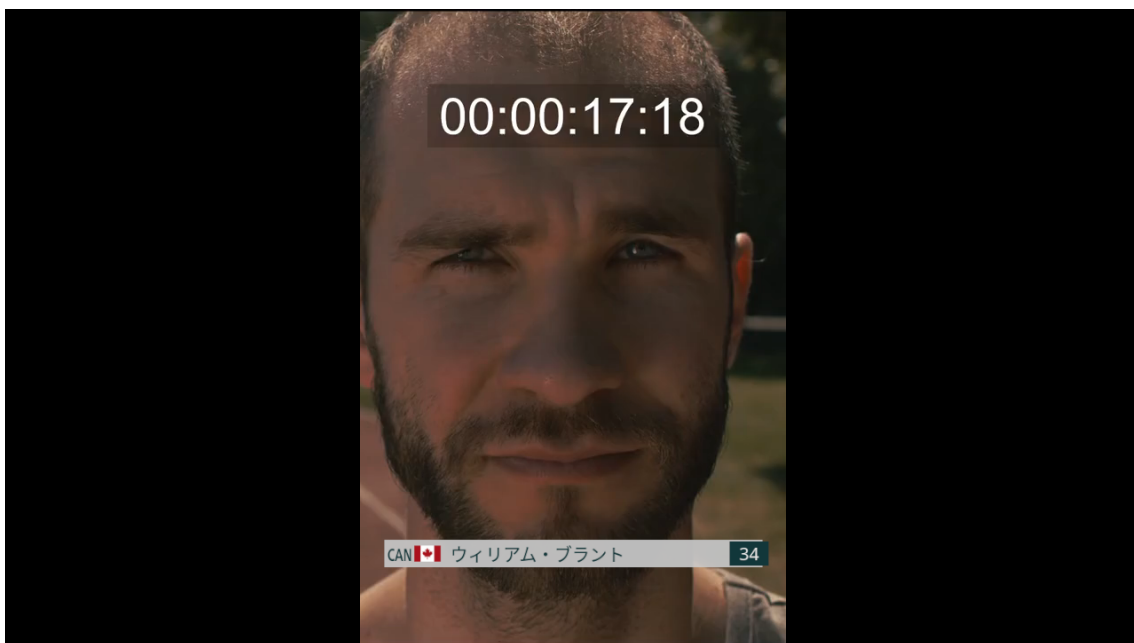


Figure 5.18: Athlete ID graphic, displayed in adapted format for mobiles and Japanese.

If the input signal is switched, the generator automatically starts capturing the corresponding SCTE notifications, leading to the rendering of graphic compositions for the new feed, as shown below. As expected, disruptions on the video decoding process are minimal, but eventually perceptible.





Figure 5.19: Schedule graphic, displayed in widescreen and English.

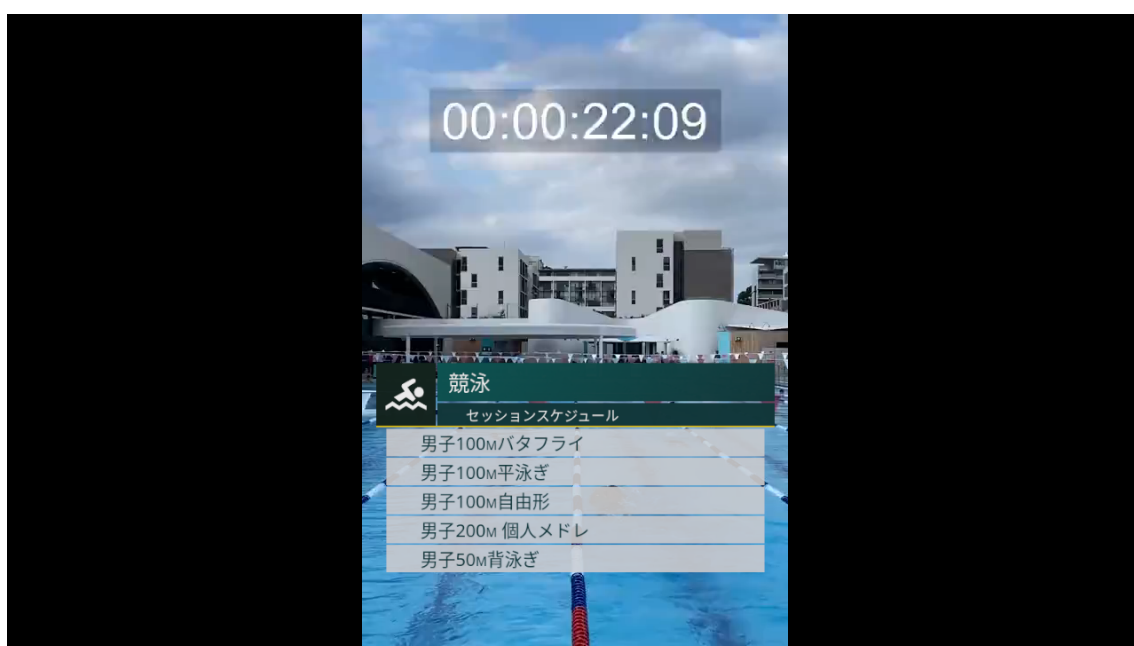


Figure 5.20: Schedule graphic, displayed in adapted format for mobiles and Japanese.

Configuration file for OBS scenes used is placed at the data folder, in *main bundle* directory. All code developed for the project, including the two SCTE-decorated feeds can be found in the *GitHub* repository [35].

# Chapter 6

## Conclusions

This work described an alternative method for representing graphics assets, introducing a new approach for carriage of data and metadata embedded in the media stream or using an external database. By separating the graphics essence from the video content, it becomes possible to reprocess and adapt visual elements and inner data for various applications, thereby creating new possibilities to enhance the audience's experience.

Chapters 2 and 3 delve into the analysis of current broadcast standards and protocols, providing readers with insights of workflows implemented by the industry, particularly in large-scale sporting events. Special attention has been given to the exploration of the ancillary data space, which emerges as the most logical resource to be investigated within the scope of this work.

Furthermore, the proposal for using SCTE-35 and SCTE-104 protocols, as presented in Chapter 4, has shown to be potentially advantageous in light of the mechanisms for monitoring, injecting, and extracting these messages. Since these are already established standards, current broadcast tools would be capable of supporting the implementation of the new workflow, whether in legacy SDI infrastructure or in the latest IP-based workflows.

The experimental implementation tested the use of an external database with a collection of editable commands and data sets, stored on a remote database instance. Additionally, two media files were created and used to simulate a live broadcast.

It was possible to demonstrate some scenarios in which the use of message IDs alone proved sufficient for retrieving graphic content. However, in more complex compositions where fields and elements are updated rapidly, the adoption of an alternative strategy becomes necessary.

In this regard, the use of standalone ancillary messages becomes an interesting subject for investigation in future research. While the option of employing private messages within the SCTE protocol exists, it is imperative to investigate how the semantics of these messages would be structured and evaluate the feasibility of implementing such a solution.

Finally, this work addressed key aspects of ANC message structure, traffic, and transmission across real-time and transport infrastructures. This focus was given due to the nature of the broadcast productions studied, in general live broadcasts with multiple feeds.

However, the new approach did not explore the possibilities of optimization and automation in other production workflows, particularly those focused on retrieving edited or ingested content in digital media. This scenario is more prominent in news and entertainment divisions, as well as in other platforms and channels with on-demand content consumption.

To address this, it is crucial to elaborate and discuss mechanisms for transporting messages in the ancillary data space (and consequently, the graphics essence) to the file-based domain and vice versa. Future works can delve deeper into this topic from the perspective of the guidelines defined by SMPTE 436M, which deals with the mapping of VBI data into MXF media. By doing so, it will be possible to have a new dimension of the proposal presented in this work, encompassing the entire broadcast production chain and its major pillars.



# Bibliography

- [1] “ST 274:2008 - SMPTE Standard - Television — 1920 × 1080 Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates”, *ST 274:2008*, pp. 1–35, 2008.
- [2] SCOTT, B., “The Story of TV Graphics”, <https://www.linkedin.com/pulse/story-tv-graphics-scott-barber>, 2016, (Access in October 2022).
- [3] “Deafness and hearing loss”, <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss>, 2021, (Access in December 2022).
- [4] OWENS, J., *Television Sports Production*. 5 ed. London, England, Routledge, 2015.
- [5] SALAZAR, R., “File-Based Workflows in Broadcasting”, <https://www.broadcastbeat.com/file-based-workflows-in-broadcasting>, 2021, (Access in December 2022).
- [6] DEVLIN, B., “MXF”, <https://www.tvtechnology.com/news/mxf-262167>, 2012, (Access in December 2022).
- [7] “ST 292-1:2018 - SMPTE Standard - 1.5 Gb/s Signal/Data Serial Interface”, *ST 292-1:2018*, pp. 1–20, 2018.
- [8] TEKTRONIX, “A Guide to Standard and High-Definition Digital Video Measurements”, [https://download.tek.com/document/25W\\_14700\\_6.pdf](https://download.tek.com/document/25W_14700_6.pdf), 2009, (Access in October 2022).

- [9] “Core Insights: Advances in 12G-SDI”, ”[https://www.aja.com/pdf/2020/Core\\_Insights\\_Advances\\_In\\_12G\\_SDI.pdf](https://www.aja.com/pdf/2020/Core_Insights_Advances_In_12G_SDI.pdf)”, 2020, (Access in December 2022).
- [10] “From SDI to IP: The Evolution of Distribution”, <https://www.smpste.org/blog/sdi-ip-evolution-distribution>, (Access in January 2023).
- [11] “SMPTE ST 2022: Moving Serial Interfaces (ASI SDI) to IP”, <https://www.smpste.org/webcast/standards-smpste-st-2022>, (Access in January 2023).
- [12] “Professional Video, Audio and Data over IP”, <https://www.smpste.org/standards/st2110>, (Access in January 2023).
- [13] “ST 2110-10:2022 - SMPTE Standard - Professional Media over Managed IP Networks: System Timing and Definitions”, *ST 2110-10:2022*, pp. 1–23, 2022.
- [14] “ST 2110-20:2022 - SMPTE Standard - Professional Media over Managed IP Networks: Uncompressed Active Video”, *ST 2110-20:2022*, pp. 1–23, 2022.
- [15] “ST 2110-30:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: PCM Digital Audio”, *ST 2110-30:2017*, pp. 1–9, 2017.
- [16] “ST 2110-40:2018 - SMPTE Standard - Professional Media Over Managed IP Networks: SMPTE ST 291-1 Ancillary Data”, *ST 2110-40:2018*, pp. 1–8, 2018.
- [17] EDWARDS, T., “RTP Payload for Society of Motion Picture and Television Engineers (SMPTE) ST 291-1 Ancillary Data”, *RFC 8331*, , 2018.
- [18] “ST 291-1:2011 - SMPTE Standard - Ancillary Data Packet and Space Formatting”, *ST 291-1:2011*, pp. 1–17, 2011.
- [19] “ITU-R BT.1364-3 - Format of ancillary data signals carried in digital component studio interfaces”, *ITU-R BT.1364-3*, , 2015.
- [20] “ST 336:2017 - SMPTE Standard - Data Encoding Protocol using Key-Length-Value”, *ST 336:2017*, pp. 1–36, 2017.
- [21] “ISO/IEC 13818-1 - Generic coding of moving pictures and associated audio information - Part 1: Systems”, *ISO/IEC 13818-1*, pp. 1–316, 2022.

- [22] ESTRADA, C. R. D., “Analizador de stream de TV Digital para o padrão ISDB-T”, Fevereiro 2008.
- [23] TSDUCK, “MPEG Transport Stream Toolkit User’s Guide”, <https://tsduck.io/download/docs/tsduck.pdf>, (Access in November 2022).
- [24] “ST 2038:2021 - SMPTE Standard - Carriage of Ancillary Data Packets in an MPEG-2 Transport Stream”, *ST 2038:2021*, pp. 1–7, 2021.
- [25] “ANSI/SCTE 35 - Digital Program Insertion Cueing Message”, pp. 1–101, 2022.
- [26] HELIKER, J., “SCTE 35 vs. 104 Explained”, <https://www.linkedin.com/pulse/scte-35-vs-104-explained-james-heliker/>, 2021, (Access in October 2022).
- [27] “ANSI/SCTE 104 - Automation System to Compression System Communications Applications Program Interface (API)”, pp. 1–125, 2022.
- [28] “ST 2010:2008 - SMPTE Standard - Vertical Ancillary Data Mapping of ANSI/SCTE 104 Messages”, *ST 2010:2008*, pp. 1–11, 2008.
- [29] “DPI Trigger Insertion: SCTE-104 and SCTE-35”, <https://eegent.com/support/resources/FAQs/DPITriggerInsertion:SCTE-104andSCTE-35>, 2022, (Access in October 2022).
- [30] “SCTE-35: Supplementary Information”, <https://gridshot.net/features/advertising-insertion/scte-35/supplementary-information>, (Access in January 2023).
- [31] “SCTE-35 XML schema document”, <http://www.scte.org/schemas/35>, (Access in January 2023).
- [32] “9950-EMDE-ANC 3G/HD/SD-SDI Ancillary Data Embedder/De-Embedder”, <https://www.cobaltdigital.com/products/3011/9950-emde-anc>, (Access in January 2023).
- [33] VIDEO, R., “Carbonite Black Setup: Ancillary Data”, <https://help.rossvideo.com/carbonite-02/Topics/Setup/Video/Anc.html>, 2023, (Access in May 2023).

- [34] “JSON Databases Explained”, <https://www.mongodb.com/databases/json-database>, (Access in January 2023).
- [35] COUTO, G., “gcgen”, ”<https://github.com/gdancouto/gcgen>”, 2023, (Access in June 2023).

# Appendix A

## SCTE-35 adapted XML schema

```
1 <splice_information_table
2   protocol_version="uint8, default=0"
3   pts_adjustment="uint33, default=0"
4   tier="uint12, default=0xFFF">
5   <_any in="_metadata"/>
6   <!-- Splice commands, only one of them is allowed -->
7   <splice_null/>
8   <splice_schedule>
9     <!-- One per splice event -->
10    <splice_event
11      splice_event_id="uint32, required"
12      splice_event_cancel="bool, default=false"
13      out_of_network="bool, required when splice_event_cancel is false"
14      utc_splice_time="YYYY-MM-DD hh:mm:ss or uint32, required when
15 splice_event_cancel is false and program_splice_flag is to be set"
16      unique_program_id="uint16, required when splice_event_cancel is
17 false"
18      avail_num="uint8, default=0"
19      avails_expected="uint8, default=0">
20      <!-- Optional -->
21      <break_duration auto_return="bool, required" duration="uint33,
22 required"/>
```

```

20     <!-- One per component when splice_event_cancel is false and
    utc_splice_time is not specified -->
21     <component component_tag="uint8, required"
    utc_splice_time="YYYY-MM-DD hh:mm:ss or uint32, required"/>
22 </splice_event>
23 </splice_schedule>
24 <splice_insert
25     splice_event_id="uint32, required"
26     splice_event_cancel="bool, default=false"
27     out_of_network="bool, required when splice_event_cancel is false"
28     splice_immediate="bool, default=false"
29     pts_time="uint33, required when splice_event_cancel is false and
    splice_immediate is false and program_splice_flag is to be set"
30     unique_program_id="uint16, required when splice_event_cancel is
    false"
31     avail_num="uint8, default=0"
32     avails_expected="uint8, default=0">
33 <!-- Optional -->
34 <break_duration auto_return="bool, required" duration="uint33,
    required"/>
35 <!-- One per component when splice_event_cancel is false and pts_time
    is not specified -->
36 <component component_tag="uint8, required" pts_time="uint33, required
    when splice_immediate is false"/>
37 </splice_insert>
38 <time_signal pts_time="uint33, optional"/>
39 <bandwidth_reservation/>
40 <private_command identifier="uint32, required">
41     Hexadecimal content
42 </private_command>
43 <!-- Splice descriptors, depend on splice command -->
44 <_any in="_descriptors"/>
45 </splice_information_table>

```

# Appendix B

## Cobalt Insertion Overview



### *SCTE-104 / SCTE-35 Insertion Overview and Methodologies Using Cobalt® Models*

Cobalt Digital Inc. • 2506 Galen Drive • Champaign, IL 61821 USA • 1-217-344-1243 • [www.cobaltdigital.com](http://www.cobaltdigital.com) • [support@cobaltdigital.com](mailto:support@cobaltdigital.com)

### Overview

SCTE-104 and SCTE-35 are standards that define protocols for ad insertion.

- **SCTE-104** markers are placed on **SDI baseband video**
- **SCTE-35** markers are placed in **transport streams (compressed content)**

The two standards are equivalent and have the same functionality – they just apply to different signal types. When appropriately licensed, Cobalt® ancillary data injectors, frame syncs, and cross-converters have the ability to place SCTE-104 markers on SDI baseband video. Cobalt® encoders can convert SCTE-104 within inputted SDI to SCTE-35 in the transport streams, and Cobalt® decoders can perform the inverse operation extracting SCTE-35 and embedding SCTE-104 in the outputted SDI.

### Methods for Signaling Injection of SCTE Messages

The Cobalt® devices that provide SCTE ad insertion support all need some sort of signaling from an outside traffic system to inject the SCTE messages at the desired time in the program content. The methodology options for this type of signaling are:

1. **Manual Insertion by an operator.** All the message parameters are manually entered in the GUI, and a button is pressed to cause the insertion. The Cobalt® proprietary *Reflex* protocol can be used to automate this.
2. **Automated Insertion Using GPIO.** All the message parameters are pre-defined, and a GPI triggers insertion. This type of insertion is useful for “hard breaks” where insertion of interstitials is fully automated.
3. **Using the Cobalt® Proprietary XML Interface.** This allows for frame-accurate insertion, based on timecode or UTC time, or immediate injection in a low-latency network.
4. **Using the Standard SCTE-104 over TCP Interface.** This allows for frame-accurate insertion, based on timecode, or immediate injection in a low-latency network.

Methods 1 and 2 are accommodated by the **+SCTE104** license. These methods are intended for use in simpler setups, where frame accuracy is not required, and the contents of the marker message are static and known ahead of time. Many traffic systems have GPIO interfaces for this purpose.

Methods 3 and 4 are accommodated by the **+SCTE104-FAST** license (which also accommodates methods 1 and 2). These methods are intended for situations where the contents of the marker message change dynamically, and/or frame accuracy is required. The support for SCTE-104 over TCP makes the equipment compatible with any standard traffic system, without the need for custom development.

Methods 1 thru 4 are available in a number of Cobalt® baseband video processors, and result in a SCTE-104 message being inserted in the output SDI signal. If SCTE-35 is required, an encoder can be added to the workflow to convert a SCTE-104 message to SCTE-35 in the transport stream. Conversion of SCTE-104 over SDI to SCTE-35 is a standard feature in Cobalt® encoders.

If there is a need to signal the insertion directly at the encoder, without using a baseband video processor, Cobalt® encoders will support method 4 above. This requires the optional **+SCTE104-TCP** optional license (this license is subject to future availability and is currently pending).