MASTER IN COMPUTER SCIENCE

Unstructured documents in EDIFACT

A survey of the inclusion of unstructured documents in the EDIFACT standard, with an introduction to CALS

Marchal, Benoît

Award date:
1994

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Academic year 1993-1994

Unstructured documents in EDIFACT

A survey of the inclusion of unstructured documents in the EDIFACT standard, with an introduction to CALS.

Benoît Marchal

'What is the use of a book', thought Alice, 'without pictures or conversations?'

Lewis Caroll

Thesis presented
in order to obtain the degree of
Licencié et Maître en Informatique
by Benoît Marchal
E-mail: 100345.354@compuserve.com

Printed on recycled paper.

Abstract

This work studies the inclusion of so-called unstructured documents in an EDI standard, the EDIFACT international standard in particular.

We will show that the notion of business documents, as known in EDI, needs to widen in order to encompass all sorts of documents exchanged amongst companies, notably including technical documents. Indeed EDI is traditionally thought as a commercial or administrative exchange but EDI can and must be enhanced to include other documents.

In the process we will review CALS (Continuous Acquisition and Life-cycle Support), a US DoD initiative popular for technical data exchange. A significant part of this work is dedicated to an introduction to the various CALS standards. In the end, we will see how to include the techniques developed by CALS in EDIFACT.■

Résumé

Ce travail est consacré à l'introduction de documents dit non structurés dans un standard EDI, en particulier le standard international EDIFACT.

Nous allons montrer que la notion de documents d'affaires, utilisée dans l'EDI, doit être élargie pour inclure tous les documents échangés par les entreprises, notamment les documents techniques. De fait l'EDI est traditionnellement perçu comme l'échange de documents administratifs ou commerciaux mais l'EDI peut être appliqué à d'autres documents.

Pour ce faire nous étudierons CALS (Continuous Acquisition and Life-cycle Support), une initiative du DoD (le ministère de la défense américain) populaire pour l'échange de documents techniques. Nous terminerons en voyant comment les techniques développées par CALS peuvent être appliquées à EDIFACT.■

Acknowledgements

In doing this work, we have contracted a debt with many people. Now we are pleased to thank them all for their help.

First Ph. van Bastelaer, our promoter, for guidance throughout this work.

Ray Walker, SITPRO's Chief Executive, and John Berge, Director of the EDI Standards department at SITPRO, who admitted us for a five months training period at SITPRO. In this dynamic environment, we learned a great deal — not only on EDI.

Also all the personnel of SITPRO, for their sympathy and willingness to help. There was always someone to assist us when we needed it. In particular we would like to thank the EDI section: John Berge, Peter Wilson, Kulbir Kaur and Harsha Karunaratne.

Kulbir and Harsha deserve special thanks for all the guidance they provided in early stages of this project.

Numerous other people have contributed to this work in a way or another. Any attempt to list them all is doomed to fail and would lead to oversights we would regret. But we remember them and want to thank them.

Contents

1 Introduction

Studies serve for delight, for ornament, and for ability.
Francis Bacon

The 3 October 93 an exhibition dedicated to the work of the French designer Philippe Starck ended in the London Design Museum. Its name was «Is Starck a designer?».

In ten years, Starck has become France's most celebrated designer and architect, and the world's most visible designer. What attracts both his ardent followers and those who recognise the products but ignore the name, is certainly the arresting forms that he creates, the wit and sheer originality of his work.

But Starck's work is remarkable not only for what he creates but also for how he creates. Speed is essential in Starck's working method, «*to capture the violence of the idea*».

The most audacious example of Starck's working practice is the Restaurant Manin, an architectural project in Tokyo. Starck never appeared on site until the building was officially opened. Preliminary rough sketches were passed down the fax between Paris and Tokyo. Once they were approved, a firm of Japanese structural engineers was called upon to figure it out.

> *It is a project of which I am very happy conceptually and very proud on a practical level, because the whole project was conducted in correspondence, via the fax, and I therefore never saw it before the opening day. [1]*

During interviews, Starck talks of a new world where only ideas, not people, would move. He is enthusiastic with this idea. Of course if one really wants to move he can, but why not sending a fax?

Although widespread, fax is a limited tool. Limited to transmission of paper sheet images. One can improve resolution, speed or size but fax will only *move* a sheet of paper copy.

Paper is a limited medium and we want our networks to exchange every kind of information. If Starck is right, if we are heading for a world where only ideas move, technical EDI will be part of it.

1. Time for a Change

New management techniques have been developed to compete successfully in today's market. Companies seek to reduce time to market and costs, improve quality and processes. Since they focus on different stages of the production process, these techniques vary but somehow they all centre on information availability. They make information availability a key to effective management, with availability meaning *where needed, when needed and in an appropriate form.*

For example, traditionally design and manufacturing are considered two separate activities, but the lack of communication between the various departments of a company means that each department must do what it thinks best for the others and sometimes (often), may be wrong. Therefore mistakes are common; resolving them takes time. If something goes wrong in manufacturing the product, then it will go back to the design phase where the defect will be fixed. The product then re-enters manufacturing until another defect is found.

Increasingly, the solution to this problem is to have the various people involved in the product working together throughout the whole product-life cycle, sharing their knowledge. This can significantly reduce the likelihood of an error. It requires new communications facilities, both to store data in some place where they are accessible to the whole team and to transfer them where they are needed. That is some sort of shared and, potentially distributed, multimedia corporate database.

> *...In a paper world documents often spend much of their life in someone's in-basket or out-basket. George might have finished his changes to the new operations manuals, but Sarah, the reviewer in the quality assurance department, doesn't know this because the document didn't make the morning delivery to her mailbox. While Sarah is wondering why her workload is so light, four other authors are preparing to send her documents later in the day — after an unproductive morning Sarah will be working overtime. Meanwhile, Larry, from advanced manufacturing is in a state of panic because he has a customer on the factory floor who wants to see how well the new operations manual reflects an actual operating procedure — but where is it? [2]*

Traditionally in organisations, computers handle structured information. Structured information can be processed automatically because by definition it is *structured* in a semantic way (such as an invoice). This is reflected in commercial and administrative EDI and traditional database packages.

Structured information, unfortunately, accounts only for about 10% of the total amount of information a company process. The other 90% are reports, documentation, manuals, etc. Just look at your desk! Most of the information is presented as text but also it might be graphics or sounds. Figure 1 illustrates this.

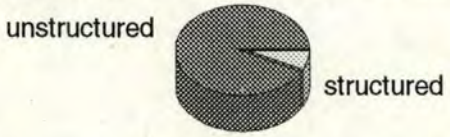


Figure 1: Structured Vs unstructured information in company

Currently, computers are very helpful editing such documents but they cannot process them the way they process structured information, i.e. without human intervention. Also databases have just start integrating it and the exchange amongst heterogeneous systems is a problem.

Therefore there is a need, a business need, to treat unstructured information efficiently with computers. This need is reflected in new knowledge/databases and multimedia bases which store and manipulate almost every sort of information.

In this work, we will concentrate on the use of EDI, i.e. the electronic exchange of business data, to communicate unstructured information. By linking communication techniques and multimedia base, one can prefigure a world of free information exchange where the appropriate information is immediately available to the one who needs it. That vision, not far away from Starck's dream, is also the basis of initiatives like CALS which we will review.

We will see how EDI needs to evolve to integrate those changes. When EDI is used to carry unstructured data, it is sometimes referred to as technical EDI, as opposed to administrative and commercial EDI. The latest we will abbreviate in commercial EDI.

We will then interest ourselves to the kind of document that need to be exchanged, mainly concentrating on engineers needs as the reader is supposed to be familiar with reports and other forms of documents. We will also study Interactive Electronic Technical Manuals as an example of modern information techniques used to represent reports.

Then we will have a brief look at the various technologies' issues relating to technical EDI. As promised we will review the CALS initiative with an overview of the various standards on which it is built.

We complete our study by the introduction of technical EDI in EDIFACT.

Knowledge of batch commercial and administrative EDI is assumed throughout this document.

2. A Word on this Document

This work originates during a training period at SITPRO, London. SITPRO, the Simpler Trade Procedure Board, is the UK's trade facilitation body. SITPRO is very active in the development and promotion of EDI. As part of its effort, SITPRO actively supports the development of the UN/EDIFACT international EDI standard.

Ray Walker, SITPRO's Chief Executive, was the first Western European Rapporteur for EDIFACT and Chairman of the Western European EDIFACT Board. He now chairs the United Nations WP.4/GE.1 committee which is responsible for the development and maintenance of UN/EDIFACT.

As we will see in the next chapter, there is a growing demand to incorporate technical documents within the EDIFACT standard. This work began as part of SITPRO's effort to meet this demand. We surveyed international standards for technical documents exchange; this document is a sequel of that survey.■

3. References

[1] Design Museum
Is Starck a Designer?
Design Museum, London, 1993

[2] Frank Gilbane
Integrating New Technologies: Workflow Systems
CALS Journal, Summer 1993, p. 65

[3] Technology Appraisals
Open Information Interchange
Technology Appraisals Ltd., UK, 1993

2 Technical Documents

If they say
Why, why, tell 'em that it's human nature
Michael Jackson

In this introductory chapter, we will now concentrate on a short review of some type of technical documents that might be exchanged with EDI. We feel it is important to know what a typical document is and how it is used, to understand the standards to represent it on a computer. We assume the reader is already familiar with paper reports or similar documents such as technical notices, so we will concentrate specifically on engineering documents. We will briefly review Interactive Electronic Technical Manuals as an example of new technologies used to treat unstructured documents. Then we will see why it is important to be able to exchange technical documents using EDI.

Many objects that surround us are engineering products. The cars we are driving, the road on which we drive them, the clock on our desks are engineering products, even the computer used to design them all is an engineering product. Engineering has become a key activity in today's economy.

> *The design process is the sequence of operations which links a concept, the idea of a thing existing only inside someone's head, to a description, the complete design of that thing which specifies it with sufficient accuracy and in sufficient detail for someone to go away and actually make one of them. [3]*

Products tend to be more and more complex and must be delivered following tighter schedules. This requires processing lots of information. Just think of the numerous parts that compose an aircraft. All of them were designed, manufactured and will have to be maintained. Not surprisingly engineers have turned to computers to help them manage such huge amounts of data.

1. Computer-Aided Design and Buzzwords

When thinking of technical data, one probably thinks of Computer-Aided Design (CAD) first. CAD is only one aspect, albeit an important one, of computer assistance to engineers. Other approaches include concurrent engineering (CE) where the design is conducted simultaneously by more than one team.

Manufacturing has also benefit from computer technology with techniques known as Computer-Aided Manufacturing (CAM). CAD and CAM are used together to link more closely the design and manufacturing processes. This gave birth to the acronym CAD/CAM (sometimes written CADCAM to emphasis on the close integration of both techniques).

Overall one can say without taking any risk that technical data exchange, integrated design or manufacturing, and global product life-cycle management are in-fashion.

But the field for their application is so broad that not a single system will use all the available techniques. By combining them, one can create lots of acronyms (indeed, that's what many authors do) although their meaning will often overlap. Each of those words intends to focus on the integration of two or more steps in the product life-cycle. Some are for standards, systems or just commonly used acronyms. We won't even attempt to list them all.

We will introduce the reader to CAD (and partly to CAM) in chapter 5 when we will consider standards for the exchange of CAD information.

2. Interactive Electronic Technical Manual

Although they are only in early stage of development Interactive Electronic Technical Manuals (IETMs) promise to be a better way to deliver technical documentation. The name is almost self explicit, IETMs are:

- technical manuals presenting technical information (TI) used by maintenance technicians;
- in electronic form which means a seamless use of electronic media from authoring to distribution and permits the use of animation or sound;
- interactive since they adapt the presentation of information to better suit the practical problems a technician is confronted to.

IETMs are now made possible by advances in information handling technologies like greater storage density, powerful microprocessors, high resolution displays, availability of electronic technical manual repositories through the advance in computer-aided engineering technologies.

They proved to be more efficient than traditional documentation, in areas like:

- technician training

> *For example, in a test at Miramar Naval Air Station, using inserted faults in the Navy's F-14A aircraft [1], not a single inexperienced technician using a conventional paper Technical Manual was able to isolate the fault without assistance.* ***All*** *[their bold] were able to isolate the fault with interactively displayed troubleshooting TI. In the same test, only 71.4% of the* ***experimented*** *technicians were able to isolate the fault with*

conventional TI; but, again, all were able to isolate the fault successfully using IETM material. [2]

- production costs through use of automated production methods and a coherent representation throughout the production process: electronic data can even be directly extracted from CAD systems, databases, etc.
- logistic management processes, e.g. storing, updating, etc.

  For example, the US Navy Aegis class cruiser Vincennes carries 23.5 tons of technical manuals. Note that replacing them by electronic data would rise 7.5 cm of water or alternatively permit it to carry 20,000 extra litres of fuel below the water line, when the manuals are kept above the main desk, which would make the ship more stable.

3. Business Data

So far, we have discussed technical information. In the previous chapter, we showed that it is business information and consequently should be integrated both in corporate databases and EDI. This does not mean we must forget commercial data. Commercial data can and should be part of the integration process. Indeed commercial EDI is often the starting point to so-called advanced management techniques, focused on the whole product life-cycle.

The important point is that business data is not limited to commercial EDI data. All technical documentation referring to a product is business data.

As companies exchange invoices or orders electronically, it is as important for them not to use slow, error-prone paper when they exchange technical documents. Consequently, the notion of business data in EDI needs to widen to encompass all data exchanged between companies, including technical.

4. Management

To cope with more intensive competition and world-wide recession, new management techniques have been devised which intend to reduce costs and lead time, improve quality and consumer satisfaction. They rely heavily on the sharing and the availability of information.

However, one must understand that those techniques are not specific to EDI, nor do they require EDI. Though EDI is often the enabler for such a managerial change. Also, EDI can be used without these techniques and still permits some significant gains but using EDI without changing the way business is done means ignoring most of the benefits of electronic trading.

Don't be fooled, the real change is managerial. Electronic exchange of, say, a drawing certainly creates certain difficulties, but having people from different departments working together is a bigger problem, especially when they have never worked in close partnership.

To benefit massively from the new communication technologies, a complete change in the way we conceive business is required. Better communications will only bring all the promised benefits if production is reorganised. Design, manufacturing and commercial teamwork should no longer exist in isolation but must cooperate.

The key words are *integration* and *communication*. Integrate the systems used to design, manufacture and sell a product or a service; and have all the people involve in the product life-cycle communicate. From a technical point of view, this is somehow a natural evolution. The real revolution comes from the managerial aspect.

5. Technical EDI in an EDI Standard

5.1. Structured and unstructured documents

When EDI is used to carry technical document data, we call it technical EDI. Since technical documents are business documents, as we saw, they conform to the definition of EDI:

> *EDI is the electronic transfer of computer processable data relating to a business or administrative transaction using an agreed standard to structure the data. [4]*

The main difference between technical EDI data and commercial EDI data is that the semantic of the information is no longer entirely included in the data. Therefore, those documents are often referred to as *unstructured document*.

The expression unstructured documents might be misleading. Of course even so-called unstructured documents are somehow structured, this is mandatory since computers manipulate them. Rather than *unstructured*, we should talk of *less structured*, i.e. less than so-called structured documents. What the expression unstructured documents really means is documents that cannot be entirely automatically processed, from creation to archival. Structured emphasises on automatic treatment.

Lest the importance of this observation be missed, let us be more direct and take two examples: an EDIFACT invoice as a structured document and an IGES drawing as an unstructured document. An EDIFACT invoice can be automatically processed throughout all its life cycle, from its production by the seller system to its receipt and treatment by the customer system, no human intervention is required†. An IGES drawing, on the contrary, will require human intervention for its creation and most of its further processing. The design activity has and will likely defy automation since it's a creative activity. And this makes all the difference.

5.2. Technical Data Exchange

5.2.1. Need for Standards

We have already identified the need for the exchange of engineering data amongst companies. Unfortunately since the systems that produce engineering data were developed by independent enterprises, each uses its own format and exchange is not directly feasible.

The most trivial way for two companies to exchange technical data is to use the same systems (same software, same release and same hardware). This is not always possible, particularly if a company has to trade with various partners, each using their own systems.

Another solution is to re-encode systematically the data from one system to another. This is tedious, error-prone and can be almost impossible when the information gets complex.

A better solution is to agree on some neutral format understood by systems of both companies. This has the further advantage that the same data can be used by a variety of tools, each performing a highly specialised task. Furthermore, it allows for easy migration from one system to another.

To avoid an exponential growth of the number of format, which would deserve communication, some sort of standard must be devised. A strong opposition to the standardised approach came from the graphics industry where standards are often perceived as the lowest common denominator amongst systems and therefore seldom used. Nevertheless standards are a prerequisite to open exchanges.

† It does not mean that the process will always be automated. Some companies prefer to print EDI messages and process them manually.

5.2.2. Standard Levels

When engineering data have to be exchanged, we can recognise the need for 3 levels of standards:

- the standards for the presentation of information which existed prior to computers; they ease the reading of information coming from various sources. This is especially used with drawings;
- the standards for the computer representation of data, e.g. the neutral formats;
- the computer communication standards: they allow the actual electronic exchange of data between two or more independent systems.

5.3. UN/EDIFACT and Technical Documents

An EDI standard provides some mechanism for at least two distinct functions:

- the formatting of computer data;
- the control of the exchange.

The control mechanisms exist in EDI standards, but there is no formatting rules suitable for technical documents. However there is no need to design new representation standards, rather EDI can rely on the work already done in this field. So technical EDI simply needs to use the control mechanism of existing EDI standards with formatting rules suitable for technical data.

Therefore EDI appears as the federating agent of all business data exchange whatever they may be. We can see the various databases as a logically single information base exchanging data, when required, with EDI.

How can we carry technical documents in an EDIFACT interchange? We will see that some mechanisms must be found to provide transparent inclusion of data in a non EDI native format, i.e. a binary stream of data.■

6. Reference

[1] Joseph J. Fuller, Theodore J. Post & Anne S. Mavor
Test and Evaluation of the Navy Technical Information Presentation System (NTIPS), F-14A Field Test Results, DTRC-88-036
US, September 1988

[2] Joseph J. Fuller & Samuel C. Rainey
The Interactive Electronic Technical Manuals
CALS Journal, Winter 1992, p. 63-69

[3] A. J. Medland & Piers Burnett
CAD/CAM in practice
Kogan Page, UK, 1986

[4] SITPRO & PFA
The UN/EDIFACT Workshop - Class Notes
SITPRO & PFA, UK, 1993

3 CALS

Experience keeps a dear school but fools will learn at no other.
Maxim prefixed to "Poor Richard's Almanac", 1757

There is little knowledge on technical document exchange in the EDIFACT community which, until recently, was mainly concerned with commercial and administrative documents.

It does not mean that the transfer of technical document is a new area. Other communities have been involved in such transfers for years. The EDIFACT people can and ought to learn a lot from these experiences. One of the most famous (if not the most famous) such community is the CALS community. CALS, which stands for Continuous Acquisition and Life-cycle Support (formerly Computer-aided Acquisition and Logistic Support), is an initiative launched by the US Department of Defense (DoD) in 1985.

Throughout this document we will draw from CALS experience. This chapter and the next three are entirely devoted to a (partial) study of CALS and the techniques it has developed. We start our trip in the CALS land in this chapter with an introduction to the initiative and its development. The reader interested only in the technical aspects may choose to skip this chapter altogether.

1. Historical Perspective

Over years CALS has evolved and has grown in popularity until it reached its present status. This section describes the evolution of CALS.

CALS was initiated in 1985 when the US Deputy Secretary of Defense William H. Taft released what is known as the Taft Memorandum of 24 September 1985. CALS as we now know it was officially launched by the Taft Memorandum of 5 August 1988. Appendix A is a copy of the latter.

In 1985 CALS stood for Computer Aided Logistic Support. By 1987 it has become Computer-aided Acquisition and Logistic Support to reflect the growing importance of the acquisition of information while retaining the original acronym. In 1989 the growing importance of concurrent engineering (CE) leads to the inclusion of CE in CALS. This was reflected in a new acronym: CALS/CE — although CALS remained the preferred acronym. Recently, in 1993, CALS became Continuous Acquisition and Life-cycle Support to reflect its strategic rather than technical emphasis. In the meantime EDI was included into the CALS program.

CALS started as a DoD initiative to facilitate electronic exchange of information between the DoD and its suppliers; applying computer technology to the specification, design, ordering and maintenance of its weapons systems in order to reduce lead time, reduce cost and improve the quality of weapon systems. Many western countries have launched similar CALS programs.

One of the problem the DoD faces for the maintenance of its weapon systems is the availability of accurate and reliable information. Weapon systems are particularly challenging in this respect due both to their generally long life-cycle and to their inherent complexity. Some weapon systems are used over very long period of time, some live more than 30 years. They have to be maintained throughout those years despite the many problems inherent to long life cycles: the engineers who designed the systems may have retired or died, the technology will evolve to the point where it becomes impossible to order original parts and new (hopefully compatible) parts are used for maintenance. This does not go without pain as two parts are rarely fully compatible.

> *We may build a product once. But over its 30-years life, all the maintaining and upgrading that's done is like tearing down and rebuilding it many more times. [1]*

Another problem the DoD faces is the huge number of suppliers it deals with: 300,000 all around the world. The DoD buys not only missiles and guns but also suits, cars, armchairs and soft drinks! At present it employs 1000 procurement officers.

To give the reader an idea of the amount of data involved, suffices it to say that a single weapon system can generate up to a million pages of documentation, split over 3,500 technical manuals. Of this million, roughly 20 percent need to be adapted every year. The US Navy alone holds more than 200,000 separate manuals.

Clearly the traditional paper based process has reached its limits. CALS addresses all these issues by applying new information technologies to the management of product documentation — promising improved readiness and reduced costs.

Various cost/benefit analyses have been conducted. The interested reader is referred to them. Such an analysis is presented in chapter 3 of V. Daniel Hunt's *Enterprise Integration Sourcebook* [2]. Similar analyses can be found in Joan M. Smith's *Introduction to CALS* [5] and many other sources.

CALS is not a government solution to a government specific solution. The documentation nightmare is not the privilege of the government. The industry faces the

same challenge and increasingly turns to CALS or CALS-like techniques to solve it. CALS techniques can and have been successfully applied by the industry.

To name but a few, Rockwell based its Automated Logistics System (ALS) on CALS standards. ALS reduces maintenance time and costs for Rockwell aircraft and space-related products. General Motors too is using CALS-like techniques. Says Ian McEwan, Executive-In-Charge of the General Motors NAO Validation/QRD Center:

> *...as we installed more and more workstations, and proceeded to install local area networks, wide area networks and then, eventually, our corporate network, users quickly focused on their real problem. They could not exchange data with their customers and suppliers. While they now had physical connectivity, they could not read the data they received without massive manipulation. They could not send out data without long consultations with the person receiving the data, even when using the same hardware and software. [3]*

There is a business need to exchange technical information but this exchange is painful without a coherent standardisation program, like CALS. The CALS program enables open exchange of data. It does not force people to use a particular system but frees them from incompatibility problems.

Also one cannot help noticing the similarities between CALS and the new standard drafted by the automotive industry to address its communication needs. This standard is based on SGML. McEwan adds:

> *... Because the CALS concept is no longer confined to the defense industry. It is internationally recognized as a realistic approach for the management and exchange of information during product and process development, for the exchange of manufacturing information, for providing technical manuals and maintenance records, and for safe disposal. [3]*

Overall it is felt that, even if it has to be adapted to particular needs, CALS is a good basis for open exchange of technical documents in the industry and in the government. The choice of standards is sane and complete.

2. CALS Vision and Path

The CALS vision as defined by the US Defense CALS Executive is:

> *An integrated data environment created by applying the best commercial technologies, processes and standards for the development, management, exchange and use of business and technical information among governmental and industrial enterprises.1 [4]*

CALS aims at producing a single database, the Integrated Weapon System Data Base (ISWDB), which will contain all information on every weapon system used by the DoD and which will be accessed by the government and authorised industrial partners.

Of course we mean a logically single database. The size, the complexity, the number of partners involved call for a distributed approach where data is spread across multiple physically distinct databases interconnected in such a way as to act as a unique entity. The database is physically distributed (i.e. implemented as a set of databases located on different places) but present a single face to the user, which is fooled into thinking that he accesses one database only.

To achieve this goal (ambitious by current standard), CALS faces two distinct challenges:

- the availability of data in electronic form;
- the integration of those data in the shared database.

Until recently the preferred exchange mean between the DoD and its contractors was paper. Surely most of the information was on a computer somewhere in the world but in formats unsuitable for exchange (like proprietary format of systems unsupported by the DoD). Information was therefore converted to the lowest common denominator, namely the infamous paper. After the exchange, it may be re-entered on DoD computers before being made available to the end user. A process which was slow and error-prone.

With CALS the DoD wants to raise the lowest common denominator to electronic medium so that the information can transit directly from computer to computer. This is very similar to the evolution from non-EDI to EDI for commercial and administrative documents.

Alas not all information the DoD processes exist on electronic form. There are older systems for which maintenance information exists solely on paper (or other non electronic medium like microfilms or aperture cards). Since, for evident economical reasons, the DoD cannot afford to throw away all its weapon systems, CALS must incorporate existing systems too. We will see that one of the CALS standard specifically targets quick conversion of legacy data to electronic form.

Once information is available on electronic medium, the shared database can be built and data made available. This too raises some interesting problems which are out of the scope of this document. The interested reader is referred to other sources like Amjad Umar's book [6] for more information on distributed databases.

Recognising that converting documents and building a database are two distinct problems, CALS is divided into 2 phases:

- phase I focuses on the availability of electronic documents. It builds interfaces to allow data exchange between the DoD and its contractors. This phase focuses mainly on common conventions for data representation.

  There are some benefits inherent to having information in electronic form and being able to distribute it through electronic medium, even in the absence of a shared database. Phase I also intends to rip as much of these benefits as is possible;
- phase II is concerned with the actual building of the database and the integration of data acquired during phase I.

Lets the reader might be confused by the somehow misleading denomination, we want to stress that the two phases are not strictly sequential; rather they are interspersed. Phase II is not due to start once phase I is completed. The two phases should progress in parallel — admittedly at different paces.

Of course, it is useless to create the database (phase II) before there is some data to store within (phase I). Furthermore while phase I uses current technologies, phase II technologies are still under development. So there is some sequencing between phase I and II due to differences in nature and technical mastership. Phase I is more advanced than phase II but prototypal implementations of phase II have started although phase I is far from being completed.

Figure 1 is a classical† scheme to show how data interchange progress from a non-CALS situation to phase I and phase II.

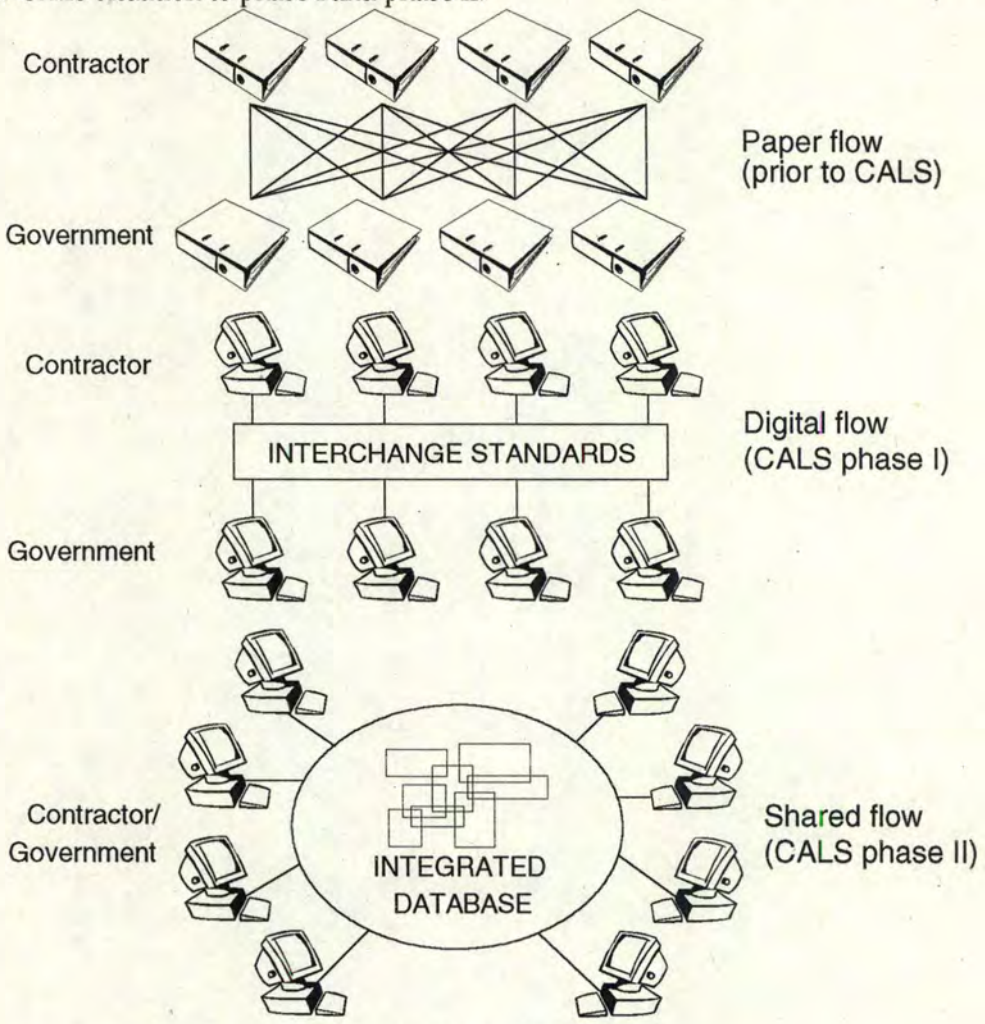


Figure 1: CALS development

Prior to CALS, links were numerous, difficult to manage. The same data typically existed in 5 different locations, even worse it has often been entered as many times with great risk of errors. The process was slow and error-prone. Moreover delays existed between the revision of a document and its availability to end users. It was not uncommon to have various copies at different levels of revision on different locations. This too leads to many errors and problems.

With the introduction of phase I standards, the flow of information is simpler and more manageable. Data is available more rapidly since it is not re-entered for every copy. Typing errors are reduced. Delays between the correction of a document and its distribution are reduced.

In the last step, phase II, the information will be accessed through a shared database. Delays will be reduced to an absolute minimum since a correction will be immediately reflected in the database and made available to all users.

As we already noticed, there is a strong parallel between phase I and EDI. In this document we will draw heavily on CALS phase I to see how and which techniques should be included in an EDI standard to prepare it for the exchange of technical documents.

---

† This scheme is such a classic of the CALS literature that we felt oblige to put a copy of it somewhere. We feared this document would have appeared uncomplete otherwise.

In the next three chapters we will set forth for a study of CALS standards. In chapter 6 and 7 we will see that those standards (or other equivalent standards) need not to be included in the EDIFACT standard for technical EDI.

Nevertheless, we feel it is important to devote two entire chapters to their study because we target readers informed of commercial EDI and willing to learn of the adaptation to technical EDI. Such a reader is presumed ill-at-ease with standards for technical documents. So the study was felt important as it:

- gives the reader a good background on technical document;
- illustrates the use of international standards in technical documents and exchange.

It is our hope that the study will:

- improve the reader understanding of what technical data might be;
- show some example of current techniques for technical document exchange to the reader unaware of this subject;
- help the reader making informed choices based on a good understanding of the problem.

CALS serves perfectly our purpose because the complexity and variety of weapon systems is such that it has selected a variety of standards which cover most domains relevant for technical exchange.

3. CALS Activities

CALS is active in five areas:

- standards;
- technology development and demonstration;
- weapon system contracts and incentives;
- DoD systems;
- management.

3.1. Standards

Standards are essential for the success of the CALS vision and the creation of the ISWDB. When two people want to exchange electronic documents they must agree on a common convention. Indeed CALS has selected open standards in every area pertinent for technical documents. The next section describes major CALS standards.

In the standard arena, CALS is concerned with:

- data interchange standards which provide common rules for digital encoding of documents. CALS phase I focuses on this category of standards. Refer to the next section for a description of currently available standards;
- data management and access standards are the focus of phase II. They will provide common definition of data elements, their relationship and access rules for the IWSDB;
- application guidance: the CALS program offers guidance (currently in the form of the MIL-HDBK-59B military handbook) to guide managers who implement CALS and help them achieving greater efficiency.

Two other aspects of data interchange standardisation are explicitly not covered by the CALS program. They are:

- functional requirement standards which define information and capabilities required by the DoD. They are the responsibility of functional managers;
- communication protocols to use networks instead of physical medium (like tapes). They are developed by specialists such as the Defense Communication Agency.

3.2. Technology Development and Demonstration

CALS supports the development of integrated database technologies and other technologies required to achieve its vision of an integrated data environment. CALS has set up a test network and finances various tests.

3.3. Weapon System Contracts and Incentives

The success of CALS depends on its actual adoption in contracts between the DoD and the industry. As part of its CALS effort, the DoD encourages investments by the industry in integrated processes. For example it gives priority to contractors who comply to CALS when delivering data.

3.4. DoD Systems

In order to be able to receive, manage, store and later retrieve and access information delivered in electronic form, the DoD has to modernise its infrastructure. This is also part of the CALS initiative.

3.5. Management

CALS is a joint initiative of the DoD and the industry. This is reflected in the management which is handled by both DoD and industry representatives. The interested reader is referred to Joan M. Smith's *Introduction to CALS* [5], chapter 2 for a comprehensive introduction to CALS management.

From the very beginning of the CALS initiative, the DoD has adopted an open approach. For example, the DoD did not use its power as a significant customer to impose its own set of standards but selected international versions whenever possible. This openness of mind is probably for a significant part in the success of CALS within the industry.

4. CALS Standards

What matters for us is less the CALS initiative self than the techniques it puts in practice. The techniques are based on standards for technical documents representation. We will therefore study the key CALS standards in the next three chapters. This section will present them and show how they relate to each other.

CALS standards sit at layers 6 (presentation) and 7 (application) of the OSI model (Open System Interconnection).

Apart from MIL-STD-1840B those standards are always based on international standards or US national standards when no equivalent international standard is available.

International standards are by definition very general since they must adapt to a wide variety of situation. They are doomed to be used all around the world by persons from different cultures and must fit amazingly different needs in the widest combination of environments possible. They must be open and malleable.

The malleability of international standards is their strength but also their weakness. The price to pay is that international standards must offer options, usually have conditional parts and even (intended) ambiguities.

Therefore, before using the international standard for actual exchange, the two parties of the exchange must agree on a common understanding. These agreements have various names: subsets, profiles or protocol suites.

The standardisation effort of CALS is twofold:

- select appropriate international standards if available, US national ones otherwise, participate in their development;
- publish an interpretation of the standards, what is known as the CALS standards, to be used when exchanging data with the DoD i.e. it standardises on standards.

CALS sets a framework which is particularly well suited for technical documents.

4.1. Military Standards

4.1.1. MIL-STD-1840B

Automated Interchange of Technical Information (MIL-STD-1840B) is the federating agent of CALS. This standard prescribes the organisation of files in an exchange. We will study MIL-STD-1840B in chapter 6.

4.1.2. MIL-28000 Series

Apart for MIL-STD-1840B, the CALS standards are in the MIL-28000 series. At present there are four standards in the MIL-28000 series. We will review them all in chapters 4 and 5.

MIL-D-28000A (amendment 1)

Digital Representation for communication of Product Data: IGES Application subsets and IGES Application Protocols (MIL-D-28000A) organises the exchange of CAD information. It is based on the ANSI standard for product data exchange, IGES.

MIL-M-28001B

Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text (MIL-M-28001B) is built on an ISO standard: SGML. It establishes a convention for the exchange of texts (manuals, guides, reports, etc.).

MIL-R-28002B

Raster Graphics Representation in Binary Format; Requirements for Quick conversion of documents (MIL-R-28002B) defines a raster format based on CCITT group 4 fax standard.

Scanning pages (which produces raster images) is a quick and cheap path for conversion of paper documents to electronic forms. This standard is specifically intended for rapid conversion of existing documents expecting little or no changes.

MIL-D-28003A

Digital Representation for Communication of Illustrative Data; CGM Application Profile (MIL-D-28003A), which is based on the ISO CGM standard, is used for the exchange of illustrative images, mainly as part of SGML documents.

In the CALS environment, most illustrations come from CAD models and could therefore be coded in IGES. Unfortunately IGES produces very large files for a practical usage. CGM is the preferred standards for illustrations.

5. CALS and EDI

The relation between CALS and EDI is not always clear. Depending on whether the author has an EDI background or a CALS background, CALS is considered an extension of EDI or EDI is considered part of CALS.

EDI people, who until recently were mainly interested in commercial and administrative documents, feel that the exchange of technical documents and their inclusions in EDI messages are natural extensions of EDI. They are right.

CALS people, on the other hand, believe the exchange of commercial and administrative documents along with technical documents is a natural complement of their strategy. CALS has recently issued statements to announce it will now include EDI. This is perfectly legitimate.

Whatever the point-of-view however it is clear, and that is what matters, that EDI and CALS are getting closer to each other. Both communities are looking at each other. This is coherent with what we said in chapter 2: technical documents are part of business documents and ought to be exchanged side by side with commercial and administrative records.

6. What Next?

The next three chapters are devoted to the study of CALS standards. But, apart for MIL-STD-1840B (studied in chapter 6) which is specific to the DoD, we will focus on the international versions of the standards, i.e. the broader versions and not their particular CALS interpretations. We hope this will help in applying CALS to another area of document exchange.

As we already notice it is not uncommon that industries develop their own subsets or profiles based on the same international standards as CALS. It is logical: industries operate in another environment than defence and have particular requirements which require specialised subsets. The fact that the same base standards are used shows that CALS choice was sane. If we study the international standards we are better prepared for this sort of application.

SGML is the subject of chapter 4. SGML is an ISO standard for the exchange of textual and compound documents. It is the basis of MIL-M- 28001B.

Chapter 5 is concerned with graphics standards:

- general purpose graphics with a study of the ISO CGM standard which is the basis of MIL-D-28003A;
- raster graphics with an introduction to the CCITT group 4 standard which is the basis of MIL-R-28002B;
- product model with some explanations on the ANSI IGES standard which is the basis of MIL-D-28000A.

We will also adopt a prospective view on product model and consider the emerging international Standard for Exchange of Product Data (STEP). STEP is fundamental to phase II and is considered such a major improvement over current standards that it deserves some consideration in this document if only for the sake of completeness.

7. Further Reading

We have only sketched the CALS approach. The interested reader is referred to Joan M. Smith's *Introduction to CALS* [5] for a complete presentation of CALS, including management issues and standards. A valuable reading for anyone interested in CALS and in data exchange is the *CALS Journal* which features both technical and managerial aspects of present and future approaches therefore covering in a complete and effective way the CALS initiative. The *CALS Journal* is available from: CALS Journal, 14407 Big Basin Way, Saratoga CA 95070-9808, USA.■

8. References

[1] Joseph R. Goss and James Brunke
Rockwell Implements CALS
CALS Journal, Winter 1993, p. 42-45

[2] V. Daniel Hunt
Enterprise Integration Sourcebook
The Integration of CALS, CE, TQM, PDES, RAMP, and CIM
Academic Press, San Diego (US), 1991

[3] Ian McEwan
Product Data Exchange in a Global Manufacturing Economy
CALS Journal, Winter 1993, p. 25-27

[4] Office of the Defense CALS Executive
CALS Definition and Vision Statement
September 21, 1993
in CALS Journal, Winter 1993, p. 11

[5] Joan M. Smith
An Introduction to CALS:
The Strategy and the Standards
Technology Appraisals, UK, 1990

[6] Amjad Umar
Distributed Computing, a Practical Synthesis
Prentice Hall, New Jersey (US), 1993

Appendix

Appendix A: the Taft Memorandum of 5 August 1988

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS DIRECTOR, DEFENSE LOGISTICS AGENCY

SUBJECT: Computer-aided Acquisition and Logistic Support (CALS)

To achieve productivity and quality improvements, my September 1985 letter on CALS set the goal of acquiring technical data in digital form (rather than paper) for weapon systems entering production in 1990 and beyond. We have now taken a major step toward routine contractual implementation. The Department of Defense (DoD) has coordinated with industry the first in a series of CALS issuances of national and international standards for digital data delivery and access. These standards have been published in MIL-STD-1840A, "Automated Interchange of Technical Information," and supporting military specifications. The CALS standards will enable either digital data delivery or government access to contractor-maintained technical data bases.

Effective immediately, plans for new weapon systems and related major equipment items should include use of the CALS standards. Specifically:

- For systems now in full-scale development or production, program managers shall review specific opportunities for cost savings or quality improvements that could result from changing weapon system paper deliverables to digital delivery or access using the CALS standards.
- For systems entering development after September 1988, acquisition plans, solicitations, and related documents should require specific schedule and cost proposals for: (1) integration of contractor technical information systems and processes, (2) authorized government access to contract or data bases, and (3) delivery of technical information in digital form. These proposals shall be given significant weight for their cost and quality implications in source selection decisions. The CALS standards shall be applied for digital data deliverables.

DoD components shall program for automated systems to receive, store, distribute, and use digital weapon system technical information, including achieving the earliest possible date for digital input to DoD engineering data repositories. These systems shall be configured or adapted to support the CALS standards. Plans for CALS implementation and productivity improvements will be addressed in Defense Acquisition Board and Major Automated Information System Review Council acquisition reviews, and in corresponding Service and Agency reviews.

Each application decision shall be made on its own merits with respect to the productivity and quality improvements expected at either prime contractor or subcontractor level. The Under Secretary (Acquisition) will issue further guidance on contract requirements, such as CALS options, in invitations for bid; opportunities and safeguards for small business and other vendors and subcontractors; government and contractor incentives; and funding mechanisms for productivity-enhancing investments in automation and CALS integration by defense contractors.

I believe that CALS is one of the most important and far reaching acquisition improvements we have undertaken. I would appreciate having the Assistant Secretary (Production and Logistics) receive your plan of action within 90 days, including identification of systems where opportunities exist for cost savings or quality improvement through application of CALS technology, the investment required to achieve these benefits, and proposed schedules for implementation.

William H. Taft, IV

cc: Under Secretary of Defense (Acquisition) Assistant Secretaries of Defense

4 Compound Documents

A programming language is a manifest from its creator declaring what's good and bad in programming. The good becomes a feature, the bad an error.

Robert Jervis

In this chapter we study a format for the exchange of compound documents, namely the Standard Generalized Markup Language (SGML, formally ISO 8879†). Compound documents are documents consisting of texts, graphics and, maybe, other media. They are often referred to as texts although, strictly speaking, they include non-textual elements.

These are probably the documents we know best because we meet them every day. They are the letters, books, lists, memorandum, reports, newsletters, advertisements, catalogues, notes, diaries, agendas, etc. which pass on our desks every day (sometimes they even stop) or cluster onto our shelves.

For years, standard character sets (such as ASCII or ISO 646) were all the standard one needed to exchange texts between computers. In the post-Macintosh era, in the world of computer assisted publishing this is no longer true. A modern text format must be concerned with layout, for example fonts or character position.

A compound document groups and organises data coming from various sources therefore its format often acts as federating agent for data presented in various formats. It must provide facilities for non-textual information, mainly graphical one (logos, charts, etc.).

Ideally such a format will be flexible enough to cope with new reading habits like hypertexts. Their use will probably spread in the near future (remember the IETMs introduced in chapter 2) but we do not want to have to re-encode our documents for every new tool. The format should also be able to integrate new media when they

† Also known as BS 6868, CEN/CENELEC 28879 or AS 3514. These standards (British, European and Australian respectively) are strictly identical to the ISO standard. SGML forms the basis of CALS MIL-M-20081B standard.

will be available. Today's computers are definitely graphical ones and they quickly learn how to deal with sounds and animation.

Numerous incompatible standards for text exchange exist. Some are proprietary like the Rich Text Format (RTF) from Microsoft, used mainly for exchange between word processors, or Digital's Compound Document Architecture. PDLs† (Page Description Languages) such as Adobe's PostScript® might turn out to be a good choice in some cases. And there is the Open Document Architecture (ODA, ISO 8613) and SGML alternatives, both being international standards from ISO.

They originate from different worlds and emphasise on different aspects. ODA comes from the business world and manages both the actual appearance of documents (the layout) and their logical structure. To date ODA has yet to be implemented in real products.

SGML was developed in the publishing industry but its use is wider than publishing. Unlike ODA, SGML concentrates on the document's structure which accounts for most of its originality and power; it is a very flexible standard with a multimedia extension. Also, unlike ODA, it is widely implemented. CALS elected it as the standard for text documents exchange.

Compound documents standards are important because they federate other exchange standards. SGML is a very important standard for CALS. Hence we will study SGML more than the other CALS standards. However, we will not attempt to provide a complete coverage of SGML in such a short document. Some good books do that better than we can (Cf. 'Further Reading' section at the end of this chapter). Our goal, in writing this chapter, is to provide the reader with an introduction to some key aspects of compound documents in general and SGML in particular. What we consider key aspects, in other words the features we chose to present, may appear arbitrary. The choice was guided by a threefold goal. It is our hope that after having read this chapter, the reader will:

- understand what generalized markup is;
- understand what SGML can be used for;
- understand what compound documents are and how to represent them on a computer.

Incidentally, the reader will have a good start for further learning on compound documents and SGML.

† A PDL is a language which permits the presentation of a complete, formatted page image to output devices. "Virtual paper" is a good metaphor for PDLs. The most famous one is PostScript® which has given birth to an ISO standard.

Background on Generalized Markup

1. Background on Generalized Markup

1.1. Document Structure and Attributes

1.1.1. First Look

SGML is all about documents structure so we will first interest ourselves with the document structure and how it relates to text formatting. Hopefully, the reader understanding will refine as he progresses along this paper. We will first have an informal introduction by means of an example. The example, which we will use all the way through this paper, will be the fictitious memorandum depicted in figure 1 (inspired by [6]):

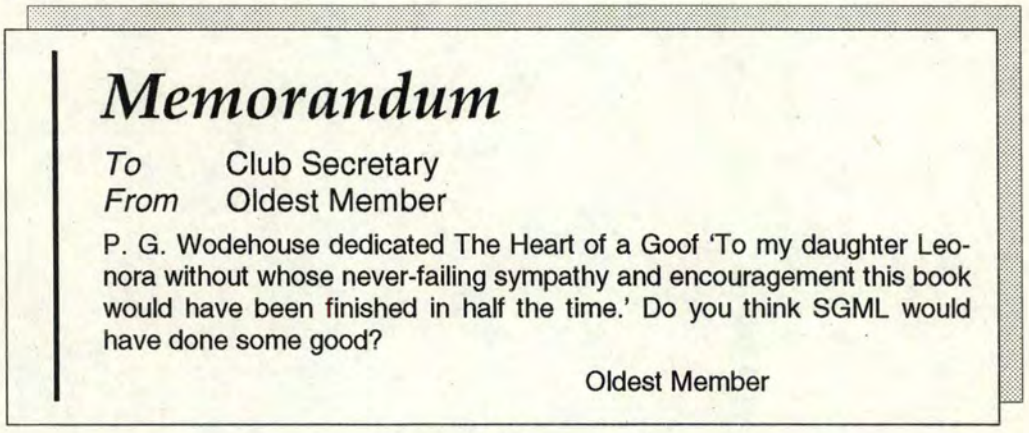


Figure 1: Memorandum

It looks like a classical memorandum, probably very similar to most memorandum one usually comes across.

As we are interested in document structure, what matters for us is that, at first sight, this memo is composed of four distinct elements:

- a document title;
- an introduction which states the sender and recipient names;
- the body text;
- a signature.

These elements are organised in relation to each other following a structure. For example the title names the document and the signature ends it.

If we examine the document more closely, we find that the body text itself consists of various elements, namely:

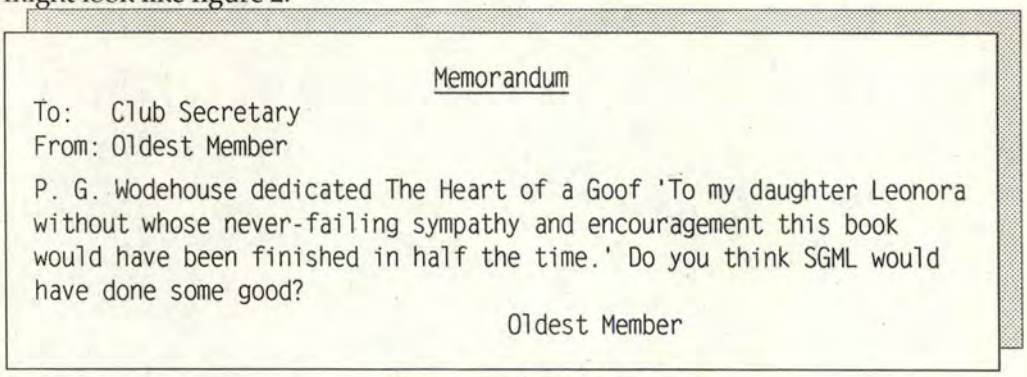
- a paragraph;

which includes:

- a quotation.

Suppose now that the memo was produced with a typewriter. Typewriters, apart from being less pleasant to use than word processors, offer fewer options for formatting. They generally offer only one typeface. With a typewriter, our memorandum

might look like figure 2:



Memorandum

To: Club Secretary
From: Oldest Member

P. G. Wodehouse dedicated The Heart of a Goof 'To my daughter Leonora without whose never-failing sympathy and encouragement this book would have been finished in half the time.' Do you think SGML would have done some good?

Oldest Member

Figure 2: Memorandum

Although the formatting is completely different, poorer than the previous example, the structure has not changed. The memorandum still contains:

- a title;
- an area for the sender and recipient name;
- the body text which further divides in:
  - a paragraph;

  containing:
  - a quotation;
- a signature.

In the first version of the document, the author chooses to reflect parts of the document structure in the formatting. However, the typewriter dictates another choice. The formatting is influenced also by the taste of the author. For example, the quotation is not highlighted in any of those two versions.

What we have learned in this example is that a memorandum is composed of various elements organised according to a document structure and that this structure might be partially reflected in the text formatting. Partially only because it depends on the author's taste and the tools he used to print the document.

Our discovery is easily generalised to any sort of document.

We have established intuitively the difference between the document *structure* and the *formatting* or text attribute. Let us refine this.

1.1.2. Second Look

Any document, in the abstract, is composed of elements which are organised in relation to each other according to some structure.

If we consider a book these elements are the parts, chapters and paragraphs. If we look at a further level of detail we will find sentences or words. How far to go in decomposing the document in its elements depends solely on what to do with the result of this decomposition.

For SGML, a text consists of interrelated elements, each contains characters which serve a particular purpose. A character can be member of more than one element, for example it can be in a word which is itself in a quotation, within a paragraph that is part of some section of a chapter.

We improve a document's readability if we format the text according to its structure. For example, in this paper, all section titles are in a different typeface to mark visually the beginning of a new section.

The only direction to go is from structure to formatting (i.e. the attributes depend on the structure of the text, however the opposite is not true), in other words an element occupies a function in a text whether or not this function is highlighted by

typographic attributes. To take a trivial example, a title names a section even if its typeface is not different from the rest of the text.

This does not mean that the formatting is not important and that only the structure matters. For someone reading a text, good formatting when constantly applies is an effective help because it clarifies the structure of the text. As we will see, SGML can help in this topic.

The actual formatting depends on many things like the system on which the document is produced, the taste of the author or the intended reader.

Electronic edition using microcomputers means that more people can now produce publication quality documents. Most documents are now electronically edited. Unfortunately each system has its own publication oddities. To exchange documents amongst those systems, some system independent format is required.

1.2. Markup History

The easiest way to understand generalized markup is probably a historical study of electronic markup i.e. the progression from *procedural markup* to *generalized markup* through *generic coding*.

1.2.1. Mark-up

Mark-up originates in the publishing industry. When the author and the editor have agreed on the text of a future book, the manuscript is passed to a typographic designer. The designer will define the appearance of the book in consultation with the editor.

The typographic designer will supply guidelines for how the text should be made up into pages. The design information, known as mark-up, will be used by the typesetter to add the instructions that will produce the correct typeset pages. In traditional editing this usually involves retyping the text.

1.2.2. Procedural Markup

Similarly text processing and word processing systems require the user to supply additional, formatting-related, information with the text. This information is the markup which is stored, as special codes, embedded within the text.

> *Markup is the term used to describe codes added to electronically prepared text to define the structure of the text or the format in which it is to appear. (Markup is spelt as one word when applied to electronically prepared copy to distinguish it from the traditional form of editorial or design mark-up, which is hand-written on the copy.) [7]*

The user of an electronic publishing package, whether a typesetter formatting a book on a professional electronic publishing system or a secretary typing a letter with a word processor, inserts commands in the stream of the text. These commands request the output device with some formatting operations. The user chooses those attributes in a three steps process (although he may not perceive it as such) [2]:

- he analyses the information structure and other attributes of the documents i.e. he identifies each separate meaningful element;
- he determines the processing information controls that will produce the format desired for that type of element;
- he inserts the chosen controls in the text†.

This process is often referred to as procedural markup because the markup is effectively some procedure, embedded within the document, to be executed by the

† Luckily, this does not mean that the user actually has to type some codes (like in the good old days of Wordstar). In all but the most primitive systems, these codes are inserted by the software when the user selects the appropriate menu option. Embedding some code can be as easy as clicking on a menu!

output device. This closely parallels the traditional mark-up activity. The main difference is that markup is electronically stored.

This solution has a number of limits [2]:

- information about the document's structure is lost.

  Since more than one sort of element can use the same typographic attributes, information about the structure of the document is lost. For example, in this paper, both quotation and highlighted words are italicised therefore no program can automatically creates an index of highlighted words based on the formatting;

- it is inflexible.

  Any change to the formatting rules implies manually changing the codes wherever they are in the document;

- it is not portable.

  The codes can be more or less system dependent which reduces portability — the less portable solution being the inclusion of instructions of the target output device. Even the reliance on some sort of metrics or on the availability of a particular typeface limits portability;

- it is time-consuming, error-prone and requires a high degree of operator training.

1.2.3. Generic Coding

Systems evolved with the introduction of 'macros' into a process known as 'generic coding'. A macro is a technique by which the controls are replaced with a call to some external formatting procedures. A generic identifier (GI) or tag is attached to each text element. A formatting procedure is further associated with each tag. A formatter processes the text and replaces GIs by the actual formatting instruction required by the output device. The benefits over procedural markup are threefold:

- the procedure set can be changed to adapt to various output devices, making it more portable.

  If a set of procedures is associated with each potential output device, the formatter can output the text on any device without any change to the markup;

- the scheme is more flexible.

  To change the formatting of a piece of text (for example change all the title from bold to italics), it suffices to change the procedures, not the markup i.e. the text remains unchanged;

- the markup is closer to describing the structure rather than the attributes.

  Users tend to give significant names to the tags, e.g. 'Heading' or 'Quotation' is preferred to 'B007' or 'X12', clearly recognising the predominance of the structure over the formatting. Therefore some previously impossible automatic processing of the document is now possible. It is possible to have a program who automatically looks for every word tagged as 'Highlighted' to create an index.

1.2.4. Generalized Markup

Generalized markup extends generic coding. A Generalized Markup Language (GML) requires two characteristics from the markup [2]:

- markup should describe a document's structure and other attributes, rather than specify processing to be performed on it, as descriptive markup needs to be done only once and will suffice for all future processing;

- markup should be rigorous, so the techniques available for processing rigorously defined objects, like programs and databases, can be used for processing document as well.

The second characteristic differentiates generalized markup from generic coding. It means that the markup language can express other attributes than the GI (like attaching unique identifiers to elements, etc.).

2. Standard Generalized Markup Language

The Standard Generalized Markup Language (SGML) is an internationally recognised GML. It is based on the early work done by Dr Goldfarb from IBM. The interested reader will find a complete history of the standard in appendix A.

SGML is one of the most popular ISO standard. SGML was part of the US DoD Continuous Acquisition and Life-cycle Support (CALS) initiative as early as 1985 before its first publication as an international standard (1986). The DoD however already adopted the standard in 1983.

There are two ways to design a markup standard. The standard can be:

- a standard set of tags to be used by every standard compliant document;
- a standardised language that permits the definition of application specific tags.

The first alternative is clearly unsuitable for an international standard. It is unrealistic to hope to be able to design a universal tag set without putting unacceptable constraints on the author. It is doubtful that a single structure, whatever smart, can satisfy the need of all authors for all potential documents. SGML follows the only viable way for an international standard, the second one.

> *...this standard was designed to formalize the way in which documents are prepared. It does this not by laying down a set of rules saying 'this is how you should code documents', but by formalizing a set of rules that can be used by document originators to say 'this is how I have coded my document'. [7]*

The SGML mechanism that enables this is the Document Type Definition (DTD). The DTD is a description, written in SGML, of the structure of a document i.e. authorised tags and the way they are organised.

SGML is an *enabling* standard, not an a complete document architecture. The force of SGML is that is a *language* to describe documents — in many respects similar to programming languages. It is therefore open and flexible. Standards committees, industry groups and others use its functionalities to build applications and document structure. Although SGML does not impose a tag set, it is particularly well suited for the development of standard sets for specific industries. These sets target a particular application and therefore legitimately impose a structure.

An example of such a standard is MIL-M-28001B which consists of DTDs for technical manuals conforming to MIL-M-38784C. MIL-M-38784C is a standard for technical documents submitted to the DoD. Another example is the design by the Association of American Publishers (AAP) of three DTDs (for books, articles and serials). This was the first big application of SGML.

Basic SGML

3. Basic SGML

The reader should now have a clear understanding of what generalized markup and SGML are. It is time to study the actual markup of a SGML document.

3.1. SGML Documents

A SGML document starts with a DOCTYPE statement which declares the document type and associated DTD.

There are four ways to declare the DTD, namely to follow the DOCTYPE keyword and associated document type with:

- the keyword SYSTEM and a system specific identifier of the DTD (normally a filename):

  ```
  <!DOCTYPE Foo† SYSTEM "c:\sgml\dtd\foo.dtd">
  ```

- the keyword PUBLIC and the identifier of a public DTD:

  ```
  <!DOCTYPE Foo PUBLIC "ISO//Foo//1994">
  ```

- the DTD itself between brackets:

  ```
  <!DOCTYPE Foo [...]>
  ```

- a mix of one of the first solutions with the last one to supplement some local declaration to the DTD:

  ```
  <!DOCTYPE Foo SYSTEM "c:\sgml\dtd\foo.dtd" [...]>
  ```

These four statements are document declarations. One document declaration must start every SGML document.

The first two solutions allow DTD to be shared amongst various documents. A public DTD is normally maintained by some standardisation authority while a system DTD is presumed to be local to the system where it is used.

The main difference is for document exchange, as a public DTD can be assumed to be known by every SGML systems and therefore need not to be exchanged while a system DTD, being specific to the local system, often has to be exchanged with the document.

Notice that the DOCTYPE statement is comprised between '<!' and '>'; these symbols enclose every declaration statement in SGML. Every SGML declaration or tag must be enclosed in some special strings for the parser†† to differentiate markup from the actual document.

To mark some part of the text as an element, it suffices to enclose it by:

- an opening tag, i.e. the element name defined in the DTD between angle brackets (< >);

  ```
  <Memo>
  ```

- a closing tag, i.e. the same element name enclosed by '</' and '>'.

  ```
  </Memo>
  ```

There is a limit size of 8 characters to the name of the element.

---

† Foo is a common nickname to replace any identifier. In an actual SGML document, Foo would be replaced with the document type. Foo is not part of the SGML standard.

†† "Parser" is the traditional name of the software which analyses the SGML document. More on this in section 5.

Here is what the memorandum example looks like when coded with SGML:

```
<!DOCTYPE Memo SYSTEM "c:\sgml\dtd\memo.dtd">
<Memo>
<To>Club Secretary</To>
<From>Oldest Member</From>
<Body>
<Para>P. G. Wodehouse dedicated The Heart of a Goof<Quote>To my
daughter Leonora without whose never failing sympathy and
encouragement this book would have been finished in half the
time.</Quote> Do you think SGML would have done some good?</Para>
</Body>
<Sign>Oldest Member</Sign>
</Memo>
```

As stated by the DOCTYPE, the document is a memo, the actual DTD being stored in the file 'c:\sgml\dtd\memo.dtd' (this example assumes that an MS-DOS filename or compatible is used for system identifier). It seems logical to store the DTD in a separate file as we probably write more than one memo.

The first tag after the document declaration must have the same name as the name used in the document declaration, e.g. <Memo>. This is the compulsory *base document element* and it makes active the document declaration.

The rest of the document consists simply of the various elements enclosed in their opening tags and closing tags: the To, From, Body and Sign elements.

The Body element itself consists of a Para (paragraph) element which encloses a Quote (quotation) element.

Notice that the memorandum title is nowhere to be found in the SGML document. Similarly, there is no 'To:' and 'From:' labels before the originator and receiver elements. This is not required as it can be deduced from the markup. This has nothing to do with the document structure and is best left to formatting routines.

If we use preprinted forms, the formatting routines will skip space accordingly but if we print on a blank sheet of paper, they will print a title.

> *In many applications creating a document can be thought of as filling in a preprinted form. For example, a memo will normally be output on a sheet of paper that has been preprinted with the name of the company and special fields for the entry of the names of the sender and recipient, and possibly the subject and date of the memo. These preprinted fields do not, as such, constitute part of a document. For preprinted sheet forms the document is simply the text added to the preprinted sheet. [7]*

Similarly, the quotes are not keyed and considered presentation matter. This is usually a good idea for some systems might be able to produce different symbols for opening and ending quotes while others cannot. By looking carefully at the memorandum in the first section, one will notice that the word processor version uses different symbols (‘ ’) while the typewriter provides only one sort of quote (').

3.2. Document Type Definition

SGML is not as a standard tag set. Rather than imposing its own text structure on the author, SGML provides authors with a mechanism to describe their structure. This mechanism is the Document Type Definition (DTD). The DTD is sometimes also referred to as a SGML application.

Each SGML document starts with a DTD which defines not only which element can appear in the document but also the order in which they can appear and whether an element can be part of another one.

Few SGML users will have to write their own DTD. DTDs writing is a difficult task to be left to specially trained engineers. It is important however, to understand what is a DTD to understand when SGML can be used and when it cannot.

An element declaration starts by the keyword ELEMENT followed by the element name. The declaration ends by the element content — enclosed in '<!' and '>' like any SGML statement.

The element content is:

- a model group i.e. one or more element names between brackets;
- a primitive content token, noted #PCDATA (which stands for parsed character data). This means that, at that point, the element can contain text which has been checked by the SGML parser to ensure that any embedded tags or entity references are resolved.

Alternatively, the element content can simply declare the type of data allowed within the element. Valid data types are:

- CDATA (character data): valid SGML characters that need no further processing. This usually corresponds to data to be processed by another processor than the SGML parser. For example PostScript® instructions.
- RCDATA (replaceable character data): text, character references and/or entity references (entities are studied later) that resolve to character data;
- EMPTY: the element content is automatically generated by the program.

Connectors indicate the order in which elements must appear in a model group:

- &: the 'and connector' specifies that the elements or model groups appearing on either side of the ampersand must both appear in the document but in any order;
- ,: the 'sequence connector' specifies that the elements or model groups appearing on either side of the comma must both appear and in the same order in the document;
- |: the 'or connector' specify that either the element or model group on the left-hand side or on the right-hand side of the vertical bar must appear in the document but not both.

Occurrence indicators indicate the repetition of the elements. Notice that occurrence indicators have a higher priority than connectors.

- ?: the 'optional occurrence indicator' indicates that the model group or element that precedes the question mark may appear once or not at all;
- +: the 'required and repeatable occurrence indicator' indicates that the element or model group preceding the plus can occur more than once;
- *: the 'optional and repeatable occurrence indicator' indicates that the element or model group that precedes the asterisk may occur zero or more times. Wherever it occurs, #PCDATA is always considered to have an asterisk indicator.

Remember the memorandum introduced in section 1: the figure 3 is a tree which parallels the structure we identified.

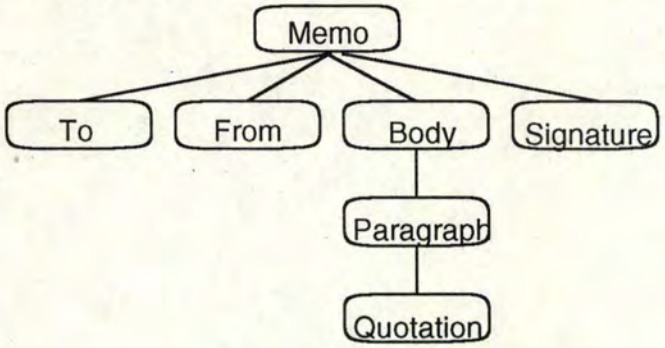


Figure 3: Memorandum structure

It is often a good idea to draw a tree with the document structure when writing a DTD. A tree provides the designer with a clearer, graphical representation of the structure. This is particularly relevant with complex documents.

Now it suffices to express this structure in a SGML DTD. For our sample memo, the DTD might look like:

```
<!-- DTD for simple office memorandum (partial)-->
<!--       ELEMENTS  CONTENT (EXCEPTIONS)        -->
<!ELEMENT Memo       ((To & From),Body,Sign?)     >
<!ELEMENT To         (#PCDATA)                    >
<!ELEMENT From       (#PCDATA)                    >
<!ELEMENT Body       (Para)*                      >
<!ELEMENT Para       (#PCDATA|Quote)*             >
<!ELEMENT Quote      (#PCDATA)                    >
<!ELEMENT Sign       (#PCDATA)                    >
```

The DTD simply expresses, in a formal language, the structure of the document.

Let us study the declaration of some relevant elements:

```
<!ELEMENT Memo ((To & From),Body,Sign?)>
```

the notation above indicates the relationship between the elements of a `Memo` (i.e. the `Memo` consists of a group of two elements, `To` and `From` or `From` and `To`, followed by a `Body` element and an optional `Sign` element).

If we turn to `Body`'s definition:

```
<!ELEMENT Body (Para)*>
```

we find that a `Body` element is composed of a (potentially empty) sequence of `Para` (paragraph) elements. `Para` is defined as:

```
<!ELEMENT Para (#PCDATA|Quote)*>
```

i.e. a `Para` is a sequence of free text and `Quote` (quotation) elements. Notice how the priority rule (occurrence indicators have a higher priority than connectors) applies. If the connector priority was higher than the occurrence indicator, this notation would mean a sequence of either text or quotation.

All the other elements are declared as `#PCDATA` i.e. they basically contain text. Notice that the further we go down in the hierarchy (the closer we get to the tree leaves) the more likely we find `#PCDATA`.

4. Minimization

Reducing the amount of markup required is an important matter. Firstly if the markup is to be added manually, it saves some typing. However even in the most likely case where the markup is automatically added, it is important to reduce markup.

Why? Because the markup is an overhead added to the text. It is costly in terms of storage space. This becomes even more important when the document is to be exchanged through a network.

SGML provides five optional mechanisms to reduce the required markup collectively known as minimization. We won't study them all; instead, we will interest ourselves with the most commonly found ones.

4.1. OMITTAG

The best way to reduce markup is simply not to insert tags. This is the underlying idea behind the OMITTAG feature.

Looking at the memo in section 3, one will see that some of the markup is redundant. Every element starts with an opening tag and ends with a closing tag. In most case, the closing tag of an element is immediately followed by the opening tag of the next element. Clearly, one of them can be omitted.

OMITTAG is a feature which, when enabled, allows the opening or closing tag of an element to be omitted provided their presence can be unambiguously deduced from the tags of the surrounding elements.

When this feature is enabled, two extra characters are required in the declaration of an element. They must appear between the element name and content, separated from each other, from adjacent name and content by one or more spaces.

The first character is the letter O (the letter 'O', not zero) if the opening tag can be omitted. It is the hyphen symbol (-) if not. Similarly, the second character is O if the second tag can be omitted and hyphen if not.

For example, if we enable the OMITTAG feature, we must change the declaration of Para and allow the closing tag to be omitted, which writes:

```
<!ELEMENT Para - O (#PCDATA|Quote)*>
```

4.2. SHORTAG

SHORTAG, when enabled, permits part of the markup to be omitted from the tag. An empty tag (</>) is assumed by the parser as referring to the previous complete tag.

4.3. SHORTREF

SHORTREF is a feature which allows short references to be used instead of complete entities. Short references are characters, or strings of characters, that provide a reference to an entity within the document.

Short references are defined in a short reference map. For example, one can associate a blank line to the beginning of a new paragraph.

By default, those three minimization options are enabled in SGML. We will see in the syntax section how to enable or disable minimization options.

4.4. Name choice

Another way to gain on storage room is the intelligent definition of the application i.e. to use short names for tags.

> *It is often a good rule to abbreviate the most commonly used tags to their first letter and, when it is no longer possible, use two letters name for less common elements, three letters name if required...[9]*

Intelligent application definition can actually save storage.

4.5. Minimized Memorandum

As an illustration, we will now rewrite the memorandum example with the minimization techniques enabled. We will also choose minimal names for elements.

4.5.1. Document

```
<!DOCTYPE Memo SYSTEM "c:\sgml\dtd\memo.dtd">
<Memo>
<t>Club Secretary
<f>Oldest Member
<b>
<p>P. G. Wodehouse dedicated The Heart of a Goof <q>To my daughter
Leonora without whose never-failing sympathy and encouragement this
book would have been finished in half the time</>. Do you think SGML
would have done some good?
<s>Oldest Member
</Memo>
```

The only element whose name has not been abbreviated is the base document element <Memo>. This is required, since its name must match the DOCTYPE declaration but it is not very important for this element occurs only once in the whole document!

Notice that no close tags are required for most elements since their ends can always be inferred from the beginning of the following elements. Notice also that a close tag is required for the quotation as its end cannot be inferred from the beginning of another element.

The last tag is the close tag of the base document element (</Memo>). One can argue that it can be deduced from the end of the file. However it is good practice to add it especially if the document can be included in another one.

One might worry that abbreviations are less readable and require higher degree of training from the various users. This is not a problem if a software tool is used to mark the text. The software will present the user with a more readable description of the elements where p, for example, is renamed as paragraph.

Minimization has reduced the size of the document from 394 characters to 340. This means more than 10% space saved!

4.5.2. Document Type Definition

Of course, the DTD must have been adapted.

```
<!-- DTD for simple office memorandum      -->
<!-- ELEMENTS   MIN  CONTENT (EXCEPTIONS) -->
<!ELEMENT Memo  - -  ((t & f),b,s?)         >
<!ELEMENT t     - O  (#PCDATA)              >
<!ELEMENT f     - O  (#PCDATA)              >
<!ELEMENT b     - O  (p)*                   >
<!ELEMENT p     - O  (#PCDATA|q)*           >
<!ELEMENT q     - -  (#PCDATA)              >
<!ELEMENT s     - O  (#PCDATA)              >
```

5. Implementation

An SGML implementation typically consists of three sorts of tools:

- an editor;
- a parser;
- a processor.

5.1. Editor

The first tool in the chain to produce SGML documents is an editor.

One can use a simple non SGML aware editor and inserts tags manually. However it is a better idea to use an SGML aware editor which provides the user with a better interface.

In particular, such an editor protects the user from SGML markup much the way a word processor hides its own markup. A specialised SGML editor can replace the abbreviations used for the tag names with a better description and hide the tags from the text stream while using some sort of highlighting to inform the user of the tag inserted. All very similar to good word processors.

5.2. Parser

An SGML parser is a piece of software that:

- checks that a document conforms to the structure defined in the DTD;
- resolves references and minimizations.

5.2.1. Structure Conformance

A parser checks if a document respects its structure as defined in the DTD. For example, given the previous DTD, had we forgotten the f element:

```
<!DOCTYPE Memo SYSTEM "c:\sgml\dtd\memo.dtd">
<Memo>
<t>Club Secretary
<b>
<p>P. G. Wodehouse dedicated The Heart of a Goof <q>To my daughter
Leonora without whose never-failing sympathy and encouragement this
book would have been finished in half the time.</> Do you think SGML
would have done some good?
<s>Oldest Member
</Memo>
```

the parser would have complained of some document structure violation. Sgmls (a domain public parser) would produce the following error message:

```
sgmls: SGML error at memo.sgm, line 7 at ">":
       MEMO element ended prematurely; required F omitted
```

This is a highly interesting feature if SGML is used to enforce a corporate style. One can be sure that every document produced in the corporation will follow a common structure. If a standard output procedure set is associated with the DTD, even the look and feel of corporate documents can be standardised.

This also makes SGML particularly suitable for team-written documents, long documents, documents which must be maintained over a long period of time or any combinations of the above. In particular, SGML:

- offers a mechanism, the DTD, to communicate the desired structure of the document amongst the authors and enforce its respect;
- can be used for system independent document exchange or storage which is especially valuable for team documents or documents which require a long period maintenance.

This is the reason why SGML was selected as a major CALS standard. Weapon system documentation is usually very large (23.5 tons of technical manuals for the US Navy Aegis class cruiser Vincennes) and has to be maintained over the whole operational life of the weapons (40 years or so).

5.2.2. Canonical Form of Documents

The parser can also 'complete' the markup by adding omitted tags, resolving references (references are introduced later), etc. to give an SGML document suitable as input for the processor. The output document is similar to the one we presented in section 3 before minimization is applied. This is sometimes called the canonical or standard form of SGML documents.

A document in canonical form can be more easily transformed by the processor in an output program for the output device.

5.3. Processor

The processor takes the text in canonical or standard form and replaces the tags with formatting instructions to be executed by the output device. At this level only, the mapping is done between tags and formatting.

This can be as easy as replacing tags with formatting instructions specific to the output device. For example, to format our memo, it suffices to apply the transformations defined in the following table:

Tag	Action
<Memo>	change typeface to Palatino + '***Memorandum***'
</Memo>	reset typeface to Helvetica
<t>	new line + skip half line + '*To*' + tab stop
</t>	no action
<f>	new line + skip half line + '*From*' + tab stop
</f>	no action
	skip half line
	no action
	new line
	no action
<q>	'"
</q>	"'
<s>	skip half line + go to middle of the line
</s>	no action

How these actions will be implemented is device dependent. This can result in a PostScript® file if the memo is to be printed on a laser printer.

Advanced SGML

6. Syntax

6.1. Abstract & Concrete Syntax

The syntax of a language consists of the rules defining reserved words, their meanings and the meaning of character codes used by the language. SGML does not impose a particular syntax, rather it provides two levels of syntax, which are:

- the abstract syntax;
- the concrete syntax.

The abstract syntax is used to specify how SGML declarations and document type declaration should be constructed using symbolic names instead of symbols and keywords. The concrete syntax defines the actual set of reserved words and symbols used by a particular application of SGML.

So, when we said that an opening tag consists of an element name enclosed by angle brackets ('<' '>'), it was not exactly true. In fact, the element name is enclosed in markup delimiters which are Start tag open (STAGO) and Tag close (TAGC) in the abstract syntax. Mapping STAGO and TAGC to whatever string is the purpose of the concrete syntax.

The syntax defines:

- the markup delimiters;
- special characters with a system defined meaning, like the Record End (RE);
- the minimization optional features available;
- the character set to be used in the document;
- the maximum length of names;
- the maximum memory requirements of the document;
- some other optional features which are out of the scope of this introduction.

6.2. Reference Concrete Syntax

Does it mean that every SGML application has to define its own concrete syntax? Luckily not, the standard defines what is known as a *reference concrete syntax* i.e. a default concrete syntax. When an application has no special requirement on the concrete syntax used, it is best to use the reference concrete syntax. If an application has special needs, it is recommended to modify the reference concrete syntax following rules defined in ISO 8879 rather than building a new one from scratch, giving what is named a variant concrete syntax.

As one might have guessed, '<' is STAGO in the reference concrete syntax and '>' is TAGC. See appendix B for the definition of most elements using the abstract syntax.

Why would an application need to modify the reference concrete syntax? Suppose we are writing a mathematical book, a lot of expressions may use the symbol '<'. To avoid parsing errors, we must replace '<' by '`<`'† every time '<' creates ambiguities. This might be perceived as an unacceptable burden.

---

† '`<`' is an entity (to be introduced shortly) which is parsed onto the symbol '<'. Since '<' has a special meaning for the SGML parser, it cannot be inserted *as is* in the text without causing parsing errors. The entity must be insterted where a '<' will appear in the final text.

Another reason might be to extend the character set used in the document, for example to include accentuated letters.

In our examples, we will continue to use the reference concrete syntax.

6.3. SGML declaration

Potentially, every application can use a particular syntax tailored for its particular needs. Therefore when a document is to be exchanged its concrete syntax definition must be exchanged too. That is the role of the document type declaration, also known as SGML declaration, which is added at the beginning of a document to be exchanged.

The document type declaration should not be confused with the Document Type Definition (DTD). The declaration specifies the syntax and the character set used while the definition states the permissible tags and the document structure.

The declaration always starts with the reference concrete syntax Markup Declaration Open ('<!') and ends with the Markup Declaration Close ('>'). The declaration simply lists all of the elements of the concrete syntax.

Here is the declaration for the reference concrete syntax:

```
<!SGML "ISO 8879:1986"
       CHARSET
       BASESET "ISO 646-1983//CHARSET International Reference
         Version (IRV)//ESC 2/5 4/0"
       DESCSET    0  9 UNUSED
                  9  2  9
                 11  2 UNUSED
                 13  1 13
                 14 18 UNUSED
                 32 95 32
                127  1 UNUSED
       CAPACITY PUBLIC  "ISO 8879:1986//CAPACITY Reference//EN"
       SCOPE    DOCUMENT
       SYNTAX   PUBLIC  "ISO 8879:1986//SYNTAX Reference//EN"
       FEATURES
       MINIMIZE DATATAG NO OMITTAG YES RANK     NO  SHORTTAG YES
       LINK     SIMPLE  NO IMPLICIT NO EXPLICIT NO
       OTHER    CONCUR  NO SUBDOC  YES 99999999     FORMAL   YES
       APPINFO NONE>
```

The various elements of the SGML declaration are:

- CHARSET: starts the declaration of the document character set;
- BASESET: a standard character set on which the document character set is based. Here it is possible to choose a character set with accentuated letters for example;
- DECSET: variation to the character set for this document;
- CAPACITY: maximal memory storage requirement for the document;
- SCOPE: whether the declared concrete syntax must be used throughout the whole document (DOCUMENT) or for the document instance (INSTANCE) only, in which case the reference concrete syntax is used in the prologue;
- SYNTAX: syntax used in the document;
- FEATURES: describes which optional features will be used in the document;
- APPINFO: application specific information.

7. Entities

ISO 8879 defines an entity as:

A collection of characters that can be referenced as a unit. [16]

Since there is no other limit to the size of an entity than the available computer memory, a complete SGML document or any subsection of such a document can be treated as an entity. When an entity contains a complete SGML document, it is known as an SGML document entity. SGML subdocument entities are used to incorporate SGML coded text stored as a separate document. A subdocument might have been prepared with a different DTD.

Entities are important, among other things, for they provide a mechanism to include non SGML elements (like graphics, sound, etc.) in a document.

There are two sorts of entities:

- general entities;
- parameter entities.

These can be further subdivided into:

- entities which are declared and referred inside the document;
- external entities.

7.1. General Entities

A general entity can be used wherever parsable text can be entered. It is used to output previously defined text and/or markup instructions.

A reference to a general entity consists of an entity name preceded by an Entity reference opener ('&' in the reference concrete syntax) and followed by any of these three possibilities:

- a reference close (';' in the reference concrete syntax);
- a Record End code (RE) (system specific and often a carriage return);
- a character, such as a space, which is not part of a valid entity name.

An entity is declared using the keyword ENTITY (or its previously defined replacement), followed by the entity name and the entity text. In its simplest form, the entity text consists of a string of characters delimited by a pair of quotation marks or apostrophes.

If we add the following declaration in our DTD memo:

```
<!ENTITY Sage "Oldest Member">
```

we can save some typing by keying &Sage; where we want the text 'Oldest Member' to appear.

We might also declare an entity which contains markup like:

```
<!ENTITY Quotation "<q>To my daughter Leonora without whose
never-failing sympathy and encouragement this book would have been
finished in half the time.</>">
```

7.2. Parameter Entities

Unlike general entities, parameters cannot be used wherever parsable text can be entered but only in SGML declarations. Their declaration is similar to general entities except that the name starts with the '%' character. Similarly, a reference to a parameter entity starts with the character '%' and not '&'.

A typical declaration might be:

```
<!ENTITY % text "#PCDATA|q">
```

which will be later used as:

```
<!ELEMENT p  - O  (%text;)*>
```

this is in every respect similar to the declaration we used before:

```
<!ELEMENT p  - O  (#PCDATA|q)*>
```

7.3. External Entities

They are two sorts of external entities:

- publicly defined ones which, like public DTD, are supposed to be maintained by a standardisation body;
- system specific ones.

To use a public external entity it suffices to follow its name by the keyword PUBLIC and a public identifier:

```
<!ENTITY % ISOgrk1 PUBLIC "ISO 8879-1986//ENTITIES Greek Letters//EN">
```

Similarly, system specific external entities are defined by following the name with the SYSTEM keyword (or its previously defined replacement) and, potentially, a system identifier:

```
<!ENTITY part2 SYSTEM "c:\sgml\doc\part2.sgm">
```

This declaration assumes that part2 consists of text markuped in accordance with the present document DTD. An external entity can also contain a subdocument i.e. a document markup with its own DTD or even non SGML data. The valid types for an entity are:

- text markup in accordance with the current DTD;
- subdocument i.e. text coded in SGML but using an alternate DTD;
- character data (CDATA, or its previously defined replacement) that contains valid SGML characters but coded using a special notation. This is data with valid SGML characters but intended for another processor;
- non SGML data (NDATA, or its previously defined replacement) that contains codes outside the set declared to be valid SGML characters for the document;
- specific character data (SDATA, or its previously defined replacement) that contains characters whose role is specific to the local system.

To allow the SGML program to process the entity, the notation used must be declared using the keyword NOTATION (or its previously defined replacement) followed with the notation name and an identifier. The notation name is the one to be used after CDATA, NDATA or SDATA. A notation can be declared as system specific:

```
<!NOTATION TIFF SYSTEM "TIFF reader">
```

When it encounters an entity defined in another notation, the SGML parser will use the notation to request the system (the computer on which the SGML parser executes) to process the entity data. The system will invoke the appropriate formatter. When the data have been processed, the system will pass the result to the parser for blind incorporation in the parsed document.

```
<!ENTITY graphic1 SYSTEM "c:\graphics\foo.tif" NDATA TIFF>
```

Therefore an SGML document can contain data in any format, even those which did not exist when the SGML standard was designed. SGML is not concerned at all by the data notation of those external entities, it only knows about their role in the document structure.

Finally, a locally stored subdocument is declared with the SUBDOC keyword (or its previously defined replacement):

```
<!ENTITY appendix SYSTEM "appendix.doc" SUBDOC>
```

7.4. Exotic Characters

The possibility to insert other characters than those commonly found on a QWERTY keyboard is a very important feature of an international standard, especially in publishing. It is often necessary to insert special characters in a document. By special characters we mean those reserved by SGML for a special usage ('<', '&') and which cannot therefore be inserted anywhere in the document, any exotic character which are not part of a normal keyboard (ℵ, ©, ←, ∇, etc.) and, of course, the accented letters (é, î, ë, etc.).

We saw it is possible to redefine the character set used in the document when we studied the concrete syntax. But creating a variant concrete syntax is not always desirable for it might create problems when documents are to be exchanged. This is a bigger problem when relatively few special characters are to be inserted. Luckily, in that case, it is also possible to use entities which in most cases implies less cumulative effects.

ISO 8879 declares nineteen entity sets to provide for such symbols. These are:

- sets of alphabetic characters: Latin accentuated characters used in Western European languages, Greek , Cyrillic alphabets;
- a set of numeric and special characters;
- a set of diacritical marks that can be used for building other accented letters;
- a set of publishing characters;
- a set of box and line drawing characters;
- sets of technical characters;
- sets of mathematical symbols.

8. Storage Model

SGML does not impose any storage model.

For an SGML system, a document is an entity and may include other entities. Some entities are declared as part of the document while the others are external entities. The entities constitute the virtual storage model of SGML. The mapping between the virtual storage and the actual storage is done by an *entity manager* and is not standardised.

When external entities are declared, an external identifier is associated with them. As we know, the identifier can be either public or specific to the local system. The identifier is the link between the virtual storage model and the actual storage on the computer. This consists of whatever information is required by the entity manager to do the mapping.

Note that nothing in the model imposes that entities be stored in a different file than the SGML document; rather entities are a logical division. It is up to the entity manager to provide the parser with the correct entity.

For example, the whole document can be stored in a database, each entity of the document being a different record.

The entity manager is also responsible to invoke the appropriate processor when an entity is in a non SGML format and to pass the result of the processing to the SGML system.

9. Marked Sections

Marked sections are those sections of a document that require special handling to determine whether they must be output or not, if they must be parsed, etc.

A marked section consists of a markup declaration open ('<!' in the reference concrete syntax) immediately followed by a single declaration subset open ('[' in the reference concrete syntax). Then comes a status keyword which indicates what special handling the section requires. The marked section ends with a marked section close (']]' in the reference concrete syntax) and a markup declaration close ('>' in the reference concrete syntax).

```
<![status-keyword [Text with or without markup]]>
```

Valid status keywords are:

- IGNORE: section omitted from processing;
- INCLUDE: section processed;
- TEMP: section temporary part of the document but might be removed later and treated as an INCLUDE section;
- CDATA: character data. This is a valid SGML character but intended for another processor;
- RCDATA: replaceable character data, similar to CDATA except that entity reference are replaced.

Marked sections are particularly useful when used in conjunction with parameter entities. The status keyword is defined as an entity. If one needs to change the status, it suffices to change a line at the beginning of the document. Suppose that the memo exists in two languages (English and French), we want to store them both in a single document but in such way that only one is used at a time. The solution is to include the parts that may vary from one language to the other in marked section:

```
<!DOCTYPE Memo SYSTEM "c:\sgml\dtd\memo.dtd"[
<!ENTITY % ENGLISH "INCLUDE">
<!ENTITY % FRENCH "IGNORE">]>
<Memo>
<t>Club Secretary
<f>Oldest Member
<b>
<![%ENGLISH [<p>P. G. Wodehouse dedicated The Heart of a Goof <q>To my
daughter Leonora without whose never-failing sympathy and
encouragement this book would have been finished in half the time</>.
Do you think SGML would have done some good?]]>
<![%FRENCH [<p>P. G. Wodehouse a dédié The Heart of a Goof <q>A ma
fille Leonora, sans la sympathie et les encouragements de laquelle ce
livre aurait été écrit en la moitié du temps.</>Croyez-vous que SGML
l'aurait aidé?]]>
<s>Oldest Member
</Memo>
```

10. Parameters

We said that the ability to carry more information in the markup is an essential feature that distinguishes generalized markup from generic coding. In SGML this is accomplished through parameters.

Extra arguments can be added like an identifier, useful when concerned with a database or multimedia. A parameter declaration starts with the ATTLIST keyword (or its previously defined replacement) followed by the element name we want to define parameters for, the possible value for the parameter and a default value.

We will enhance our DTD with two parameters:

```
<!-- DTD for simple office memorandum                    -->
<!--      ELEMENTS  MIN  CONTENT (EXCEPTIONS)             -->
<!ELEMENT Memo      - -  ((t & f),b,s?)                    >
<!ELEMENT t         - O  (#PCDATA)                         >
<!ELEMENT f         - O  (#PCDATA)                         >
<!ELEMENT b         - O  (p)*                              >
<!ELEMENT p         - O  (#PCDATA|q|r)*                    >
<!ELEMENT q         - -  (#PCDATA)                         >
<!ELEMENT s         - O  (#PCDATA)                         >
<!ELEMENT r         - O  EMPTY                             >
<!--      ELEMENTS  NAME    VALUE              DEFAULT-->
<!ATTLIST Memo      status  (confiden|public)  public   >
<!ATTLIST p         id      ID                 #IMPLIED >
<!ATTLIST r         id      REFID              #REQUIRED>
```

A status can be attached to a Memo, giving some information on its confidentiality. A Memo can be confidential ('confiden', don't laugh they only give you 8 characters) or public. By default, it is public. This information can be used by the software which stores the SGML document to restrict the access to authorised persons only. An SGML document can contain its security information.

A paragraph can have an identifier associated with it. The ID keyword (or its previously defined replacement) means that the value must be a document unique identifier. The #IMPLIED keyword (or its previously defined replacement) means that the SGML parser will generate a default value when none is supplied.

To use the reference, we also defined a new element (r) whose content is EMPTY for it will be automatically generated by the program. The #REQUIRED keyword (or its previously defined replacement) used for the default value means that the parameter must be supplied in the document.

Now if we want to refer to a paragraph in a document we will:

- add an identifier to the paragraph:

  ```
  <p id="unique id">Some text to be referred later on.
  ```

- use an r element where we want the reference to appear. This way the SGML parser will ensure that the reference is up-to-date.

  ```
  See <r id="unique id"> for more on this.
  ```

11. System Dependent Markup

It was impossible, during the design of SGML, to foresee every usage that might occur in the future. Therefore, the designers provided some means to escape the SGML markup. It is possible to insert processing instructions i.e. instructions specific to the processing system.

> *... processing instructions serve as a useful escape valve for failures of rule-based application design or implementation. In a perfect world, they would not be needed, but, as you may have noticed, the world is not perfect. [16]*

Basically, there are two ways to insert processing instructions in a document:

- processing instructions can be added as markup enclosed between '<?' and '>';

  ```
  <?49 694 moveto 285 0 rlineto>
  ```

- entities can be defined as containing processing instruction with the keyword PI. Those entities are similar to a data entity in that they are not parsed by the SGML parser but they differ in that they are not considered to contain data.

  ```
  <!ENTITY PSLine PI "49 694 moveto 285 0 rlineto">
  ```

Other Aspects

12. Non Publishing

SGML started its life in the publishing area as a better markup mechanism. Until now, we particularly insisted on the publishing aspect of SGML but it would be a mistake to think that SGML is for publishing business only. SGML is concerned about document structure, whatever the document, whatever the media.

Notice that nowhere in SGML documents did we talked of pages because pages are physical matters. The formatting procedures associated to a DTD will take care of the mapping to pages, if the media turns out to be paper print.

Documents in SGML format can be printed, stored in a database for on-line consultation, sent by E-mail, etc., with SGML serving as a common storage format. Likewise, SGML can be used to express the structure of a multimedia database (Cf. HyTime in the next section).

In the first chapter, we saw that 90% of the total amount of information a company processes is unstructured information. It is illustrated in figure 5.

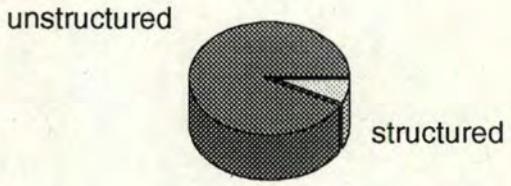


Figure 5: Structured Vs unstructured documents

In the first section of this chapter, we saw two characteristics generalized markup must fulfil, the second one was:

> *markup should be rigorous, so the techniques available for processing rigorously defined objects like programs and databases can be used for processing document as well[15]*

It seems they were born to meet each other! SGML makes it possible to use techniques developed to process structured information with unstructured information.

> *... The concept of corporate database was born. [23]*

Therefore SGML usage is much wider than publishing. SGML can be, and has been, successfully used in other areas. SGML has been selected as a CALS standard for the exchange of technical document, the US Patent and Trademark Office (PTO) considers the use of SGML as a storage format for exchange of patents [14], the automotive industry is adopting SGML for the exchange of service information [18]. And SGML is the basis of HyTime, the new hyperdocument interchange standard (to be introduced shortly).

Lest the importance of this remark be missed, let us repeat it once again: there is more in SGML than publishing. SGML is about document structure.

13. Related Standards

13.1. SGML Document Interchange Format

SGML is independent of storage format. Nevertheless, recognising the need for a standard interchange format, the ISO designed a standard interchange format named SGML Document Interchange Format (SDIF) or ISO 9069.

Basically SDIF defines a simple rule to transform an SGML document and all its associated files into a binary stream. The encoding rules are written in the Abstract Syntax Notation One (ASN.1 - ISO 8824) and the actual binary encoding is supposed to follow the Basic Encoding Rules (ISO 8825). This is the standard for ISO interchange of information.

SDIF transforms an SGML document, potentially composed of various entities, and assembles it in a single data stream. Those entities can be any valid SGML entities i.e. whatever the computer can handle (as well as document text, it can be vector or raster graphics, sound, etc.). Associated documents (accompanying letters, memos, etc.) can also be incorporated in the SDIF data stream.

The data stream assigns SDIF names to the data stream self, the SGML document and its associated entities to provide a unique identifier mechanism throughout the data stream.

The exchange follows a three steps process:

- the SGML document and associated documents are *packed* in SDIF format;
- the SDIF stream is exchanged;
- the receiver *unpacks* the SDIF stream and organises the entities according to its own local system requirements.

We can visualise SGML documents, with associated entities, as being enveloped at the time of transmission. This is synonymous to the enveloping of data in a data interchange syntax, such as EDIFACT.

Everything is transmitted as if the SGML document and all its associated entities went in an envelope for the time of the transmission.

13.2. Open Document Architecture

SGML and ODA do not emphasise on the same points. SGML emphasis is on the transmission of the structure of a document while ODA encodes both the structure and the layout of the document.

> *Keep anything concerned with presentation distinct from the information content of an SGML document. [22]*

This reflects the different backgrounds from which the two standards originated. ODA originates in an office culture, where the word processor is the dominant publishing tool. Therefore the emphasis is on presentation. ODA tries to achieve identical look amongst various platforms.

On the contrary SGML, coming from the publishing industry, leaves presentation matter to the formatter and is designed specifically to enable exchange of documents amongst systems with different and potentially incompatible output devices. It permits the output of a draft on a low quality printer and the output of the final document on very high quality printer.

> *SGML is concerned only with structure, so it is up to each user community to determine document layout. The* ***Document Style Semantics and Specification Language (DSSSL†)*** *is being developed to provide a means of defining further processing of documents. It will also enable the*

† This acronyms looks like some sort of joke.

sender of a document to specify how it should be formatted for presentation and printing. [23]

Within CALS, the need to exchange formatting instructions has outpaced the standardisation of DSSSL. Therefore a special specification with limited scope, the Formatting Output Specification Instance (FOSI), was incorporated into MIL-M-28001. It addresses simpler needs than DSSSL and is due to be replaced when the development of DSSSL will be over.

13.3. HyTime

Hypermedia/Time-Based Structuring Language (HyTime) or, more formally, ISO 10744 is a standard for the representation and the exchange of hypermedia documents also known as hyperdocuments. HyTime is an application of SGML.

> ***What is hypermedia?*** *Hypermedia is the union of two information processing technologies: hypertext and multimedia. Hypertext information is accessed in more than one order. Multimedia information is communicated by more than one means. [17]*

There are numerous hypermedia applications running on a wide variety of platforms, each using its own representation. HyTime provides a universal format for hyperdocument exchanges amongst those applications.

The pace of evolution in these technologies is a problem for any standardisation effort. It would have been unrealistic to design a notation that encompasses the functionalities of all hypermedia applications (like it is done for graphic interchange standards). Such a standard would have been obsolete on the day of its adoption!

HyTime resolves this problem by standardising only some facilities of hypermedia applications, those functionalities that are the core of hyperdocuments, in particular those dealing with the addressing of portions of documents. This includes linking, alignment and synchronisation, but not data content notation.

As would be expected for an application of SGML, only the central hyperdocument must conform to HyTime while the associated documents can be expressed in whatever notation is best appropriated. Like SGML, HyTime is an enabling standard, it provides a neutral base for the interchange of application-specific hypermedia information.

There are two parts in HyTime:

- an hypertext part which provides linking;
- a multimedia part which provides time synchronisation.

Like SGML, HyTime is independent of the actual storage but Hypermedia applications are limited by today's technology. Specifically, they often require the hyperdocument to be self-contained and coded in a format specific to the platform on which they run, for maximum efficiency. Of course, an ideal interchange format would be platform independent. Therefore it has been suggested to use two sorts of interchange standards for HyTime documents:

- archival: a system-independent format used to store hyperdocuments for interchange or archive;
- delivery: a system-dependent format, optimised for speed, which is used by the hypermedia application.

Therefore before a hyperdocument can be read by an application it must have been translated from the archival format to the delivery one.

This is probably a transitory situation because of:

- hardware improvements;
- SGML-B: a standardised binary encoding of SGML parsed document optimised for hypertext access.

Hypermedia technology will certainly evolve too; new functionalities will probably be added over time. All this raises the question: «Why use HyTime today?», the answer is simple: because HyTime documents are ready for evolution. HyTime is flexible, modular and extensible enough to cope with future innovations. It provides a path to bring today's hyperdocuments into tomorrow's world.

This makes HyTime an ideal standard for documents with an expected long lifetime, like SGML. This explains why the DoD is so interested by HyTime.

14. Further readings

Despite its apparent simplicity† SGML is a very powerful tool. It is not possible to introduce every aspect of its use in such a short chapter.

For further study of SGML, the following books are recommended:

- SGML an Author's Guide [20]: a good introductory book. Particularly concerned by an author's point of view;
- Practical SGML [19]: study of some practical issues raised by SGML. Notice that chapter 13 is concerned with the relation between SGML and EDI;
- SGML Handbook [16]: the annotated standard by the editor standard. This book will be of interest to anyone implementing SGML;
- SGML and Related Standards [22]requires knowledge of SGML, this book provides a good study of various SGML related issues;
- SGML: The User's Guide to ISO 8879 [21]: a reference book to ease the reading and use of the standard, of little use without a copy of the standard. ■

† After all the SGML standard is not very thick, especially when compared to other standards concerned with document interchange like ODA, for example.

15. References

[14] Dr. Donald P. D'Amato and Rex C. Klopfenstein
A Study of the CALS Standards for the Interchange of Patent Documents
CALS Journal, Summer 1993, p. 28-35

[15] Charles F. Goldfarb
A Generalized Approach to Document Markup
ACM Sigplan Notices, Volume 16, Number 6, June 1981, p. 68-73

[16] Charles F. Goldfarb
The SGML Handbook
Clarendon Press, UK, 1990

[17] Charles F. Goldfarb
HyTime: A Standard for Structured Hypermedia Interchange
IEEE Computer, August 1991, p. 81-84

[18] James H. Harvey
SGML Applied to Automotive Service Information
CALS Journal, Fall 1993, p. 27-31

[19] Eric van Herwijnen
Practical SGML
Kluwer Academic Publishers, The Netherlands, 1990

[20] Bryan Martin
SGML an Author's Guide
Addison-Wesley, UK, 1988

[21] Joan M. Smith and Robert Stutely
SGML: The User's Guide to ISO 8879
Ellis Horwood, UK, 1988

[22] Joan M. Smith
SGML and Related Standards
Ellis Horwood, UK, 1992

[23] Technology Appraisals
Open Information Interchange
Technology Appraisals Ltd, UK, 1993

Appendices

Appendix A: A Brief History of the Development of SGML†

SGML, in its present form, is the result of the efforts of many people, channelled into four major activities that occurred over the past twenty years: generic coding, the GML and SGML languages, the SGML standard, and major SGML applications.

The Generic Coding Concept

Historically, electronic manuscripts contained control codes or macros that caused the document to be formatted in a particular way ('specific coding'). In contrast, generic coding, which began in the late 1960s, uses descriptive tags (for example, 'heading', rather than 'format-17'). Many credit the start of the generic coding movement to a presentation made by William Tunnicliffe, chairman of the Graphic Communications Association (GCA) Composition Committee, during a meeting at the Canadian Government Printing Office in September 1967: his topic — the separation of the information content of documents from their format.

Also in the late 1960s, a New York book designer named Stanley Rice proposed the idea of a universal catalogue of parameterized 'editorial structure' tags. Norman Scharpf, director of the GCA, recognized the significance of these trends, and established a generic coding project in the Composition Committee.

The committee developed the 'GenCode® concept', recognizing that different generic codes were needed for different kinds of documents, and that smaller documents could be incorporated as element of larger ones. The project evolved into the GenCode Committee, which later played an instrumental role in the development of the SGML standard.

GML and SGML: Languages for Generic Coding

In 1969, Charles Goldfarb was leading an IBM research project on integrated law office information systems. Together with Edward Mosher and Raymond Lorie he invented the Generalized Markup Language (GML) as means of allowing the text editing, formatting, and information retrieval subsystems to share documents.

GML (which, not coincidentally, comprises the initials of its three inventors) was based on the generic coding ideas of Rice and Tunnicliffe. Instead of a simple tagging scheme, however, GML introduced the concept of a formally-defined document type with an explicit nested element structure.

Major portions of GML were implemented in mainframe 'industrial strength' publishing systems, by IBM and others and achieved substantial industry acceptance. IBM itself, reckoned to be world's second largest publisher, adopted GML and now produces over 90% of its documents with it.

After the completion of GML, Goldfarb continued his research on document structures, creating additional concepts, such as short references, link processes, and concurrent document types, that were not part of GML but were later to be developed as part of SGML.

---

† The following copyright notice and permission applies only to this appendix (Appendix A) of this chapter.

Development of SGML as an International Standard

In 1978, the American National Standards Institute (ANSI) committee on Information Processing established the Computer Languages for the Processing of Text committee, chaired by Charles Card, then of Univac, with Norman Scharpf as a member. Goldfarb was asked to join the committee and eventually to lead a project for a text description language standard based on GML. The GCA GenCode committee supported the effort and provided a nucleus of dedicated people for the task of developing Goldfarb's basic language design for SGML into a standard.

The first working draft of the SGML standard was published in 1980. By 1983, the GCA was able to recommend the sixth working draft as an industry standard (GCA 101-1983). Major adopters included the US Internal Revenue Service (IRS) and the US Department of Defense.

In 1984, with feedback from the GCA standard in hand, three more working drafts were produced. The project, which had been authorized by the International Organization for Standardization (ISO) as well as ANSI, re-organized. It began regular international meetings as what is now called ISO/IEC JTC1/SC18/WG8, chaired by James Mason of the US Oak Ridge National Laboratory. Work also continued in the ANSI committee, now called X3V1.8, chaired by William Davis of SGML Associates, and supported by the GCA GenCode committee, chaired by Sharon Alder of IBM. Alignment between ISO and ANSI was maintained by Goldfarb continuing as technical leader, serving as project editor for both groups.

In 1985, a draft proposal for an international standard was published and the international SGML Users' Group was founded in the UK by Joan Smith, who became its first president. Together with the GCA in North America, it played a vital role in educating the public about SGML and communicating user reactions and comments back to the development project.

A draft international standard was published in October 1985, and was adopted by the Office of Official Publications of the European Community. Another year of review and comment resulted in the final text, which — using an SGML system developed by Anders Berglund, then of the European Particle Physics Laboratory (CERN) — was published in record time after approval (ISO 8879:1986).

Important Early Applications of SGML

SGML applications are frequently developed for use by a single organization or a small community of users. Two early applications were developed with much broader participation: the Electronic Manuscript Project of the Association of American Publishers (AAP), and the documentation component of the Computer-aided Acquisition and Logistic Support (CALS) initiative of the US Department of Defense.

Electronic Manuscript Project

From 1983 to 1987, an AAP committee, chaired by Nicholas Alter of University Microfilms, developed an initial SGML application for book, journal, and article creation. The application is intended for manuscript interchange between authors and their publishers, among other uses, and includes optional element definitions for complex tables and scientific formulas.

The technical work was led by Joan Knoerdel of Aspen Systems, with participation by over thirty information processing organizations, including the IEEE, Council on Library Resources, American Society of Indexers, US Library of Congress, American Chemical Society, American Institute of Physics, Council of Biology Editors, and American Mathematical Society.

The AAP industry application standard has achieved significant acceptance, and has particularly been embraced by the emerging CD-ROM publishing industry. It has been adopted as a formal ANSI application standard (Z39.59) and a corresponding ISO standard is under development.

Computer-aided Acquisition and Logistic Support (CALS)

The SGML portion of CALS was initiated in February 1987 when Bruce Lepisto of the Department of Defense organized a committee to address the subject. The committee consisted of John Bean of Northrop, Pam Gennusa of Datalogics, Ed Herl of the US Army, and Mary McCarthy and Dave Plimier of the US Navy. They were subsequently joined by hundreds of representatives of military standard (MIL-M-28001) in February 1988.

Similar SGML projects are under way in the defence departments of Canada, Sweden, and Australia, and are under consideration by other countries. ■

Appendix B: Reserved Names and Keywords

This appendix specifies what can be redefined in the concrete syntax.

This table shows the keywords that can be changed by defining a variant concrete syntax.

ANY	ENDTAG	LINK	NUTOKEN	SGMLREF
APPINFO	ENTITIES	LINKTYPE	O	SHORTREF
ASN1	ENTITY	MD	OMITTAG	SHORTTAG
ATTLIST	EXCLUDE	MINIMIZE	OTHER	SHUNCHAR
BASESET	EXPLICIT	MODEL	PACK	SIMPLE
CAPACITY	FEATURES	MS	PCDATA	SPACE
CDATA	FIXED	MSICHAR	PI	SRCNT
CHANGES	FORMAL	MSOCHAR	PUBLIC	SRLEN
CHARSET	FUNCHAR	MSSCHAR	POSTLINK	STARTTAG
CONCUR	GENERAL	NAME	QUANTITY	SUBDOC
CONREF	ID	NAMECASE	RANK	SWITCHES
CONTROLS	IDREF	NAMES	RCDATA	SYNTAX
CURRENT	IDREFS	NAMING	RE	SYSTEM
DATATAG	IDLINK	NDATA	REQUIRED	TEMP
DEFAULT	IGNORE	NMTOKEN	RESTORE	UCNMCHAR
DELIM	IMPLICIT	NMTOKENS	RS	UNPACK
DELIMLEN	IMPLIED	NO	SCOPE	UNUSED
DECSET	INCLUDE	NONE	SDATA	USELINK
DOCTYPE	INITIAL	NONSGML	SDIF	USEMAP
DOCUMENT	INSTANCE	NOTATION	SEPCHAR	VALIDATE
ELEMENT	LCNMCHAR	NUMBER	SEQUENCE	YES
EMPTY	LCNSTRT	NUMBERS	SGML	

The delimiters, which can be changed in a variant concrete syntax, are presented in the following table with their abstract name, the default assigned by the reference concrete syntax and their role:

Name	Default	Role
AND	&	And connector
COM	--	Comment start or end
CRO	&#	Character Reference Open
DSC	]	Declaration Subset Close
DSO	[	Declaration Subset Open
DTGC	]	Data Tag Group Close
DTGO	[	Data Tag Group Open
ERO	&	Entity Reference Open
ETAGO	</	End-tag Open
GRPC	)	Group Close
GRPO	(	Group Open
LIT	"	Literal Start or End
LITA	'	Literal Start or End (Alternative)
MDC	>	Markup Declaration Close
MDO	<!	Markup Declaration Open
MINUS	-	Minus; Exclusion
MSC	]]	Marked Section Close
NET	/	Null End-tag
OPT	?	Optional Occurrence Indicator
OR	|	Or Connector
PERO	%	Parameter Entity Reference Open
PIC	>	Processing Instruction Close
PIO	<?	Processing Instruction Open
PLUS	+	Required and Repeatable; Inclusion
REFC	;	Reference Close
REP	*	Optional and Repeatable
RNI	#	Reserved Name Indicator
SEQ	.	Sequence Connector
STAGO	<	Start-tag Open
TAGC	>	Tag Close
VI	=	Value Indicator

Here is the set of limits defined by the reference concrete syntax that can be changed in a variant concrete syntax:

Name	Value	Description
ATTCNT	40	Number of attribute names and name tokens in an element's attribute definition list.
ATTSPLEN	960	Normalized length of a start-tag's attribute specifications
BSEQLEN	960	Length of a blank sequence in a short reference string
DTAGLEN	16	Length of a datatag
DTEMPLEN	16	Length of a data tag template or pattern template (undelimited)
ENTLVL	16	Nesting level of entities (other than primary)
GRPCNT	32	Number of tokens in a group
GRPGTCNT	96	Grand total of content tokens at all levels of a content model
GRPLVL	16	Nesting level of model groups (including first level)
LITLEN	240	Length of a parameter literal of attribute value literal (interpreted and undelimited)
NAMELEN	8	Length of a name, name token, number, etc
NORMSEP	2	Used instead of separators when calculating normalized lengths
PILEN	240	Length of a processing instruction (undelimited)
TAGLEN	960	Length of a start-tag (undelimited)
TAGLVL	24	Nesting level of open elements

5 Graphics Standards

You can dream, create, design and build the most wonderful place in the world, but it requires people to make the dream a reality.

Walt Disney

This chapter presents four graphics standards. Namely:

- the Computer Graphics Metafile (CGM) standard (ISO 8632), a general-purpose international standard for graphics;
- the CCITT Group 4 (CCITT Recommendation T.6) raster graphics compression scheme;
- the IGES file format for product data exchange (ANSI Y14.26M);
- the STEP draft international standard (ISO 10303) for product model exchange.

The first three standards form the basis of CALS standards; respectively MIL-D-28003A, MIL-R-28002B and MIL-D-28000A.

Because we found that few people clearly understand how computers handle graphics, this chapter also includes an introduction to computer graphics.

Although computer graphics is an exciting subject, it is also a difficult one partly because computers have at least three ways of dealing with graphics and partly because an in-depth study leads to mathematics which are beyond the scope of this document. As usual we have tried to give the background information for the reader to understand which problems are involved by the exchange of images.

Background on Computer Graphics Formats

1. Background on Computer Graphics Formats

Essentially there are two types of format to store an image on a computer:

- a colour sample of individual points in the image. This is the *raster* format;
- a description of the image in terms of its geometrical components (like circle, line, etc.). This is the *vector* format.

We will also consider a third way:

- a *product data model* of some artefact like models produced by CAD systems.

  This is not strictly a graphical format as a product model usually includes lots of non-graphical information, such as the material used, information for the machine tool, etc.

  If the model is 3D, it contains even more non graphical information (like metrics, etc.). But the model is often viewed and manipulated through a graphic. Consequently people tend to consider these models as graphical data. Furthermore, they can be considered as a sort of extension of vector formats.

Notice that each of these formats carries more information than the previous one. We go from low level formats (samples) to higher level ones (models).

1.1. Raster Image

Conceptually, the raster format is probably the easiest of the three. The colour of the image is sampled at individual points on a grid which covers the picture. See figure 11.

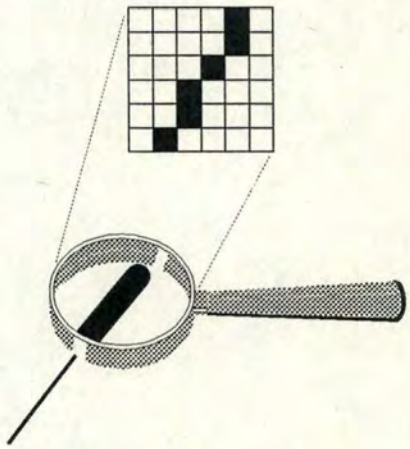


Figure 1: The image is sampled using a grid

The points are usually referred to as pixels (abbreviation for picture element). The set of colour of each pixel constitutes the image data; it is stored in a bitmap (sometimes spelled bit map).

The quality of the image is determined, of course, by the sampling frequency, that is known as the resolution of the image. The problem with raster images, is the problem of every sample: information is lost during sampling. The colour of every pixel on the grid is stored; however, everything between two pixels is lost. Given that, as mathematics teaches us, there is an infinity of dots between two points on a plane, the information on an infinity of points is lost.

The resolution is commonly expressed either by the size of the grid, for example 800 by 600 or, for printers, in dot per inches (dpi), for example 300 dpi.

The other parameter that influences the image quality is the number of bits used to store each pixel colour. Typically used values are:

- 1 bit for bitonal (often black and white) images;
- 4 bits which allows the use of 16 different colours;
- 8 bits for 256 colours. A current use is grey-scale images: human eyes cannot discriminate more than an average of 64 nuances of the same colour; therefore if the 256 colours are all levels of grey, we can store all the visible nuances of a so-called black and white photograph;
- 24 bits or True Colours store 16 millions colours; it is generally considered that this suffices to store all the colours the human eye can discriminate;
- 32 bits; these are 24 bits picture with an extra byte used to store transparency information which specifies how 2 images must be superposed.

The sampled nature of raster images raises problems when the image has to be scaled. It is not uncommon: printers and screens have different resolutions therefore if the resolution of a raster image is appropriate for the screen, it is not for the printer or vice versa. The problem is that the sample does not provide enough information to scale wisely. One must resort to complex transformations that either try to guess what the missing pixels were (scaling up) or to remove intelligently some pixels (scaling down).

To minimise the information lost one may increase the sampling frequency i.e. sample more points. But this raises the second problem of raster images: the size. A raster image is usually huge. For example, a photographic quality image (24 bits) of 640 by 480 pixels will amount to almost a million bytes ((640*480) pixel * 3 bytes/pixel) which is commonly considered as big.

The only realistic solution to store raster graphics is to compress the data by identifying and suppressing redundancies in the image. Many compression techniques have been devised which achieve various savings at various costs.

1.2. Vector Images

Vector formats are geometrical descriptions of images. A vector oriented format stores an image as a collection of geometric primitives like circles, rectangles, lines.

This often produces more compact data: a circle, for example, can be represented by three numbers (the coordinates of its centre and its radius) where it would have required the storage of a potentially huge bitmap in raster formats. This is shown in figure 2.

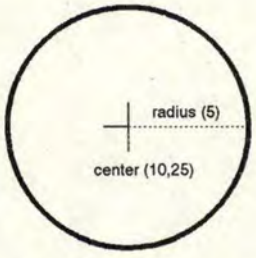


Figure 2: A circle is stored as its center and radius

There is no problem to scale the image: to draw the circle at a different size, it suffices to multiply its radius by a scaling factor. These formats store more information on the structure of images than the raster ones because they store the geometrical nature of image components, not samples.

This power has its drawback. Namely to produce a vector image one must be able to describe the image mathematically which is not always trivial. A photo for example is almost impossible to describe completely as a set of vectors.

Realistic use of vector images is limited to a subset of computer generated images, mainly technical documents and some art images. Raster images on the contrary can be used with any sort of images.

1.3. Model Description

The notion of model is central to the design activities. It is also what CAD tools manipulate. We will review more thoroughly the notion of model in design in section 11 of this chapter.

This format stores all the product data which includes non-graphical elements: the model carries information on which device is represented, eventually how it can be manufactured, etc.

> *The term* ***product data*** *denotes the totality of data elements that completely define the product for all applications over its expected life cycle. Product data includes the geometry, topology, relationships, tolerances, attributes, and features necessary to completely define a component part or an assembly of parts for the purposes of design, analysis, manufacture, test and inspection [1].*

It has many properties of the vector oriented description (2D models are in fact a superset of vector description) but is usually bigger because of the extra information in the model.

1.4. Need for Different Formats

Those different formats cohabit because they serve different purposes.

Raster formats are very good with so-called natural images, i.e. images of the real world like photographs, or would-be natural images like photorealistic computer generated images.

Vector formats are especially appropriate for computer generated images while model formats are particularly adapted for engineering design.

Let's take some examples which will illustrate the need for different formats.

First, it has already been said that each format carries more information than the previous one. But in some circumstances, we do not need all the information.

Let's take a building as an example. Assume the building was designed with a CAD system. It is not an unrealistic assumption if the building is recent.

Suppose we want to write a booklet to present the building to potential buyers. It is important for the buyer to see a drawing of a typical apartment. It would be inefficient to include the whole CAD model in the booklet file. The model contains architectural data that are not relevant to the buyer. Rather we will extract a graphic representation of an apartment from the model, probably in a vector format, and include only this smaller image in the booklet. Notice that the extraction can be done automatically.

Similarly, we may want to have a photo of the apartment on the cover. A photo is a natural image, it was not created by a computer and the best way to store it on a computer is in raster format.

Notice that even on films, photos are colour samples: tiny silver grains play a role similar to the computer pixels.

Figure 3

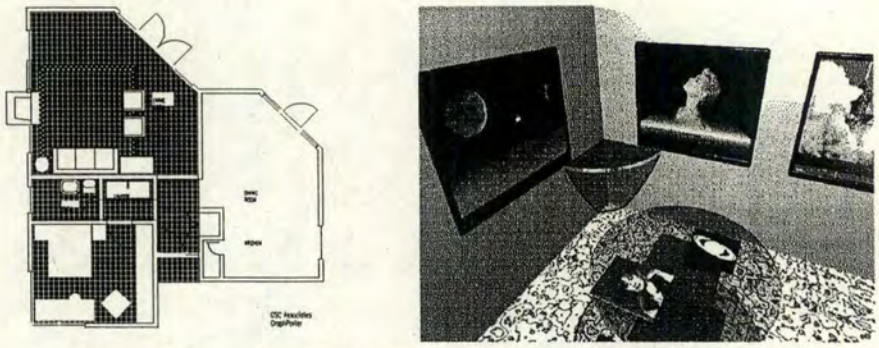


Figure 3: Apartment drawing (vector) and one room (raster)

Another reason to have various encoding techniques is that, for cost reasons, today most output devices are raster ones. This includes screens and printers. Plotters are the last vector oriented output devices. They draw an image by moving pens on a piece of paper. Since plotters avoid the sampling process, they draw better looking images, at least as good as what a designer can do. But plotters, being mainly mechanical, are slow and expensive. Moreover they cannot print raster images. Plotters are mainly used for engineering.

Anyway, it is not because a device is raster oriented that the most efficient way to control it is raster oriented. Most software tools generate vector drawings; this includes GUIs (a window is in fact a rectangle), presentation packages, spreadsheets, etc. Consequently it is not uncommon to control a raster device (like a screen) through software libraries which accept vector oriented instructions. It achieves a certain independence from the actual resolution of the device, may save on internal storage and means that the difficult routines that rasterised vector primitives can be reused and shared amongst software products.

So to draw a view of a 3D model, a CAD system will probably first produce a 2D vector graphic which will then be rasterised by the drawing library that drives the screen.

In general, it is easier to go from a high-level format to a lower one than to go the other way, especially if the process is to be automated. Truly some software tools exist which convert raster images to vector ones but their use is limited.

Again it is important to emphasise that three types of graphics formats exist because they cover different needs. In document exchange activities, we must provide for all those needs.

Computer Graphics Metafile

2. Usage

Originally the Computer Graphics Metafile (CGM) was a vector format. Since the 1992 revision, the standard incorporates full featured raster graphics support. This was felt an important requirement by CGM users. Notice that ISO has no dedicated raster format.

CGMs are used to store pictures:

- for further processing later in time;
- for further processing on another computer;
- for a combination of both previous points.

CGMs are for still pictures only. Though more than one picture can be stored in a single metafile, it is not appropriate for animated pictures or movies. Also as CGMs store the state of pictures frozen at a certain moment (like a snapshot), CGMs cannot be used to record the dynamic sequence of operations that leads to the creation of a picture.

Before going any further, we will highlight some cases where CGMs are used. This section does not intend to provide a complete survey of all potential uses of CGMs: what is so exciting with graphics is that there are always new, more interesting and funnier ways to use them.

A typical usage of CGM is the insertion of graphics in text documents. The two international standards for document exchange, the Standard Generalized Markup Language (SGML) and the Open Document Architecture (ODA), allow the inclusion of CGMs in a standard way.

Since CGM is a standard, many off-the-shelf applications are capable of interpreting and/or generating CGM files. Many graphical packages, word processors and desktop publishing systems are now able to process CGMs. This makes CGMs an excellent choice for the exchange of graphical data whatever they are.

Very often in technical documents, we are not interested by the complete model of an artefact as exported from a CAD system. The image suffices. It would be inefficient to store the complete model which includes data used by machine tools, analysers, etc. Rather an image of the artefact is extracted, potentially as a CGM, and will become part of the document.

Another potential application of CGMs is to drive output devices like plotters or printers. CGMs makes the application that requests the output independent from the output device. In this role, CGM is used instead of PDLs (Page Description Language), like PostScript®. This is sometimes necessary because PostScript®, although device independent, assumes a raster device i.e. not a plotter. To drive plotters there is only *de facto* standards like HP-GL™, which CGM can replace.

3. Organisation of the Standard

A file format can attempt to fulfil various requirements. For the applications targeted by the CGM standard, the potential requirements are:

- minimal file size;
- ease of transfer across networks;
- speed with which the data can be generated and interpreted;
- human-readability of the stored files (for debugging).

Alas, those requirements are conflicting. For example sophisticated compression techniques minimise file size at the cost of processing speed and human readability.

The standard elegantly handles these conflicts by dividing its specifications in two logical levels:

- a functional level;
- an encoding level.

3.1. Functional Level

The functional level defines the functionalities and general organisation of a CGM. At the functional level, a metafile is described in terms of elements. An element is defined by a name and a potentially empty list of parameters.

Examples of element are:

- 'Rectangle' which takes two corners as parameters (Cf. figure 4);
- 'Polygon' which takes a list of points as parameters;
- 'Begin Metafile' which takes an identifier as parameter;
- 'New Region' which takes no parameter.

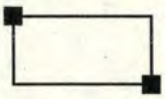


Figure 4: A rectangle has two corners as parameters

Note that the notion of instance of an element does not exist in the CGM literature. Therefore, the word *element* is used both to describe the abstract notion of element (the rectangle seen as a particular sort of polygon) and a particular occurrence of an element within a file (a rectangle in a given image i.e. where it has a size, a colour, etc.).

3.2. Encoding Level

What elements constitute a metafile and how these elements relate is defined at the functional level. How these elements will map into an actual file on a computer is described at the encoding level.

This approach eases the design of various encodings each of which satisfy a different set of requirements while all being functionally equivalent.

Let us, as an example, consider the case of the rectangle element. We saw that the standard defines a primitive for rectangles. At the functional level, a rectangle is defined by two points giving the position of two of its corners. What we consider, at this level, is the abstract idea of rectangles. We also consider the concept of points that we defined as two numbers†. We are not, at the functional level, interested in the actual pattern of bits that will represent a rectangle on a given computer.

Transforming this conceptual vision of a rectangle in something that can be written to and read from a file is the resort of the encoding level. For example, with the Clear Text Encoding (to be introduced shortly), we would write something like:

```
Rectangle (12,15) (25,60);
```

The structure of the metafile is completely defined at the functional level and is therefore independent from any encoding. This contrasts with older graphics format where the distinction between functional and encoding levels was not drawn: the functionalities were described with and by their encodings.

† By default, two integers whose actual precision (number of bits) is encoding dependant.

3.3. Organisation of the Standard

The ISO standard consists of 4 parts. The first one defines the functional level and the other three, encodings. Those three encodings are designed to be fully compatible i.e. it is possible to convert from one encoding to another without loss of information. This is logical since they are all based on the same functionalities.

The three encodings are:

- character encoding: optimised for electronic communication, it consists only of characters and is very compact;
- binary encoding: optimised for speed of decoding, it is close to most computer representation of elements;
- clear text encoding: optimised for human readability, it uses meaningful strings of character to code elements which gives it the look and feel of a computer language.

Since the functionalities are defined independently from any encoding, it is also possible to write a CGM in a private encoding.

4. Structure of a CGM

A CGM is organised as a series of layers of detail. These layers are:

- metafile;
- pictures;

  which consist of:
  - a picture descriptor;
  - a picture body;
- elements;

  made of:
  - a name;
  - parameters.

The elements are the building blocks of a metafile. They are defined by a name and a potentially empty list of parameters. A metafile is simply an ordered sequence of elements. There are elements to structure the metafile, others to express graphical primitives, etc. Elements are studied in the next section.

A picture starts with a 'Begin Picture' element and ends with an 'End Picture' element. A picture consists of two logically separated parts:

- the picture descriptor that gives the interpreter† some information to help it decode the picture;
- the picture body that contains the actual description, in terms of graphical elements, of the picture.

Zero, one or more pictures are grouped in a metafile. The metafile is the highest level of the hierarchy. It starts with a 'Begin Metafile' element and ends with an 'End Metafile' element. Apart from pictures, metafiles contain metafile descriptors that supply the interpreter with information which helps it interpret all the pictures of the metafile.

Each picture in a metafile is independent from the others. Although in practice a metafile is often processed sequentially, it need not be.

---

† In the CGM literature, the software that creates (writes) a CGM is a *generator* and the one that reads it is an *interpreter*.

The structure of CGM metafiles is illustrated in figure 5.

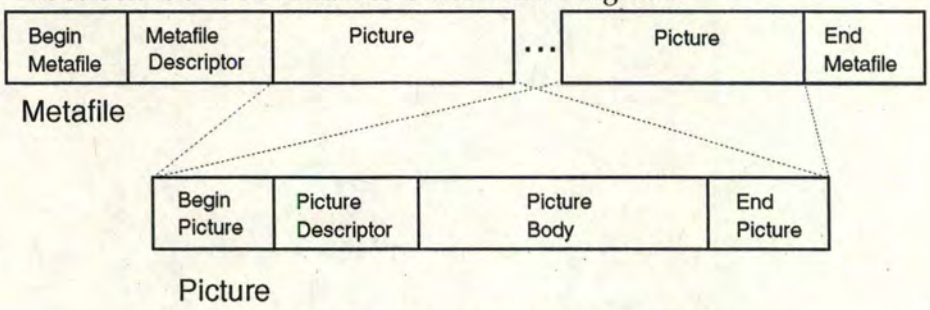


Figure 5: Format of a CGM metafile

5. Functional

An in-depth study of every element is out of the scope of this paper and of little interest to anyone who is not implementing the standard. Rather we will concentrate on a high level understanding of the various functionalities. We will adopt a twofold approach:

- first, we will overlook the classes of elements: the standard organises the elements in classes. We will briefly review those classes to acquire an high-level understanding of what functionalities the standard provides;
- next, we will have a deeper study of the graphical primitives for the two sorts of graphical data supported by the standard:
  - vector graphics primitives;
  - raster graphics primitives.

For historical reasons there are three versions of CGM in use which are upwardly compatible. Version 2 is a superset of version 1, and version 3 is a superset of version 1 and 2. A new version (version 4) is currently under development.

5.1. Element Classes

The standard organises the various CGM elements in a few classes. We will now review those classes with some explanation for each of them. The classes are:

- delimiter;
- metafile descriptor;
- picture descriptor;
- control;
- graphical primitives;
- attributes;
- escape;
- external;
- segment.

5.1.1. Delimiter

The delimiter class counts 17 elements that serves to divide up the metafile into its structural components. The 'Begin Picture' element, for example, is part of this class.

5.1.2. Metafile Descriptor

The 23 elements of the metafile descriptor class appear at the beginning of the metafile to announce features and characteristics which apply to every picture in the metafile. They provide information which assist the interpreter understanding the

metafile. They even help it decide if it can decode the metafile at all. Examples are metafile version element or font list element.

5.1.3. Picture Descriptor

There are 19 elements in the picture descriptor class. They supplement the metafile descriptor with information specific to one picture like background colour or scaling information.

5.1.4. Control

These 16 control elements define viewing aspects applicable to all graphical primitives. This includes the definition of clipping rectangles or coordinates precision.

5.1.5. Graphical Primitives

There are 29 graphical primitives in the CGM standard which define the drawing objects supported by the standard (geometrical or raster objects). Examples include 'Rectangle' or 'Polygon' elements. (Cf. infra)

5.1.6. Attributes

The 51 attribute elements influence the way the primitives are drawn; they precise the actual appearance of the graphical primitives. Examples are line width or colour.

5.1.7. Escape

The escape class contains two elements to access non-standard graphical primitives or attributes. It is recommended that the elements of this class are use only in a registered context (Registration will be introduced shortly).

5.1.8. External

The two external elements permit the communication of non-graphical information, for example application specific data.

5.1.9. Segment

Segments permit the grouping of collection of graphical primitives and attributes that are then manipulated as a single entity. Segments provide a convenient shorthand way of referencing group of elements. Segments can be defined both at the metafile level (where they are shared by all the pictures of the metafile) or at the picture level. There are 7 segment elements.

5.2. Vector Graphical Primitives

Historically, this is the most important aspect of CGM. Out of the 29 graphical primitives, 26 are vector oriented. In its version 3, CGM provides a thorough coverage of vector oriented primitives found in common graphical packages.

Three types of vector oriented primitives can be identified:

- line and curves;
- filed areas;
- text.

5.2.1. Line and Curves

CGM supports all the graphical primitives found in common graphical packages. This includes:

- polyline, circles, circular arcs, ellipses and elliptical arcs as found in presentation graphics and general computer graphics commercial products;
- hyperbolic arc, parabolic arc which are in common use in engineering products (CAD);
- spline curves including Bezier curves, polynomial B-splines and rational B-splines widely used in graphics arts packages and in font description.

  Bezier curves and B-splines are very powerful equations that can describe any curve given enough resources. Bezier curves were popularised by PostScript® which uses them to describe its fonts.

  From a mathematical point of view, this makes all the other line and curve primitives redundant! Nevertheless, less generic primitives are supported by the standard because they are shorter and more efficient to draw. Furthermore, every graphical package uses them so that their absence would force tedious conversions (or more probably disinterest for the CGM standard).

5.2.2. Filed Areas

In many pictures, we need to fill an area in some way: either with a colour or with some pattern. The CGM standard allows for a number of different specifications of filled-areas:

- basic shapes: circle, rectangle, closed circular arc, ellipse, closed ellipse arc. These are simple, short and easily computationable definitions of simple area to fill;
- polygon and polygon set: to draw more complex filled-area;
- closed figure: the closed figure was introduced in version 2 of CGM and subsumes all other filled primitives. It is a composite primitive made up by connecting individual primitives. Therefore they can have any shape as they can include Bezier curves. The closed figure is considered as a filled-area element.

5.2.3. Text

CGM offers good support for the drawing of text. Including the support of both ISO 646 and ISO 2022 character sets and advanced support for font description.

ISO 646 is the "ISO 7-bit coded character set for information interchange". It is a simple character set which roughly corresponds to the ASCII standard. To be correct, ASCII is the American national code derived from ISO 646.

ISO 2022, the "ISO 7-bit and 8-bit coded character set — Code extension techniques", is more complex and provides, through an escape mechanism, access to an unlimited number of characters. How this escape mechanism works does not matter for us; what matters is that ISO 2022 permits the inclusion of text even when written in the enormous Japanese Kanji character set.

The standard also offers good support for fonts i.e. the typefaces or actual drawings of characters. Font examples are Helvetica, Letter Gothic or Palatino.

Fonts create specific problems in an open environment because an interpreter cannot reasonably be expected to have access to all available fonts†. It is often impossible to include a font definition in the metafile because most fonts are copyrighted.

The standard tries to resolve this problem by allowing the interpreter to substitute a font it cannot reproduce with another similar one. To help it, there is a Font Descriptor element that passes enough information to the interpreter so that it can wisely choose a substitute font.

---

† ISO sets up a registration scheme for fonts and expects 50,000 of them to be registered!

5.3. Raster Graphical Primitives

Originally, CGM offered very limited support for raster image data in the form of the Cell Array primitive. The Cell Array is a parallelogram defined by a bit map.

Cell Array is usable with very limited amount of raster data only. It was not designed as a real raster primitive. In particular it offers no real compression (although the encodings use a limited compression technique to actually store the bitmap).

To overcome this limitation and make CGM a full blown raster graphics format, two new raster primitives that support compression were introduced in version 3.

Many compression schemes have been devised that achieve various levels of savings at various computational costs; most *de facto* standards use at least one of them.

The new CGM primitives were designed for easy importing and exporting to and from major *de facto* standards. Therefore they support many compression techniques so that an image can be exported to and from a CGM without being first decompressed and then recompressed. From most *de facto* standards, it suffices to copy the compressed data without any other processing.

> *It is not particularly costly to put a CGM-specific wrapper around 250 Megabytes (Mbyte) of compressed data, or even to do byte reversal on the data as it is stuffed into the metafile, but it is considered unacceptable to have to uncompress the data from a standard source and then recompress, in order to put into the metafile [6]. [6]*

6. Encodings

Before explaining the encodings, we first need to define some vocabulary. We saw in the previous section that each element of the CGM standard is defined by a name and a potentially empty list of parameters.

To encode them elements are assigned *op-codes*. All three encodings define opcode for every element and provide rules to encode the parameters.

Op-code	Parameters

In the clear text encoding, it gives something like:

Rectangle	(12,25) (25,60)

;

The three encodings are defined for total compatibility with each other. They all implement the functionalities of part 1 of the standard. Therefore they are intertranslatable: a CGM in one encoding can be translated into another encoding without loss of information.

6.1. Clear Text Encoding

The goal of clear text encoding is to provide an encoding humans can read, write and edit. Therefore the rules are simple and flexible but, since it must remain computer processable, they are also unambiguous. This encoding has the flavour of a programming language. Its main characteristics are:

- it uses only characters, as defined in ISO 646, for coding;
- it provides a mechanism for the insertion of comments;

- it provides formatting facilities (the characters '_' and '$' can be interspersed for readability; text formatting characters like CR, LF, etc. are interpreted as empty characters);
- op-codes are encoded as meaningful English names while parameters are encoded as plain text;
- it represents numbers as their corresponding ISO 646 characters (the number 646 is encoded as the string '646').

Therefore this encoding is the most difficult to read... for computers.

6.2. Binary Encoding

The binary encoding's goal is speed of processing although compactness has not been forgotten. It is close to the representation that most computers use internally. Of course, probably not a single computer uses this representation exactly but the encoding is close enough to common practices so that the required transformation remains minimal.

The metafile is physically a stream of bits with each bit combination legal.

6.3. Character Encoding

This encoding trades ease of processing in favour of communication issues. Its design ensures that:

- the encoding does not produce any bit combinations reserved by some archaic communication protocols for exchange control (ACK, NAK, etc.);
- the encoding strikes for compactness;
- the encoding is relatively invulnerable to errors.

The metafile is still a stream of bits but the encoding algorithm ensures that, when the bits are grouped into bytes, only those which have an equivalent in ISO 646 (excluding some control characters) are used. Although this encoding can be opened and manipulated with an editor, it is not really human friendly.

The syntax might better be called "character-coded binary"[6].

Depending on the type of data, character encoding can be up to 25-30% shorter than the binary encoding.

The encoding is also designed to permit data recovery from a corrupted file. The binary encoding is very sensitive to loss or distortion of data: if one element is distorted, the complete metafile can be unreadable. The character encoding takes great care to limit the effect of a distortion to the corrupted element only.

6.4. Private Encoding

Since the standard specifies abstract functionalities independently of the encoding, it is possible to write a CGM in a private encoding that follows the principles laid down in part 1 of the standard.

This is usually not a good idea since it destroys most benefits of using a standard in the first place. In particular the file requires specific interpreter and generator or it must be translated before being processed by common CGM software.

Nevertheless if, for whatever reason, a proprietary format that features easy exchange to and from CGMs is necessary, a proprietary encoding is probably a good idea: conversion will be easy as both formats will be functionally equivalent.

7. Non-Standard Standard Elements

The standard recognises that users may have special needs for which nothing is normally provided and it leaves open doors by mean of three elements:

- Escape Element;
- Generalized Drawing Primitives (GDP);
- Application Data.

These elements are really a standard way to do non-standard stuff. Consequently their use should be restricted to elements defined and published by a user group maybe in a profile (profiles will be introduced shortly). We will also consider the CGM registration scheme.

7.1. GDP and Escape Element

GDP and Escape Element are similar. Functionally, their first parameter is a code (registered or not) that tells the interpreter what comes next. GDP are used to define new graphical primitives and Escape Element to define new attributes and controls.

7.2. Application Data

The Application Data element is used to store non graphical information pertinent to an application. CAD-like data, for example, can be stored as application data.

7.3. Registration

The use of any of these non-standard elements requires that the developers of interpreters and generators agreed prior to any exchange can take place. Since their use is not standardised, these elements bring all the problems of a non-standard format. In particular if the programmers fail to agree, no exchange is possible.

When an Escape, GDP or Application Data element is agreed on, in a user community, it might be registered i.e. an international registration body will give it a worldwide unique identifier and publish its specification.

It eliminates some of the problems raised by the absence of standardisation. In particular, the specification of a registered element is known (or at least accessible) by any interpreter developer. This is supposed to give a good karma to the generator that writes such an element.

Notice that the CGM standard is not the only international standard which resorts to registration.

8. Profiles

The CGM standard is big. Therefore some developers deliberately forget those elements of the standards they found irrelevant to their targeted audience.

There is no harm if a generator never uses a particular element because it does not need it. But an interpreter which is unable to understand certain elements limits the range of metafiles it can read. If there is no agreement between developers of interpreter and generator on which element can be safely dropped and which one must be kept, exchange quickly becomes impossible. This can obliterate all the benefits of a standardised approach.

Also the standard contains some imprecisions, there are ambiguities. And the standard is flexible on certain aspects. In short, some issues must be resolved by the developers. Of course, if they are left on their own, no two developers will resolve them in the same way.

There are also the Escape, GDP and Data Application elements which, as we have just seen, require an agreement between generator and interpreter developers.

All this raises the need for an extra standardisation step. But this step is best left to user groups than imposed by a standardisation body. In the CGM vocabulary, user agreements on the use of the standard are called profiles. The first amendment to CGM:1992 adds rules to help the designer of CGM profiles.

Ideally they should be as few profiles as possible, since in a way every new one reduces the openness of the standard.

8.1. Principal CGM Profiles:

The most important CGM profiles are:

- CALS which is studied, as an example, in the next sub-section;
- MAP (Manufacturing Automation Protocols) and TOP (Technical Office Protocols) working groups: these groups from the manufacturing sectors aim to design a complete OSI compliant open environment. They were the first to select the CGM standard;
- regional groups, like the European Workshop for Open Systems (EWOS), also issue some works in the area of CGM profiles;
- ODA specifies a (very) limited subset of CGM in its FOD26 and FOD36 profiles;
- CGM Model Profile: the amendment 1 of the 1992 edition also defines a profile for developers who have no other specific profile in mind.

8.2. CALS Profile

As an example of profile, we will consider the CALS profile as defined in the MIL-D-28003 standard which is currently at its revision A. CALS uses CGM for graphics that are mostly vector oriented. Graphics containing small amount of raster data can also be encoded as CGMs. Remember that for purely raster images, CALS selected CCITT Group 4 fax and for model data exchange, the choice is a subset of IGES.

The profile supports the three versions of CGMs. The general emphasis is on fidelity and predictability of the final image therefore CALS often reduce the flexibility of the CGM standard. MIL-D-28003 limits the use of particular parameters and sets maximum for some values. It also defines three conformance classes corresponding to three sorts of pictures:

- monochrome;
- grey-scale;
- colour.

Some line styles and patterns which are relevant to the CALS community (line styles and patterns used in engineering) are also defined and have been registered.

9. Other ISO Graphics Standards

In articles and books on CGM, a few other ISO standards are often referenced. The most important one is the Graphical Kernel System (GKS) which is another graphical standard from ISO; it specifies a set of functions for computer graphics independently of applications and devices. GKS functionalities can be mapped onto CGM.

> *The Graphical Kernel System (GKS) is a system for two-dimensional graphics and provides no support for three dimensions. The Graphical Kernel System for Three Dimensions (GKS-3D) is an extension of GKS to provide basic functions for computer graphics programming in 3D [2].*

Another relevant standard is the Computer Graphics Interface (CGI).

> *GKS standardizes the interface between graphics systems and application programs, but leaves open the question of communication between the graphics system and the workstation or device. It is this graphics workstation interface, however, along with its environment, which is the subject of the Computer Graphics Interface standard... [3]*

CCITT Group 4

10. Group 4

A long study of group 4 would be redundant with other sections of this chapter. Furthermore it is not a very interesting standard from our point of view. As we already mentioned, group 4 is a compression scheme for raster graphics; it was adopted by CALS as a quick migration path to electronic form for legacy documents.

Nevertheless we mention group 4 for the sake of completeness and because it provides us with an opportunity for a brief discussion on compression schemes. As we already mentioned, efficient compression is particularly important with raster graphics which tend to be very large.

The group 4 compression strategy was originally defined by the CCITT for fax transmission. It was published as FIPS PUB 150 (CCITT Recommendation T.6). Group 4 is limited to black and white images only which is consistent with its fax background. This is not a problem in the CALS environment where it is used to compress scanned pages only.

Many techniques have been devised to compress data. The efficiency of a particular method is highly dependent on the sort of data it works on. For example some methods will be more efficient with texts while others are at their best with sounds or graphics. Techniques that target a specific data type, like group 4, can take benefit from particular data properties and achieve impressive compression ratio. For example, group 4 reduces the overall size of an image by a mean factor of 40 compared with a brute force approach [9] .

The idea behind group 4 is based on a statistical analysis of the data. In a typical page certain patterns of pixels appear more frequently than others. For example white areas are more frequent than black ones. A brute force encoding stores every pattern with the same number of bits. A statistical encoding will use variable length bit sequence; shorter sequences are used for more frequent pattern while the rarest are encoded with longer sequences. It is hoped that on a normal page the overall size will be reduced significantly. The CCITT had to run extensive analysis of typical pages to select the most appropriate bit sequence for every pattern of pixel. This work resulted in the group 4 compression scheme.

MIL-R-28002B defines two file formats to store group 4 compressed images.

IGES & STEP

11. CAD Models

Apart from raster and vector formats, we identified a third type of graphical data: product model data as manipulated by CAD systems. Strictly speaking product models are not restricted to graphical information but the most common way to use them today is through graphical representations; therefore they are often referred to as graphical format.

The bases of Computer-aided design (CAD) are the model and its companion notion, the representation of a model. At first sight, CAD can be understood as the use of computers to manipulate representations of engineering models. A representation is the particular form in which a model is expressed. One can only manipulate a model through one of its representation.

During the design process, the design is abstract: no artefact exists. Therefore the designer needs some model of its design to work on. It serves primarily two roles in engineering:

- a reminder of the design for the designer,
- a communication mean between the people involved in product production.

The design process is a repetitive activity where models are drawn, assessed, amended and drawn once again. They can take several forms or representations. In the earliest stages, it can be an idea in the designer's head but, as design refines, a complete and formal description is often required. Also, the representation frequently needs to be adapted to a particular work. A major design activity is the drawing of new representations based on old ones. For example, the representation will differ whether aesthetic analysis or technical assessment is undertaken. In the first case, photorealistic representation of the artefact is suitable while an annotated draughting is preferred for assessment.

> *The term* ***product data*** *denotes the totality of data elements that completely define the product for all applications over its expected life cycle. Product data includes the geometry, topology, relationships, tolerances, attributes, and features necessary to completely define a component part or an assembly of parts for the purposes of design, analysis, manufacture, test and inspection[1]*

The importance of formal models and the number of representations used tend to increase when the device is complex and/or when a large team is involved in the design process. So does the management complexity.

Product models share some properties with vector formats. Both are a description of something in terms of primitives. But the description of a product is more complete; it may include information on volume (solid modeling), size, composition, etc. of the artefact. Therefore files are bigger which is the reason why CALS provides both a vector format (CGM) and a product data model format (IGES): CGM is intended to store an image of an artefact when one does not need the whole model, like pictures in manuals.

11.1. Models for Form or Structure

11.1.1. Form & Drawings

There are many engineering activities. Each has its own constraints. Two properties are recurrent to most design, namely the form and the structure. Each is best rendered by a particular type of model.

Form is the most important property in most engineering activities. This is the case for the design of cars, bridges, buildings, etc. Form is normally modelled by drawings. This is usually achieved by a process know as descriptive geometry or *Mongian projection*:

> *Three-dimensional forms are represented in two dimensions by mapping points on the object into multiple mutually perpendicular planes of projection using parallel projectors that are normal to the planes of projection. From the projection of* ***points*** *may be derived the projection of* ***edges*** *of the object, and from the* ***edges*** *the* ***surfaces*** *that bound the object[7] [7] .*

Traditionally a solid object (a hammer in figure 6) is modelled as a set of 2D models:

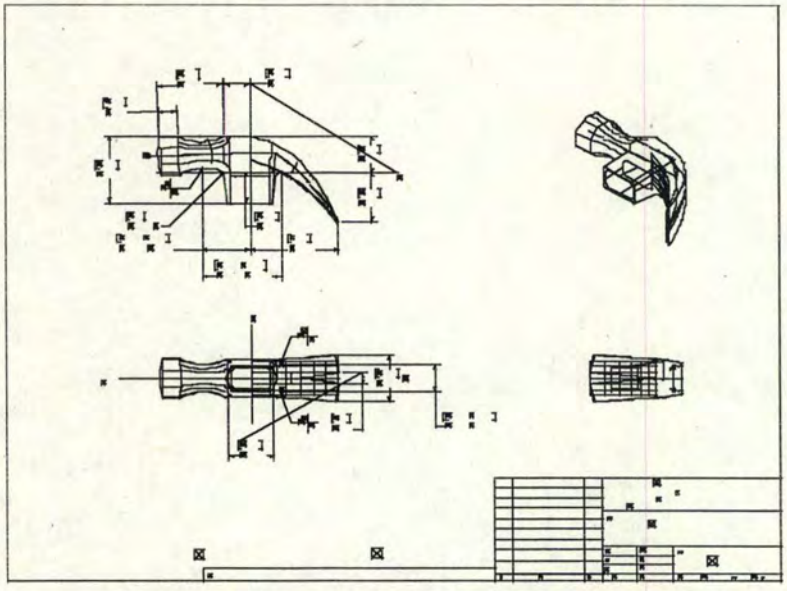


Figure 6: A hammer modelised as a set of 2D models

11.1.2. Structure & Diagrams

Other engineers are more concerned by the assembly of standard elements as in the electronic industry (this does not mean that electronic engineers are unconcerned by the form of their circuits, rather that it is relevant only in last steps of design).

When the most important property of a design is the structure of the device, diagram models are used:

> *In engineering diagrams the logical or physical structure of a system, is shown by a series of* ***symbols*** *joined by* ***connections*** *[7] .*

Figure 7 is a sample diagram of an electronic device (a multi-vibrator) [11]:

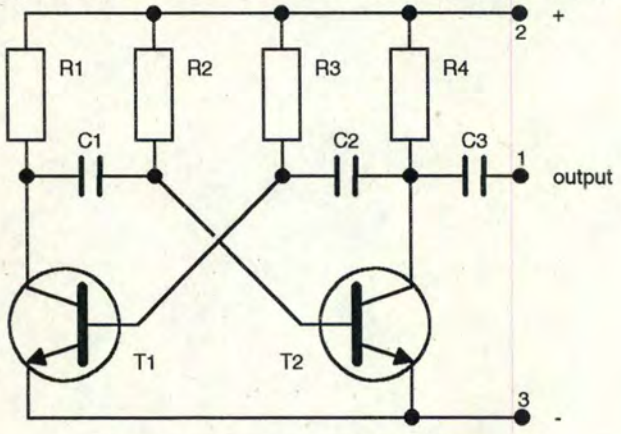


Figure 7: Electronic device diagram

11.2. From Pen to Plotter

We will conclude this introduction to CAD models with a short overview of the history of CAD systems. CAD developed in four major steps:

- graphic assistance;
- 3D modelling;
- modelling of non geometrical properties;
- information exchange.

This evolution reflects the two trends commonly found in the development of information technologies, i.e. the move towards:

- the integration of more data (here, 3D models and non geometrical data) in corporate database;
- the availability of the data to those who need it (communication).

11.2.1. Graphical Assistance

The earliest CAD systems were a substitute for the classical designers' pencils and erasers. When compared with a piece of paper, a computer drawing package offers erasing facilities and greatly eases repetitive drawings.

The initial computer-made design may take as long to draw as its paper-based counterpart but computers dramatically simplify amending a design: a paper-based draughting must be entirely redrawn when a big change is required while with computer draughting only the actual change needs to be redrawn.

Also, with a CAD system, a complex motif (say a gothic window or a transistor), that had to be repeated identically on various places of the drawing needs to be drawn only once. With a pencil, it must be redrawn every time it appears.

The CAD systems of this category only provide a replacement to artisanal draughting. When designing a solid object, the model does not include volume information in itself, the volume is modelled by a set of 2D drawings. This means that nothing is done by the tool to ensure the coherence of the various draughtings. This of course was the major improvement in the next generation of CAD systems.

11.2.2. Solid Modelling

With solid modelling systems, the model of a solid object is no longer a set of 2D representations, the volume information is included in the model itself. This is a better model of the device since it allows the computer to test the model coherence.

Of course the solid model is mapped to a 2D representation (the screen or paper) for drawing or editing. In other words, the computer is able to produce the classical 2D projections but, because they are made out of a plain-vanilla 3D model, the computer *ensures* that 2D projections made from different angles are coherent with each other.

This is a clear illustration of the difference between the model and its representation. The model, i.e. the mathematical data that store the properties of an object, is solid while the representation on screen is flat.

With these two generations of systems, non geometrical properties such as composition, part references or resistance to workload are added to the model as annotation texts.

11.2.3. Modelling of Non Geometrical Properties

The next improvement is to incorporate non geometrical properties in the model; properties like the materials used, supplier reference, etc. This information is interpretable by the computer. Hence some automated analyses, like resistance to workload, are possible. Also the system can generate bill of materials, price predictions or helps in planning.

With these systems clearly the model is no longer only graphical.

11.2.4. Information Exchange

We saw that one of the main purpose of models is to allow information interchanges between the various people engaged in the design process, e.g. the designers and the manufacturers:

> *The design process is rarely undertaken by a single designer, however, and therefore the models have a major role in the* ***communication*** *of the design between participants in the process, and those involved in the manufacture, development and subsequent use of the product [7] . [7]*

An important part of the design process is the generation of models based on other models for assessment, manufacturing, etc., e.g. a simpler model is extracted from a complete description to present a product to a potential customer. This generation process gains being automated. The automation is often done by specialised tools, different from the design systems, which must access the design.

Hence we identify two types of information exchange, amongst:

- the various team members (humans);
- the software tools (computers).

By enabling information exchange we ease both applications.

Integration is much concerned with communication, i.e. tools or team members exchanging data. The product management is integrated through the product lifecycle. The technical database can thus store data concerning design but also manufacturing, maintenance and commercial data.

11.3. Computer-Aided Manufacturing

EDI is not directly concerned with Computer-aided manufacturing (CAM) because, if CAM does imply data exchange, it is a highly system dependent one. Nevertheless, we have seen that CAD tends to integrate with manufacturing. A brief discussion of CAM will illustrate this trend.

The manufacturing machines in modern factories are usually numerically controlled devices. In other words, they are generic tools adapted to a specific production by mean of a program (this is called numerical control or NC) in sharp contrast to their ancestors which where designed for one sort of production only. Today when one wants to change the machine's production, one just has to change the program, like switching from a word processor to a spreadsheet on a PC.

Keep in mind however that if this gives great flexibility, practically those programs are tedious and difficult to develop.

> *The use of NC could only be justified, therefore, if the cost of creating a part program could be spread over a long production run or if the job required a degree of precision which could not be achieved by any other means [8] .[8]*

One way to overcome this problem is to develop computer systems to assist in programming. Some systems receive their data directly from the CAD systems. In this respect CAM is linked with CAD. Some product models are transferred, as

CAD data, from the R&D center to the factory potentially using techniques we will shortly examine.

11.4. Product Data Exchange Standards

We will study two standards for the exchange of product data model:

- IGES: the Initial Graphics Exchange Specification is the ANSI standard CALS elected;
- STEP: the Standard for Exchange of Product Data is a draft ISO standard.

Although STEP is still under development, it is felt that it will play a significant role in the near future. It is intended to replace IGES and the other standards for product data model exchange. Hopefully STEP will overcome many limitations of IGES. Therefore, for product model data, we choose not to restrict ourselves to the CALS standard due to the importance of the STEP initiative.

12. Initial Graphics Exchange Specification

12.1. The Standard

IGES, which stands for Initial Graphics Exchange Specification, is probably the first significant work in product data exchange. The effort started in 1979, supported by the US National Bureau of Standards. The standard was developed mainly by major US CAD vendors. Eventually it was adopted as an ANSI standard (ANSI Y14.26M).

The standard originates from the mechanical engineering industry. It has gone through five revisions, each one broadening the field of application. Now IGES encompasses such areas as architectural & construction, electrical applications, solid modelling, etc. Not all of these revisions have been adopted by the ANSI as standards. Since version 3.0, IGES provides for solid modelling data transfer.

IGES defines a neutral format. CAD systems work with their own formats which are then translated to and from the neutral format. Of course this raises problems since, as we all know, converting data often means losing information. This approach contrasts with the approach taken by the new ISO standard, STEP (to be introduced shortly).

In the IGES terminology the software which converts a proprietary file into an IGES one is a pre-processor; the post-processor does just the opposite and creates a proprietary file from an IGES one.

The underlying concept of IGES is the entity i.e. an element of the model. A companion notion is the entity type i.e. a family of entities which share a common definition. It is not uncommon to find the word 'entity' used with the meaning of 'entity type'. When the context is unambiguous, we will use entity with the latter meaning in this discussion of IGES.

12.1.1. File Structure

IGES files come in three flavours:

- the ASCII form is the original form. It was designed for easy exchange on a ½" tape, which was a popular archival medium when the IGES format was first written. ASCII files tends to be much larger than their proprietary counterpart;
- the compressed ASCII form attempts to produce smaller files;
- the binary form is shorter and quicker to handle.

The file organisation is similar in every format. In particular, the same set of entities is supported by the three versions. In this document, we will describe the ASCII format only. This suffices to give the reader a good understanding of the standard and what it can be used for.

An ASCII file is partitioned into 80 character-long lines. These lines are grouped in sections. There are six sections in IGES files. Apart for the flag section, every section must appear once and contain at least one line. Their order is also fixed by the standard.

The six sections are (see also figure 8):

- the flag section is present only in compressed ASCII and binary file; it is used by the post-processor to recognise a compressed ASCII or a binary file;
- the start section is not used by the post-processor. It is an human readable prologue to the file i.e. a text, written by the person who initiates the exchange, to help the receiver decoding the file, some sort of comment one may say;
- the global section provides information on the file, such as sender's identification, filename, name and version of the IGES pre-processor, etc. and parameters to decode the file, for example, IGES version, integer or floating-point precision, maximum line thickness, largest coordinate value, etc.
- the directory section contains an entry (a record) for each entity (each instance of an entity) in the file; it points to associated data in the next section;
- the parameter data section stores entity-specific data, such as coordinate values, annotation text, etc. Each entry points to the corresponding directory section entry.
- the terminaison section ends the file.

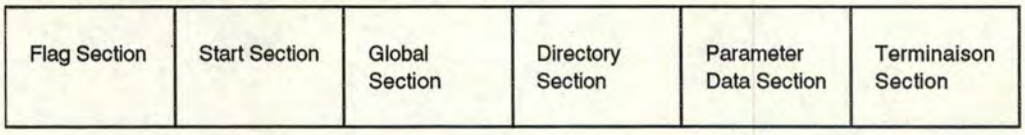


Figure 8: IGES file structure

Each line has an identifier in columns 73-80. The first character of the identifier is a letter which indicates the section of the file (start section is indicated with an S, directory with a D, etc.) and the last 7 characters are the line number. The first line of each section is always numbered one (1), the following lines are numbered in an ascending sequence up to the number of lines in the section.

This structure is depicted in figure 9.

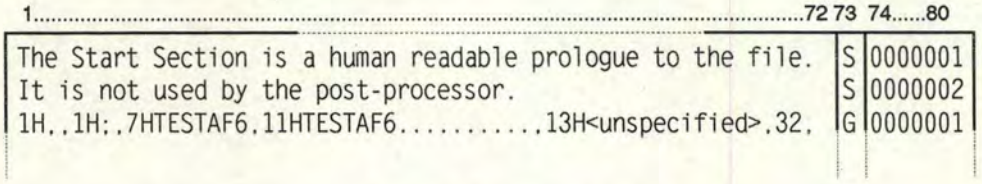


Figure 9: Format of a line

12.1.2. Elements Overview

In this document, we won't study every single entity type supported by IGES. Refer to the standard for a complete description. As usual we limit ourselves to an overview of what can be stored in an IGES file.

A product model in IGES is represented as a set of entities where each entity stores a given property of the product. The standard defines over 70 entity types which cover every kind of entities commonly found in CAD systems. The entity types are grouped in three classes:

- geometry entities represent physical shapes. Examples are points, curves, surfaces or solids;

We saw that a product model is not limited to graphical information. Therefore IGES supports entities for non-geometric data which provide additional information and enrich the model:

- annotation entities are used to enhance or clarify the geometric part of the model. Examples are entities to indicate dimensions, text, etc.
- collection entities organise geometric and non-geometric entities in a single logical unit which must be manipulated as a whole. This is particularly important in models where a part consists of two or more sub-parts: each sub-part can be modelled as a separate collection of entities.

Most entity types can exist in more than one form. For example, a plane entity can exist in any of those three forms:

- unbounded plane;
- bounded plane;
- bounded void in a plane.

Therefore there is only one definition of plane in the standard which can be instanciated in three forms instead of three different entity types. This limits the number of entity types that must be defined in the standard.

IGES entity types are a neutral definition to and from which CAD systems map their own entities. Here lies one of the major criticism against the neutral format approach: mapping from a proprietary definition to the neutral one is not always trivial, it will often result in information loss if great care is not taken.

For example, consider a proprietary format which supports a very expressive entity absent from the neutral format. To create the IGES file, the pre-processor may map this proprietary entity to a set of IGES entities. Alas such an operation is usually not reversible: the post-processor, short of conducting heavy analysis, will not be able to map the set of simple entities to a more powerful entity. In the process information is lost because in most case a complex component carries more information than the sum of its parts. STEP attempts to overcome this problem. We will consider STEP in the next section.

Appendix A lists the most significant entities supported by IGES.

12.1.3. Extending The Standard

The standard defines entities for information manipulated by most current CAD systems.

But it may not suffice. This holds particularly true if IGES is used for archival. In which case the sending and receiving systems are the same. In this special case mapping proprietary entities to their standard counterpart may cause an unacceptable burden. A system may use special data. For these reasons one may want to extend the standard and, for these reasons, the standard allows the definition of implementor-defined entities. It is not a recommended practice as it will inevitably raise problems as soon as another system tries to edit the file. The standard recommends the use of the macro facility instead. A macro is a non-standard entity described in term of standard entities.

12.2. Subsets

IGES is a huge standard. It is rare that an implementation covers it completely. Most implementors limit their support to a subset of entities. Therefore problems arise when two implementations using different subsets try to exchange data. There is nothing new under the sun about this.

Subsets are often defined around a particular application type, for example a subset suitable for electrical application or building.

To resolve incompatibilities some organisations further standardise on subsets. CALS MIL-D-28000A is such a standard. Other significant subsets include NASA 28 entities and VDA IS.

MIL-D-28000A defines five subsets (classes) tailored for different usages. It also restricts the exchange to ASCII files only, sets limits on the value of parameters and restricts the set of entity that can be present in a file. The classes are:

- Class I - *Technical Illustration Subset* is used to encode figures and illustrations normally found in a technical publication. It emphasises on visual clarity of figures and illustrations designed for human interpretation;
- Class II - *Engineering Drawing Subset* which addresses the exchange of product data acquired in accordance with another military standard, MIL-T-31000. MIL-T-31000 is not a CALS standard. The subset puts the emphasis on completeness, visual equivalency for human interpretation and functionality of the received drawing;
- Class III - *Electrical/Electronic Applications Subset* is for the exchange of data for electrical and electronic products. It stresses on component and circuit element descriptions, their placement, their connectivity and the routing of electrical paths. The subset supports both the physical view and the logical view (i.e. a diagram) of the product;
- Class IV - *Geometry for NC Manufacturing Subset* encodes product data for the subsequent purposes of manufacturing by numerical control;
- Class V - *3D Piping Application Protocol* is devoted to the exchange of three dimensional piping and related equipment models.

13. Standard for Exchange of Product Data

IGES is not the only proposition for product data exchange. For example, the French company *Aérospatiale* developed a standard of its own, SET (*Standard d'Echange et de Transfert*) that was eventually adopted by the AFNOR, the French standard body. The German automotive industry also developed a standard, VDA/FS. All these efforts try to overcome perceived limitations of IGES.

All this work serves as a basis for the development of an internationally accepted standard supported by the ISO: the Standard for Exchange of Product Data project, STEP.

With STEP we choose to depart from our CALS orientation because it is commonly agreed that STEP will have a significant impact on data exchange once completed. Also STEP is based on a different approach than the current standards like IGES. The DoD has already announced that it is its strategy to include STEP in the CALS initiative when STEP will be available.

In the US, STEP is known as PDES because the PDES standard (Product Data Exchange Standard), which originates from the IGES organization, served as a basis for the development of STEP. PDES was later renamed Product Data Exchange using STEP.

At the time of writing, STEP has attained the status of Draft International Standard under the label ISO 10303. Due to its size the standard is split across various documents (parts). The parts are developed and standardised independently; it simplifies maintenance and quickens development (since the whole process is not hampered by the weakest part). Appendix B lists the various parts of the standard.

13.1. The STEP approach

We saw that the neutral format approach adopted by IGES is a source of problems, because it lacks a standard model which would be used by every CAD system. Instead systems have to map their own model to and from a neutral representation.

The reader will remember our CGM discussion when we said there are many mathematical tools to compute a curve definition. It is not such a big problem with images because most drawing applications support all of these definitions.

Solid modelling raises a similar but more disturbing problem. Briefly, there are two ways to model solid parts:

- boundary models which describe a solid by combining geometric information about its faces, edges and vertices with topological information on how these are connected;
- constructive solid geometry (CSG) which describes a solid as a combination of simpler solids such as cylinders, rectangules and the like.

Other techniques exist but those two are by far the most common.

Although both techniques can be used to model the same parts, they exhibit sharp differences in their mathematical properties which make them somehow antagonists. IGES supports both techniques but if we try to exchange a CSG model to a system which supports exclusively boundary model, the post-processing will be impressive and probably will give poor results. Moreover it is impossible to convert automatically data from one representation into the other without loosing part of the information.

Writing an IGES file is not difficult because the pre-processor is likely to find standard entities for every entity used by the originating system. The real problem is to read a file. With no agreement before the exchange to specify which entities can be used, it is unlikely that the target system will have a direct mapping for every entity used in the file. So the post-processor will resort to conversion.

We can think of an analogy. Suppose Philip wants to send a congratulation card to Arthur. Philip only speaks English while Arthur only speaks French — clearly any exchange is impossible.

Fortunately both Philip's and Arthur's secretaries speak German. Philip may ask his secretary to translate his card in German before sending it. Arthur's secretary will then translate it in French for Arthur to read it.

But we all know that translating from one language to another one is not trivial. A word can have more than one translation, there is room for interpretation. This holds particularly true if there is a game of word or similar subtleties. On the whole the translation will differ slightly from the original.

It is even worse in this example because the card is translated twice to and from a neutral format (German). None of the secretary is a native German speaker thus they are likely to make even more mistakes. Differences will accumulate throughout the exchange. That is the neutral format approach.

STEP relies on a different strategy. STEP recognises that the neutral format approach is inadequate for product model data exchange; it improves on previous approaches by standardising a conceptual schema which enables applications to access data in a predictable way. In other words, STEP standardises the model data as it is used by the applications. In this document, we will particularly concentrate on this innovative aspect of STEP.

In a normal use of STEP the model will always be expressed as a STEP model. Therefore data conversions are limited to a minimum, maybe only to conversions between different representations of numbers. In this case both Philip and Arthur

speak German or English or French or whatever language. They have a common language.

> *In a STEP system, access to data is by using the conceptual schema, which enables foreign systems to access data in a predictable manner. In fact, if the data in a system can be accessed as if it were created from the STEP schema then it is a STEP system, regardless of whether the data was actually generated directly from those models [4].*

13.2. The Standard

Before going any further we must introduce two notions, namely the Express language and the STEP Data Access Interface.

13.2.1. Express

Express is a modelling language specially developed for STEP. It is used to define formal entities. In Express an entity is a collection of data, associated constraints and operations on the data. Every entity definition which is part of the standard is written in Express. A collection of entities definition is known as a schema.

Here is a potential definition for a point in Express:

```
ENTITY point;
   x_coordinate: real;
   y_coordinate: real;
   z_coordinate: real;
END_ENTITY;
```

meaning that a point is composed of three reals.

The language is system independent: it refers to abstract types (`string`, `integer`) which are then mapped to a system dependant representation. In the previous example, it is referred to a `real` type whose actual representation is system dependent. As opposed to the semantic which is common to all systems: a point consists of three reals whatever a real might be on a particular system. The semantic is system independent, the storage is system dependent.

In this example we see how a common model minimises data loss. Of course the precision of reals varies from system to system; therefore data will be lost when going from a higher precision system to a lower one. But, as opposed to the neutral format approach, the semantic remains consistent across systems.

For example mathematically there is also a vectorial definition of a point (made of an angle and a length). The point entity we wrote in Express explicitly excludes this alternate definition.

In STEP entities are formally defined and used consistently across systems: every STEP system has the same definition for a point. Of course this becomes particularly relevant with more complex entities than the point.

Notice that Express was not designed for automatic interpretation but as a formal notation for standard writers. Do not imagine systems where new entities are added simply by compiling their Express definition. Although Express is interpretable by a machine, it is not a programming language but a convenient notation for design. Indeed adding new entities to a system is likely to require reprogramming the system.

13.2.2. STEP Data Access Interface

STEP also standardises an interface to create, access and manipulate a model: the STEP Data Access Interface (SDAI). SDAI is at the heart of the STEP initiative. To overcome limitations of a neutral format approach, STEP defines a conceptual scheme i.e. a standard way to consult a model. Coherent with this approach is the standardisation of an interface to access, manipulate and modify STEP data. The

interface separates the storage specific issues from the rest of the application. We represent this in figure 10:

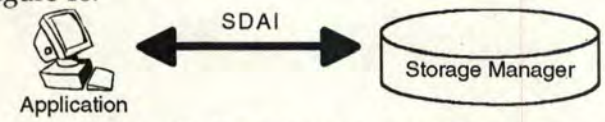


Figure 10: STEP Data Access Interface

The data storage is managed by the Storage Manager which implementation is not standardised. An application can consult the data through the SDAI.

This has some benefits:

- the storage manager can be used by several applications. We saw that the trend in CAD systems is toward data sharing where more than one application access the same data, potentially simultaneously. The separation of functions between the storage manager and the applications eases such an architecture;
- applications and storage managers can be developed independently. This will become increasingly important since, with the advance of data sharing, storage manager will become major pieces of software on their own. With STEP it is possible to develop powerful storage managers on top of which applications will come. Database developers do what they do best, they develop databases (SDAI compliant) and CAD developers benefit from their work through SDAI.

13.2.3. Levels of Compatibility

STEP supports four levels of compatibility. To comply with STEP an application must support at least one of these levels. The four levels are:

- Level 1: *File Exchange*, the data is exchanged as an ASCII file. At this level, STEP can be used like a neutral format;
- Level 2: *Working Form Exchange* which means that the data can be accessed in STEP format in memory. This level uses the SDAI;
- Level 3: *Shared Database* where the product model is directly stored in a database. Applications access the data through a standard database interrogation language (such as SQL) or through SDAI. This level implies a direct mapping of the STEP structure to the database records (or tables or objects depending on the database technology). Concurrent access by multiple users of multiple applications is now possible;
- Level 4: *Knowledgebase.* Knowledge systems are new but they will probably provide sophisticated tools to manipulate STEP models. They will probably be very similar to databases with the additional ability to be driven by some knowledge.

13.2.4. Resource Models & Application Protocols

The IGES standard is so big that it is rarely completely implemented forcing developers and users to further standardise on subsets. We examine the CALS subset as an example. This aspect may seem anecdotal but it is not, the use of incompatible subsets accounts for most of the problems when exchanging models with IGES.

STEP is an even bigger standard than IGES. One may reasonably fears that subsets will become a real nightmare with STEP. STEP addresses this problem by standardising subsets within the standard itself.

There are two sorts of subsets with different usages:

- resource models group entities related to the same area of modelling. Examples are Draughting, Ship Structure, Finite Element Analysis. Resource models group logically related standard entities;
- application protocols are application subsets. STEP identifies a number of application domains and defines corresponding application protocols.

13.3. Total Life-Cycle Support Using STEP

When we studied the historical evolution of CAD systems, we saw that the trend is toward the integration of more information in the product model. STEP is particularly interesting in this respect.

For one thing, STEP has (or will have) entities to store non-geometrical information in the model itself. In particular, STEP includes data models for approval, version and configuration, materials, supplier information and other product related information†.

For another thing, STEP is clearly oriented towards database technology. It supports databases and data sharing in a way which makes it possible for multiple applications to access the same model.

Therefore it is possible to have different applications which will enrich the model during the whole life of a product. When this approach will be implemented, the engineer will create an artefact model using a CAD system. The model will be extended by the manufacturer, the commercial, the manager each using its own application and sharing the same model.

STEP data will federate information coming from various sources throughout the enterprise and through the whole life-cycle of a product. This is very close to the CALS dream of Continuous Acquisition and Life-cycle Support.

---

† Part of this additional data may take the form of SGML text. This is still subject to debate in the STEP standardisation committee [10].

Other Aspects

14. Further Reading

To a reader wanting to learn more about computer graphics, we recommend the widely appraised *Computer Graphics — Principles and Practice* [5] (commonly referred to as Foley and van Dam).

To study CGM any further, we recommend those books:

- The CGM Handbook [6]: definitely the book to read if you are interested by CGM. It provides a in-depth coverage of the subject;
- CGM and CGI [1]: provides a good introduction to the subject;
- ISO Standards for Computer Graphics [2]: this book proposes a good introduction to the various graphics standards of ISO.

If the reader is interested by CAD & CAM and want to learn more about it, we warmly recommend the book *CADCAM — From Principles to Practice[7]* .■

15. References

[1] D. B. Arnold, P. R. Bono
CGM and CGI
Springer-Verlag, Germany, 1988

[2] D. B. Arnold, D. A. Duce
ISO Standards for Computer Graphics
Butterworths, UK, 1990

[3] Peter R. Bono, José L. Encanação, L. Miguel Encarnação, Wolfgang R. Herzner
PC Graphics with GKS
Prentice-Hall, UK, 1990

[4] R. Doty
Product Data Sharing Using STEP Technologies
Digital Equipment Corporation, US, 1992

[5] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes
Computer Graphics — Principles and Practice, Second Edition
Addison-Wesley, Reading (Massachussets), 1990

[6] Lofton R. Henderson, Anne M. Mumford
The CGM Handbook
Academic Press, London (UK), 1993

[7] Chris Mc Mahon, Jimmie Browne
CADCAM — From Principles to Practice
Addison-Wesley, Wokingham (UK), 1993

[8] A. J. Medland, Piers Burnett
CADCAM in Practice — A Manager's Guide to Understanding and Using CADCAM
Kogan Page, UK, 1986

[9] Christian Monteix
Le Format TIFF et ses Modes de Compression
Eyrolles, France, 1993

[10] Yuri Rubinsky
Technical Documentation in the World of STEP
CALS Journal, Spring 1993, p. 63-66

[11] J. Soelberg - W. Sorokine
Pratiquez l'électronique en 15 leçons
Editions Radio, Paris (France), 1986

Appendices

Appendix A: Some Entities Supported by IGES

Geometric Entities

Circular arc
Composite curve
Conic arc
Copious data
Plane
Line
Parametric spline curve
Parametric spline surface
Point
Ruled surface
Surface of revolution
Tabulated cylinder
Transformation matrix
Flash
Rational B-spline curve
Rational B-spline surface
Offset curve
Connect point
Node
Finite element
Nodal displacement and rotation
Offset surface
Curve on a parametric surface
Trimmed parametric surface

Annotation Entities

Angular dimension
Diameter dimension
Flag note
General label
General note
Leader (arrow)
Linear dimension
Ordinate dimension
Point dimension
Radius dimension
General symbol
Sectioned area

Structure Entities

Association definition
Line font definition
Macro definition
Subfigure definition
Text font definition
Text display template
Color definition
Network subfigure definition
Associativity instance
Drawing
Property
Singular subfigure instance
View
Rectangular array subfigure instance
Circular array subfigure instance
External reference
Node load/constraint
Network subfigure instance

Appendix B: STEP Parts

These are the various parts of the STEP standard. Parts are identified by a suffix number. For example, ISO 10303-41 is part 41 of the standard.

Part	*Description*
1	Overview and fundamental principles
11	Express language
12	Framework
21	Clear text encoding (level 1 file format)
22	STEP data access interface (SDAI)
31	Conformance testing methodology and framework: general concepts
32	Test laboratory requirements
41	Generic product data model
42	Shape representation
43	Product shape integration model
44	Product structure and configuration management
45	Materials
46	Presentation
47	Shapes tolerances
48	Form features
49	Product life cycle support

Resource Models

101	Draughting resources
102	Ship structures
104	Finite element analysis
105	Kinematics

Application Protocols

201	Draughting application protocol
202	Exchange of draughting with 3D geometry
203	Exchange of configuration controlled 3D product design
204	Exchange of boundary representation solid models
205	Sculptured surface models

6 Packing

The scientific theory I like best is that the rings of Saturn are composed entirely of lost airline luggage.

Mark Russell

It is a good time to stop, look back at the way we have come through and assess where we are going to.

We are studying the inclusion of technical data in an EDI standard, in particular EDIFACT. First we reviewed what technical data might be. Then we selected one initiative for the exchange of technical data, namely CALS, and set to study which standards it supports.

In the last 2 chapters we surveyed some standards for the formatting of technical data. We felt it was important to do so to have a better understanding of what technical data are and of typical formats in which they are stored. It has been an interesting study in itself but now it is time to revert to our EDI focus.

In this chapter and the following one we will see that specifications for the formatting of technical data, although a necessary condition for successful exchange, are not sufficient. We need other specifications to organise technical documents in logical groups, be able to reference them, etc. In this chapter we examine the CALS standard that fulfils this role. In the next chapter we will see how to bring all the things we have learned so far into a standard for technical EDI.

1. How to...

One may think it is desirable to extend EDI standards with new segments specialised in the encoding of a category of technical document. For example add vector and raster graphic or product model segments to EDIFACT. This approach is theoretically feasible but hardly practicable or even desirable.

For one thing, it would complicate the EDI standard. Remember the complexity of the standards we have just examined, take CGM which is a typical vector graphic standard. It is clear that, to be of any use, a hypothetical EDIFACT vector graphic segment would need to be of the same order of complexity as the whole CGM standard. This segment alone would be the size of a complete standard! Keep in mind that similar segments would have to be devised for every possible technical data type, the number of which is a continuously growing figure.

For another thing, it is a waste of resources. Both in the standardisation body which would uselessly duplicate the work of another standardisation organisation and for the user which would have to convert to and from the EDI specific format for the sole purpose of transmission. This is in sharp contrast to the most recent approaches for exchange of technical documents, as typified by STEP and SGML, which emphasise on a unique format for both exchange and storage.

And, last but not least, it would greatly slow down the inclusion of technical documents in EDIFACT. Just remember the history of the development of SGML as sketched in appendix A of chapter 4. Seventeen years have passed between the first introduction of Generalized Markup Language (1969) and the publication of the first version of the SGML standard (1986). The first working draft of the standard was written in 1980, six years before the final approval of the standard. SGML standardisation is not considered to have been particularly slow though. Remember that the same process would take place for every possible data type.

The only realistic approach for an EDI standard to earn the label *technical EDI* is to take advantage of standardisation efforts in the technical document arena. If there is a mechanism, in the EDI standard, to transparently include data presented in another format (either a proprietary or an international standard) the problem is half way solved.

There is an almost obvious solution to this problem: it suffices to consider technical data expressed in a non EDI standard as a binary stream, to incorporate the stream in the EDI interchange blindly and to deliver it unmodified to the recipient†. What one might call a black box approach since it treats non EDI data as a sealed black box. The data is extracted from the corporate database by appropriate pieces of software which output binary streams. The stream, the format of which is irrelevant to the EDI translator, is passed to the EDI translator which includes it in the interchange. At the other end, another EDI translator extracts the binary stream and passes it to appropriate software for further processing.

EDI translators are isolated from the peculiarities of technical document formats, similarly technical document formatters are isolated from the EDI interchange. Those formatters ignore they process data which, at one point, were part of an EDI exchange so that general purpose products can be used.

Notice that we talk of a binary stream solely because we refuse to open the black box and look at the stream, it may be that the non EDI data consist of ASCII characters. It will be the case if we include an IGES file, for example. More on this in the next chapter.

The data may already be in appropriate format in the database (this is likely if STEP is the chosen standard) or it may require a conversion from a proprietary

† This is why many people with EDIFACT background use the misleading term of 'binary data' in the meaning of 'technical documents' as used in this paper.

format to an international standard. In either case, the extraction and the possible conversions are handled by specialised software, not by the EDI translator which blindly passes the data to and from those translators.

Figure 1 shows an example in which companies *Up & Co.* and *Down Ltd.* exchange the model of a plane. It assumes the model is converted to the neutral IGES format for the transfer.

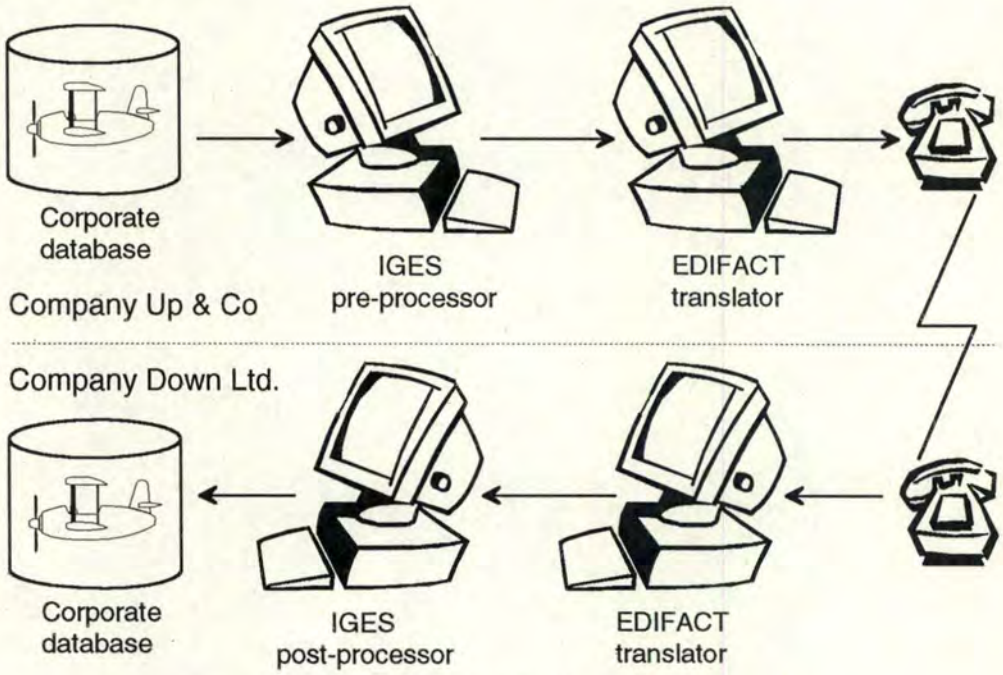


Figure 1: A transfer example

The inclusion of a binary stream in an EDIFACT interchange is not really a though problem but it deserves some examinations. We defer a complete treatment of the latter topic to the next chapter. For the remainder of this chapter, we assume it is possible to incorporate transparently a binary stream within an EDI interchange.

Is this enough for the two companies to exchange data successfully? Hardly not.

As we said earlier, the problem is only half way solved. What is missing is a way for *Up & Co.* to say to *Down Ltd.* «This is the most up-to-date IGES model of our advanced™ plane, it replaces the one you received last year.» I.e. a mechanism to describe what is in the black box so that the EDIFACT translator of Down Ltd. knows to which software it must pass the data. Also it allows Down Ltd. to link the data to previously received data. Shortly what is needed is some sort of packing standard.

As you might guess, CALS has such a standard, MIL-STD-1840B (the B means that it is currently in its third revision) which we are now to review. Our study of 1840B (as it is affectionately known) will differ from that of the other CALS standards.

SGML, CGM, Group 4, IGES and STEP all manipulate information independent from the EDI transmission and doom to remain so. Their study was illustrative and we tried not to be technical. SGML, CGM, etc. files will undergo the EDI interchange as sealed binary streams which are never open by the EDI translator.

1840B on the contrary deals with just the sort of information one likes to see in plain vanilla EDI format. For example, the EDI translator will deal with a notion of data type («this is an IGES file») to know which program to call upon for actual processing of the binary stream. We will therefore shift to a more technical point of view. Also an entire section of this chapter is devoted to mapping the concepts introduced by 1840B to EDI interchanges.

2. 1840B Overview

Notice that 1840B is not based on an international standard as there is no international standard with equivalent functionalities.

For historical reasons 1840B is defined in term of nine-track tapes as the default exchange medium. Nine-track tapes were a common exchange medium when the 1840 standard was first drafted. This does not mean that 1840B precludes the use of other media. On the contrary, significant efforts have been made, in the B revision, to abstract the aspects of nine-track tape so that the standard is suitable for any electronic medium.

ANSI tape is simply one of the possible media type. In fact, ANSI tape is one of the only non-proprietary electronic medium available and is the only one explicitly referred to in the standard.

Basically 1840B defines two things:

- a naming convention;
- a header block scheme;

which provides the following functionalities:

- organisation of the documents;
- referencing;
- transport of information used by decoders.

Before we get into the hows and whys, let us define the all important term of file. For 1840B, a file is

> *A digital repository of organized information consisting of records, items or arrays, and data elements. [1]*

This definition is very general. It certainly applies to what is commonly known a file (i.e. a data unit on a disk or a tape) but it is more general than that. Therefore we can use 1840B with other medium than disks or tapes, in particular EDI interchange.

2.1. Transfer Units

With 1840B, the data to be exchanged is organised in transfer units. A transfer unit is the smallest collection of files necessary to make a successful interchange. The standard defines five types of transfer units:

- page image transfer unit;
- Page Description Language (PDL†) transfer unit;
- SGML document transfer unit;
- product data transfer unit;
- miscellaneous transfer unit.

There is only one sort of data in a transfer unit. Therefore various documents related to a single product may be split across two or more transfer units if they are presented in different formats. For example the exchange of both the user manual (in SGML) and the model (in IGES) of *Up & Co.*'s advanced™ plane must be split in two transfer units. The only exception is if the model illustrates an SGML document but CGM is the preferred format for illustration.

For best use of high capacity medium, transfer units are grouped in transfer sets which are themselves grouped in transfer packages. They can be up to 34,695

---

† Recall from our SGML study that a PDL is a language to describe complete, formatted images of pages to output devices. 'Virtual paper' is a good metaphor for PDLs. The most famous PDL is PostScript® from Adobe Systems Incorporated. Another one, common on PCs, is PCL developed by Hewlett-Packard for its laser printer range.

transfer unit in a single transfer set. The maximum number of transfer sets grouped in a single transfer package is left to mutual agreement between sender and recipient.

As an illustration, let us consider two typical transfer units: the most complex transfer unit, the SGML document transfer unit and a simpler one, the product data transfer unit. An SGML document transfer unit consists of:

- a mandatory transfer unit declaration file;
- a Document Type Declaration (DTD) file (one file per transfer unit; this file is optional unless otherwise specified by previous agreement between the sender and the receiver);
- a SGML coded text source file (one file per transfer unit, mandatory);
- SGML text entity files (one file for each text entity referenced in the transfer unit, mandatory). Entities with PUBLIC identifiers need not to be transmitted with the transfer unit when the receiver has access to a copy of the entity;
- illustration files in IGES format, CGM format or raster format, as specified by contract or other form of agreement. Entities with PUBLIC identifiers need not to be transmitted with the transfer unit when the receiver has access to a copy of the entity;
- a Formatting Output Specification Instance (FOSI) data file (one file per transfer unit, optional). Again FOSI with a PUBLIC identifier need not to be transmitted with the transfer unit when the receiver has access to a copy of the FOSI;
- special word files as specified by previous agreement between sender and receiver;
- contract defined data files as specified by previous agreement.

This is the most complex transfer unit because it is made of many files, most of them being optional. The product data transfer unit is simpler and shall consist of the following (again most files are optional):

- a mandatory transfer unit declaration file;
- engineering drawing data files in IGES or raster format as specified by previous agreement;
- electrical/electronic application data files;
- numerical control manufacturing data files.

2.2. Naming Convention

2.2.1. Numerals

Throughout its naming convention 1840B uses *numerals*. A numeral is an identifier made of numeric characters and upper case letters. The set of all possible numerals is ordered to form an ascending sequence.

The identifier set extends from '001' to '999' and from there to 'ZZZ'. I.e. the first identifiers are the numbers from '001' to '999' written as character strings. When the numbers from '001' to '999' have been exhausted, the upper case letters 'A' to 'Z' shall be used to extend the set of identifiers as follow†:

```
001...999,
A00...A09, A0A...A0Z, A10...A1Z, ..., AZ0...AZZ,
B00...B09, B0A...B0Z, B10...B1Z, ..., BZ0...BZZ,
.
.
Z00...Z09, Z0A...Z0Z, Z10...Z1Z, ..., ZZ0...ZZZ
```

† Once '999' is reached, the set is extended as if it was in base 36 using the alphabet '0'...'9' 'A'...'Z'.

It probably seems a bit difficult to the reader. Why using letters only when '999' is reached? For historical reasons, of course. In its initial release, the standard allowed for only 999 identifiers using the numbers from '001' to '999'. Since new high capacity medium (like optical disks) can transfer more than 999 typical transfer units, the standard was adapted but managed to retain ascending compatibility with the old naming scheme — in an admittedly contrived way. This new scheme can build up to 34,695 identifiers.

2.2.2. Naming Convention

Each transfer unit contains exactly one transfer unit declaration file. The transfer unit declaration file name is made of the letter 'D' followed by a numeral. The numeral for the first transfer unit declaration file in the transfer set is '001', it is incremented for subsequent transfer unit in the same transfer set. The name is not an identifier of the transfer unit merely a mechanism to differentiate transfer unit within a given transfer set. The transfer unit is identified by data in the transfer unit declaration file (Refer to the following section).

In addition to a transfer unit declaration file, a transfer unit contains at least one data file. The name of which is eight characters long and is made of the transfer unit declaration file name followed by:

- a single letter indicating the data type. Appendix A lists all admissible code letters and the associated data type;
- a numeral used to differentiate files of the same data type in a single transfer unit. The numerals are used in ascending sequence starting with '001' for every new data type in the transfer unit.

Figure 2 is an example, taken from David Peterson [2] of the filenames from a typical transfer unit. The second column consists of comments only and is not part of the filenames.

```
D001        the Transfer Unit Declaration
D001T001    the SGML Document Entity
D001N001    an SGML Text Entity (perhaps the DTD)
D001N002    another Text Entity (perhaps Chapter 1)
.
.
.
D001N007    the last of 7 Text Entities
D001C001    a CGM graphic
D002C002    another CGM graphic
.
.
.
D001C150    the last of 150 CGM graphics
D001R001    a CCITT raster graphic
.
.
.
D001R099    the last of 99 raster graphics
D001H001    the FOSI
```

Figure 2: A typical transfer unit

This naming convention provides with three things:

- it indicates to which transfer unit in a single transfer set a file belongs. Differentiating transfer sets in a transfer package is left to mutual agreement between sender and receiver;
- it provides an indication of the type of each file;
- it differentiates between files of the same type in a single transfer set.

It is plain from this naming convention that the receiver can regroup files in transfer unit based solely on their names. How files are organised on a particular medium is definitely media-type dependent. Of course some medium offer other methods to group files than the name, 1840B nevertheless requires adherence to its naming convention.

2.3. Transfer Unit Declaration File

The transfer unit declaration file provides information to identify the transfer unit and other information used by the receiver to decode data in the transfer unit.

Every transfer unit contains a mandatory transfer unit declaration file. It uniquely identifies the transfer unit.

2.3.1. Transfer Unit Declaration File Content

Transfer unit declaration file data is organised in fixed size records of 128 bytes each. Record must be padded with blank characters (' ') until they reach the fixed size. The data is to be written in ASCII. For example the number 128 is represented as the string '128'.

Each record starts with an identifier followed by a semicolon and exactly one space as separator. The rest (including additional spaces) is part of the record value. If the value of a record exists but is not provided (whether it is known or unknown), the string 'EMPTY' must be used instead. If the value does not exist either the string 'NONE', if the value is alphanumeric, or the string '0' (the number zero), if the value is numerical, is to be used.

The allowed records are (please notice that this classification is by theme; appendix B lists the records in the order imposed by the standard):

- identifiers:
  - srcsys: any information as specified by previous agreement to identify the system from which the transfer unit originated;
  - dstsys: any information as specified by previous agreement to identify the destination system to which the transfer unit is going;
  - srcdocid: document identifier on the source system (e.g. reference number, file identifier, etc.);
  - dstdocid: document identifier on the destination system (e.g. reference number, file identifier, etc.);
  - srcrelid: identifier on the source system of another document to which this document is closely related (e.g. this document is a supplement to another document);
  - dstrelid: identifier on the destination system of another document to which this document is closely related (e.g. this document is a supplement to another document);
  - chglvl: change/revision and date of the document or data product;
  - dteisu: date this document was issued;
  - dtetrn: date of transfer;

- doctyp: identifier on the source machine of the document type (e.g. supplement, job guide, schematic diagram, work card, assembly drawing, etc.);
- transacttyp: the transfer unit type (e.g. SGML, PDL, etc.);
- dlvacc: free form record giving information as specified by previous agreement, such as contract number, CDRL item, etc.;

■ miscellaneous
- version: version and revision of the standard used for the transfer, MIL-STD-1840B constrains this string to: 'MIL-STD-1840B, 0, 19921103';
- filcnt: the numbers of each type of data file in the transfer unit;

■ security
- ttlcls: the security label security/sensitivity level or other restrictions on the title of the document;
- doccls: the security label highest security/sensitivity level or other restrictions on any file in the transfer unit;
- docttl: document title (e.g., a technical publication or engineering drawing title).

Figure 3 is an example of a declaration file unit transfer.

```
version: MIL-STD-1840B, 0, 19921103
srcsys: AJAX Inc. 100 Doe St. San Diego, CA  92110
srcdocid: Fire control system ver 14
srcrelid: F-18 avionics system ver 12
chglvl: REVISION W/CHG, G, 2, 19911201:1209:03
dteisu: 19890801/1200:00
dstsys: ABC System,  Wright-Patterson AFB, OH, 45433
dstdocid: 4SA6-11-4
dstrelid: 4SA6-11
dtetrn: 19920710:0900:31
dlvacc: CDRL item 6 of Contract N33400-93-C-1052, Due 19950731
filcnt: T8, Q4, C1, R1
ttlcls: Unclass
doccls: Unclass
doctyp: System schematic
docttl: F-18 fire control system
transacttyp: SGML
```

Figure 3: Example of a declaration file unit transfer

2.4. Data Files

Each data file starts with a header block (the size of which depends on the data file type) which contains information related to the data. It is not part of the data self and must be stripped from the file by the receiver.

The convention for record formatting is identical to the one used for transfer unit declaration file. In particular a semi-colon and a space separate the record name from its value and the strings 'EMPTY', 'NONE' or '0' must be used when no value is provided.

Some records are relevant with some data type only. Appendix C lists them all.

Some of the most significant records are:

- specversion: which identifies the version of the standard used to encode the data;
- srcdocid: and dstdocid: with which we are familiar from the declaration file;

- datfilid: which is an identifier of the file content;
- doccls: which encodes security information and is particularly relevant in defence environment.

Record usage varies according to the data file type and not all records are required for every file. I.e. some records are specific to some file types, some are mandatory, others are conditional.

Some records receive a value fixed by the formatting standard. A typical example is the dtype: record in raster files. We overlooked it but MIL-R-28002 defines two types of file organisation (1 and 2) and the dtype: record indicates which sort of file is at hand.

3. Learning from 1840B

To summarise 1840B provides the two partners of the exchange with:

- grouping and organising of the information in physical and logical entities (files, transfer units, transfer sets and transfer packages). Somehow this is similar to segments, messages and interchanges in EDI standards;
- a referencing scheme (srcdocid:, dstdocid:) to uniquely identify data transfer and link them;
- additional information supplied in accordance with encoding standards to help the receiver in decoding the data (dtype:).

The first functionality is provided by the naming convention while the other two are supported by the records.

These three points are all the functionalities a packing standard must provide but the mechanisms may differ from those adopted by 1840B. Indeed to provide similar functionalities with an EDI standard, there are two alternatives:

- declare the EDI standard as a medium type and map 1840B mechanisms to the EDI standard, for example:
  - the transfer unit becomes an interchange;
  - the file becomes a message;
  - new segments are defined to hold the various records of 1840B.

  We stress that this mapping is just illustrative and other equally acceptable mappings are possible. X.12 (the ANSI EDI standard) has such a mapping in the form of its transaction set 841 which is dedicated to the exchange of Specifications/Technical Information;
- use other mechanisms (to support the same functionalities) which suit better an EDI standard than the rather generic 1840B approach.

In conclusion to enable EDIFACT for the exchange of technical documents, we must enhance it with mechanisms to support the three following functionalities:

- organisation of the documents;
- referencing;
- transport of additional information used by decoders.

EDIFACT adds another problem to this list: the inclusion of a binary stream in an interchange. EDIFACT uses a limited subset of ISO 646 where some bit patterns have reserved meanings. This has lead to tense debates within the EDIFACT standardisation committees.

In the next chapter we will survey various techniques to satisfy those four requirements.■

4. References

[1] Department of Defense
MIL-STD-1840B Automated Interchange of Technical Information
Department of Defense, US, November 1992

[2] David Peterson
MIL-STD-1840B: Beyond Nine-Track Tape
CALS Journal, Winter 1993, p.63-65

Appendices

Appendix A: Data File Name Code Letters and File Format

Notice that some files might be in format we have not studied in this document. This is because 1840B supports file formats which are not part of CALS but are nevertheless used by the DoD. This is partly due to the fact that 1840B existed before other CALS standards.

Code Letter	*Data File Type*
A	contract defined data file
C	CGM
E	EDIF
G	SGML DTD (file contains no text)
H	FOSI
I	IPC
N	SGML text entity
P	PDL
Q	IGES
R	raster graphic
T	SGML coded text
V	VHDL
X	special word file
Z	fray scale/color data file

Appendix B: Transfer Unit Declaration File Records

This appendix lists all the records found in a transfer unit declaration file. Records descriptions are to be found in the text of the chapter, this appendix gives the order as imposed by the standard.

version:

srcsys:

srcdocid:

srcrelid:

chglvl:

dteisu:

dstsys:

dstdocid:

dstrelid:

dtetrn:

dlvacc:

filcnt:

ttlcls:

doccls:

doctyp:

docttl:

transacttyp:

Appendix C: Data File Header Records

This appendix lists all the records to use in a data file header with a short description of their meaning.

specversion:	a character string identifying the specification the data file is in accordance with Example: specversion: MIL-R-28002 19881220
srcdocid:	a character string used by the source system to uniquely identify the document to which this file belongs. Normally identical to the source system document identifier (srcdocid:) of the transfer unit declaration file
dstdocid:	identical to destination system identifier document identifier (dstdocid:) of the transfer unit declaration
datfilid:	a contract specified description of the content and processing of this file
moduleid:	PUBLIC identifier by which this module is known in the SGML document
dtype:	type or scope of the data contained in the file (Cf. chapter text for an example of its use)
rorient:	raster image orientation, as specified by MIL-R-28002
rpelcnt:	raster image pel count, as specified by MIL-R-28002
rdensty:	raster image density, as specified by MIL-R-28002
didid:	data item description identification number
doccls:	a string stating the label security/sensitivity level of the data file
fosipubid:	PUBLIC identifier of an associated FOSI
notes:	free form text

7

Technical EDIFACT

Which innovation leads to a successful design and which to a failure is not completely predictable. Each opportunity to design something new, either bridge or airplane or skyscraper, presents the engineer with choices that may appear countless. The engineer may decide to copy as many seemingly good features as he can from existing designs that have successfully withstood the forces of man and nature, but he may also decide to improve upon those aspects of prior designs that appear to be wanting.

Henry Petrosky

In the previous chapter we assumed it was possible to integrate a binary stream, taken as a black box, in an EDIFACT interchange. We said it was a simplification because nothing exists yet in the standard to support this function. We are now going to address this problem and see how it can be solved.

EDIFACT syntax has two major drawbacks for the inclusion of a binary stream:

- it uses a limited subset of ISO 646 (or ISO 6937 or ISO 8859† depending on partners agreement) compatible with telex lines;
- it assigns special usage to certain characters.

Today three solutions are considered by the standardisation body. We are going to review them all. Each of these solutions addresses the two problems listed above. The three solutions are:

- segment approach;
- envelope approach;
- transmission protocol approach.

† Briefly ISO 6937 and ISO 8859 are 8-bit versions of ISO 646. Of course, being an 8-bit character set, they encode more characters than ISO 646.

1. Syntactic Drawbacks

1.1. Background on Binary Vs Text

Up to this point we have talked of binary and text information as two completely different things. It was a gross simplification since they are really two different beasts of the same species. The simplification was fine in previous chapters and helped to keep things simple. Unfortunately, we can no longer be satisfied with simplifications, the next section requires some understanding of the relationship between text and binary data. This sub-section intends to remind the reader with some facts on computer representation. The reader familiar with this aspect will probably want to skip this sub-section altogether.

In a computer everything is represented as a run of numbers encoded in base 2. So are characters, numbers, programs, images. If we look at information in a computer memory (be it in central memory or on disks) what we see is a stream of bits, i.e. a stream of 0 and 1. Since this stream is composed only of 0 and 1 (i.e. using a two symbols alphabet or *binary* alphabet) it is often called a binary stream. How we assign meaning to this binary stream is purely conventional.

Such a convention is ISO 646 which associates a 7-bit number to every character in the Latin alphabet. For example, the letter 'A' (capital a) is number 65 which is represented as a stream of bits as 1000001. We see that a text string is actually represented as a binary stream.

For example, take the binary stream 010001100110111101111000. It represents number 4616056 in decimal. Now what does it mean? In itself, nothing. 4616056 may be the number of stock options sold in Wall Street today or the distance, in kilometres, a fly accomplishes in its live. We don't know. The binary stream may also be a raster image coded, for example, in accordance with CCITT Group 4 in which case the number 4616056 is meaningless.

To attribute a meaning to the stream 010001100110111101111000, we must know which encoding scheme was used to produce it. If we are told it is ISO 8859, we can decode it into the string 'Fox'.

So what about the relationship between text and binary stream? Internally a computer represents a text as a binary stream.

Now why do we (and other authors) differentiate between texts and binary streams? Because, prior to the arrival of multimedia, text was the most common way to store data. The word *text* is traditionally used in the literature to qualify binary streams that must be interpreted as texts as opposed to the expression *binary data* which designates binary streams that are not intended to be interpreted as text.

Of course, every binary stream has a text correspondent but if the binary stream does not actually encode a text, it is likely to decode in a meaningless one. Say we encode the number 16426, for any reason. It will be stored, in a computer, as the binary stream 0100000000101010. If, by mistake, it is taken for a text, it will decode into the meaningless string '@*'.

To summarise, the word text is used for bit pattern which must be interpreted as a run of characters while binary is used in every other case.

1.2. Limited Subset of ISO 646

To be able to include transparently any binary stream in EDIFACT interchanges, one must be able to write any bit combination in interchange bodies.

Unfortunately, the EDIFACT syntax rules preclude this. EDIFACT interchanges must be coded in a subset of ISO 646 which, for example excludes character '@' (represented by 1000000 in ISO 646). This means that the bit stream 1000000 is illegal

within an EDIFACT interchange, it can appear nowhere in the interchange. Most combinations of bits are prohibited to appear in the interchange by the current syntax rules. Unfortunately, binary data can, by definition, be made of any bit pattern. It is therefore impossible to include arbitrary binary data *as is* with current syntax rules.

There is a consensus in the EDIFACT community to amend the syntax rules if binary streams are to be incorporated in interchanges.

Truly there is another solution than to change the syntax rules but it has its costs. The alternate solution is to pre-process and post-process the binary stream to transform it into another stream which does not include any illegal bit pattern and is used for the interchange solely.

The EDIFACT translator on the sender system processes the binary data (s) with the help of a function f( ) and outputs a binary stream freed from illegal bit pattern (s'). It is this special binary stream (s') which is used for the exchange:

$s'=f(s)$

The intermediate binary stream (s') is somehow similar to the character encoding of CGM as it is composed only of values with a character equivalent. Moreover the function ensures that only those characters which are accepted by the syntax rules are used. On the receiver system, the EDIFACT translator processes s' with the help of f( ) inverse function (f^{-1}( )) to recover s:

$s=f^{-1}(s')$

Many functions can be used for f( ). The most trivial (and highly inefficient) one simply replaces a byte with its two characters long hexadecimal representation (base 16). For example, 1000001 is replaced by the string '41'. This simple function doubles the stream size. Other functions will inflate the stream by a smaller margin by making use of all the characters (i.e. bit patterns) authorised by the syntax rules.

The reader may wonder if these pre-processings and post-processings are realistic. Somehow yes because the idea is not new and has been used with other systems.

For example, PostScript level 1 can only handle binary data (such as raster images) if they are mapped to a character representation.

Also Internet users have been doing this for years with their uuencoders and uudecoders. These are simple programs which transform any binary stream onto another one with a character equivalent, the only form of data accepted by Internet mailers.

Nevertheless, there are two major drawbacks to this technique:

- it increases document size, wasting network resources. This is a major problem for technical documents which are already large by nature;
- it forces pre-processing and post-processing of binary data by the EDIFACT translator, wasting processor resource. Once again it is annoying with large technical documents;
- it makes poor usage of the most recent communications protocols, all of which offer mechanisms to transport efficiently binary data. Telex lines are definitely a heritage from the past.

Once again, there is a consensus in the EDIFACT community to amend the syntax rules.

1.3. Special Characters

EDIFACT assigns a special usage to four characters, by default '+', ':', '?', ' ' ' (the apostrophe). Of course they can only appear at special places in interchange. This creates some problems when they are part of the data (such as an apostrophe in an address). The well known escape character ('?') overcomes this limitation.

Similarly, these characters, or strictly speaking the equivalent of their coding in ISO 646, may appear in a binary stream. It is possible to extend the escape character mechanism but it is not felt desirable. Rather another technique is used to overcome this problem in binary exchange.

1.3.1. Escape Character

Extending the escape character technique to binary stream has one major drawback: it requires pre-processing in the sender system and post-processing in the receiving system.

This is highly undesirable due to the potentially huge size of binary streams. Inserting and removing escape sequences in a small amount of data (a typical EDIFACT message is quite small compared to a typical drawing) incurs little penalty performance.

With EDIFACT messages, the penalty is reduced to an absolute minimum since the escape sequences are inserted and removed while encoding or decoding the data. The translator has to process the message anyway so it needs no to be done in a separate treatment. But the binary data are not created and processed by EDIFACT translators so inserting escape sequences requires separate processing. The performance penalty is high.

Notice that this problem could be avoided if the binary stream is pre-processed with the technique we introduced in section 1.2. Although we saw it is unlikely that this solution will ever be adopted.

1.3.2. Counter

The alternate solution is to prefix the binary stream with its size so that the translator knows it has to pass a given number of bytes before resuming normal processing. Until it has reached the end of the binary stream, the translator not longer attributes special meanings to any character.

This solution is most satisfactory in term of speed (the translator simply copy a given number of bytes, it did not check the bytes). Its drawback is that it forces the translator to switch to another method of work for the binary data only. It is a less elegant but more efficient technique.

Some people fear it would mean throwing away every current EDIFACT translator. We don't think so for at least two reasons:

- X.12 (the US EDI standard) has adopted this approach and X.12 translators adapted well;
- it is not considered a big issue by developers as we learned through personal communications with Interbridge developers†. Changes are minimal, according to them.

Similarly another common objection to this method states that the translator has to process the interchange to know whether it contains a binary data or not and can be forced to switch to a special processing method at any time. Again this is not considered problem by developers but, in any case, it suffices to define a new character set which will inform the translator that a binary data is part of the interchange.

† Interbridge is Sitpro's EDIFACT translator.

2. Inclusion of a Binary Stream in an Interchange

In the following sub-sections, we will introduce three figures to illustrate our views. The key for these figures is in figure 1, please notice that rectangle sizes, in the figures, are irrelevant:

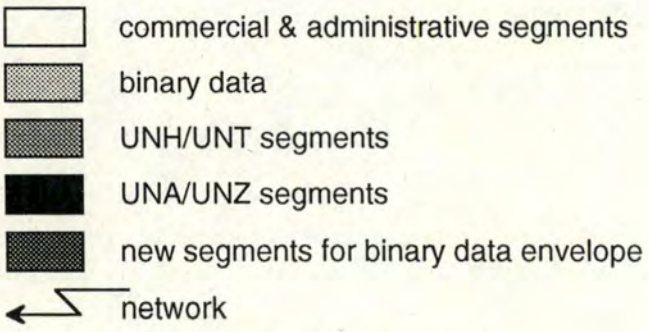


Figure 1: Key of figures

2.1. Binary Segment

This solution implies creating a new segment to hold the binary stream (suggested label is BIN). This segment can be very simple: it suffices to prefix the binary stream with its size, as we saw previously.

The binary segment is then integrated in a message with other (non binary) segments — typically reference segments. Either a specific message is designed for technical data or general purpose ones (like the invoice) includes binary segments.

An instance of the message (which starts and ends with UNH/UNT segments since it is an ordinary message in every respect) can be included in an interchange, possibly with other messages some of them may also contain binary segments. For message developers, the binary segment is in every way similar to other segments. In particular, it can be conditional, mandatory and can be mixed freely with any other segments.

Figure 2 illustrates this solution.

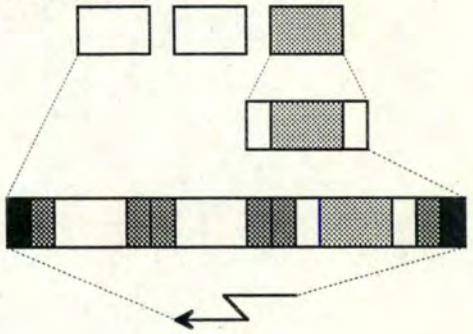


Figure 2: Binary segment

There are three common objections to this approach:

- it mixes data of different nature which, conceptually, should be at different levels and indeed are processed by different programs. This makes it difficult to use communication protocol facilities for the exchange when they are available;
- if the BIN segment is not part of a dedicated messages, it links binary information with other information based solely on locality (there is no need for an explicit reference) which is not considered the state-of-the-art in referencing scheme;
- it requires amending the syntax.

It has one major advantage though. A relatively small amount of binary data (like a small illustration) can be mixed with other data (such as price and part number) with minimal overhead. It is considered that in the case of catalogues where illustrations are numerous and small, the overhead of explicit referencing, as required by the two other approaches, might be significant.

It may be worth noting that X.12 uses this technique in its transaction set 841.

2.2. Transparent Envelope

This solution requires the creation of a new entity, at the level of the message, specifically designed to hold binary streams. The new entity would be called a transparent envelope because it *transparently* incorporates binary data. Practically, it means the creation of two new service segments, UND (transparent data header) and UNE (transparent data trailer) to envelop the binary data.

A new entity is then incorporated in an interchange. Potentially along with other transparent envelopes or messages. The new service segments must provide at least a referencing mechanism which links the binary data to messages. Explicit references are used. Notice that the binary data is not part of any message.

Refer to figure 3 for an illustration of this solution.

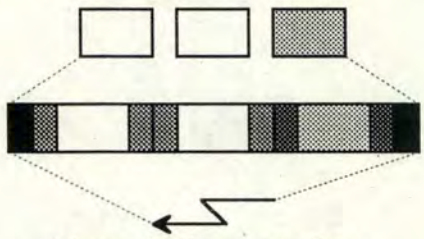


Figure 3: Binary envelope

Once again this requires amending the syntax rules. But all the other problems of the binary segment are solved with this approach:

- it separates data from different natures (EDI formatted and non-EDI formatted) which makes it easy to switch to communication protocol facilities when available. Indeed it is very easy to separate the binary data from the interchange since no message must be modify;
- it uses explicit referencing.

This approach duplicates the body parts functionality found in some recent communication protocols such as X.400 and Mime (to be introduced shortly).

It may be worth noting that CII (Japanese national EDI standard) uses a similar technique.

2.3. Transmission Protocol Reliance

The last technique relies on communication protocols to handle binary exchange. The interchange and the binary information are passed to the communication protocol as two separated but linked entities. A reference mechanism is used to link them.

Some recent communication protocols support the transfer of multi-parts documents. I.e. a single document consists of multiple entities (bodies in X.400 terminology), potentially of different types, which are considered as a whole by the communication protocol [1].

The most famous of these protocols is X.400, an E-mail standard developed by the CCITT. Internet mail has been extended with support for multiple bodies documents by the Mime protocol.

The major drawback of this approach is that it terminates EDIFACT independence from communication protocols. Its main advantage is that it does not require amending the syntax rules.

Figure 4 illustrates this solution.

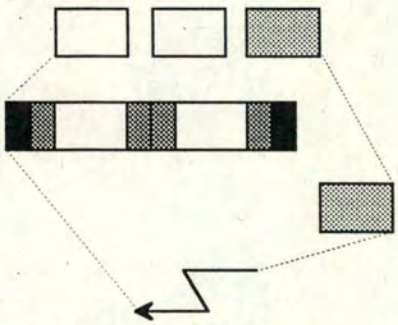


Figure 4: Communication protocol

3. Linking Binary with EDI Data

In chapter 6 we saw that other information need to be exchanged along with the binary data. In chapter 6 we said this information is best exchanged in native EDI format. In this section, we will see how EDIFACT current facilities support this. Keep in mind that additional information is required whichever method is chosen for the inclusion of binary data:

- grouping and organising of the information: this already exists in EDIFACT in the form of messages and interchanges;
- reference scheme to link data from separate transmissions. Techniques exist in EDIFACT like the RFF (reference) segment;
- additional information to help decoding: segments like the EFI (electronic format information) or, in a more limited way, FTX (free text) can be used for this function. New segments may be added to the standard.

Anyway new segments may be added to the standard but there is no need to change the syntax rules in order to convey this information. The only change in the syntax rule, if ever, will take place for the transparent transport of binary stream.

The US standard (X.12) has a special segment to include the 1840B records This segment is used solely by transaction set 841 which is dedicated to the exchange of Specifications/Technical Information.■

4. References

[1] Stanislas van Oost
The Use of CCITT X.400 Recommendations for EDI
FUNDP, Belgium, 1991

8 Views

It is a good aphorism of software design that the unexpected wouldn't be so named if it wasn't so hard to predict.

Steve Rimmer

In this chapter we present some personal views on the problems we discussed so far.

1. Links

We have almost completed our study of the inclusion of technical documents in EDIFACT. In the last chapter we introduced three methods to link EDIFACT interchanges with binary data. These approaches are currently discussed in standardisation bodies.

Our opinion is that the best approach is the second one (the transparent envelope). We consider it to be the best compromise for two reasons:

- it retains EDIFACT independence from communication protocols;
- it clearly separates EDIFACT formatted information from other information.

1.1. The Independent

The EDIFACT standard has been designed to be independent of system, machine and media constraint. To delegate the handling of technical documents to the communication protocol clearly marks the end of this independence. It is particularly worrisome as not all communication protocols support multiple bodies documents.

We consider it is vital for EDIFACT to integrate technical document exchange today for most of its user. It requires a solution which is independent of communication protocols as multiple parts protocols are not yet widely adopted.

There is a gap between present needs and common technology which the transparent envelope solution fills.

One may fear that this approach uselessly duplicates communication protocol features in the EDIFACT standard. Duplication is inevitable: to retain independence from communication protocols EDIFACT has to assume a minimalist protocol and build on top of it. In fact EDIFACT already duplicates many communication protocol features. The most typical example is provided by the interchange sender and interchange recipient composite data elements of the UNB segments: most if not all communication protocols already provide this information.

In this particular case however, duplication is minimal. There is no need to support advanced features in the transparent envelope, just the minimum to transport arbitrary binary stream.

1.2. Estranged

The main advantage of clearly separating commercial and administrative EDI data from technical information is that it allows to easily revert back to communication protocol usage when it is desirable. There is no dependence since communication protocol can be selected only when there is a strong advantage of using it.

This point may appear to oppose with the previous one. Which it does not. In the previous point we were concerned with the acceptance of the feature in today's world. In this sub-section we are concerned with the future of EDIFACT.

It is clear that the future calls for multiple parts communication protocols which will be perfect for the exchange of technical documents hand in hand with commercial and administrative ones. The X.400 protocol and the new Mime protocol, a multimedia extension of the popular Internet E-mail protocol, are both multiple parts protocols. This trend is likely to expand in the future.

Nevertheless, today those protocols are not commonly used. The market response to X.400 has been one of cold expectation and Mime has not yet reach complete acceptance.

Therefore we think it is vital, for the future of EDIFACT, to provide a solution which is both acceptable today (i.e. is part of the interchange) but is ready to evolve to rip the benefits from better communication facilities when made available.

Since the transparent envelope approach clearly separates what is in EDIFACT format from what is in another format in the interchange, it makes it easy to separate the two parts and pass them as separate entities to the communication system.

In short we think the present development of EDIFACT should not preclude its future. The transparent envelope mechanism is flexible enough not to preclude future evolution.

2. Last Reflections

In this document we have limited ourselves to a technical analysis because we felt it was important to have a clear and complete technical corpus on the subject. Also space and time were short and somehow we had to neglect other aspects. In the last section of this chapter we want to share some reflections on those issues.

2.1. Politics

As it is always the case when people from all around the world must agree on a common way of doing things, the matter is a highly politic issue too.

X.12, the American standard, has adopted a binary segment approach to include technical documents in EDI interchange. When the solution was adopted, it was introduced as temporary solution while other efforts were on their ways. It turns out that the temporary solution has become permanent and today Americans are reluctant to abandon it in favour of another, albeit compatible, one.

Similarly people from the X.400 business push towards a communication protocol solution. While some users want the EDIFACT solution to be free from X.400, or other communication protocol.

The final solution will probably be influenced by those non-technical but nevertheless important issues.

2.2. Technical Standards

We think it is desirable that EDIFACT standardises on a default set of standards for unstructured documents, to provide some guidance for EDIFACT users. Due to the potentially wide use of unstructured documents, it is dubious that any international committee can make a choice that would please every user. Nevertheless a default set would help newcomers to unstructured document exchange.

In this work we limited ourselves to CALS standards. Many others exist which support similar functionalities. Some are best adapted to a particular usage than their CALS equivalent. It In this section we will briefly examine other ISO standards and commonly used *de facto* standards.

We will only consider compound documents and graphics because today these are the most common unstructured documents. But the situation evolves and our choice is likely to widen to other medium like sounds and videos in the near future. At the time of writing we are unaware of any ISO standard in those areas.

2.2.1. Compound Documents

The three ISO standards for compound documents are SGML, ODA and SPDL (a standard PDL based on PostScript®). We think SGML is the best choice for a default set because it is simple, powerful, flexible and widely implemented. Furthermore it can be extended to hyperdocuments.

De facto standards are the Rich Text Format (RTF) developed by Microsoft and commonly used word processor file formats like *WordPerfect* and *Microsoft Word*.

2.2.2. Graphics Standards

The choice is eased by the fact that there is only one ISO standard: CGM.

Popular *de facto* standards include TIFF (Tagged Image File Format), CompuServe GIF (Graphic Interchange Format), Windows Bitmap and Targa files. All of these are raster formats only. EPSF (Encapsulated PostScript File) is another file format popular for vector and raster graphics.

For product model, we saw that the ISO standard (STEP) is still under development. Other standards include IGES, SET, EDIF, etc. A popular *de facto* standard is DXF (Drawing Exchange Format) designed by Autodesk for its best-selling AutoCAD system.

2.3. Management

In the second chapter, we dedicated a section to managerial issues. To dedicate only a single section to such an important aspect is certainly not enough. The reader should not be fooled by this short treatment however: the issue is managerial before being technical. Linking computers together by networks is one thing. Publishing standards to allow effective data exchange between computers is another, tougher, problem. But to have various departments work hand in hand can be the real challenge and it can be accomplished only through top managerial commitment.

Unfortunately we are not very competent for managerial issues and we must direct our reader to other publications, like V. Daniel Hunt's *Enterprise Integration Sourcebook* [1] or CALS' own guidance manual (MIL-HDBK-59B). Another valuable reading for this and many other issues is the *CALS Journal* we introduced in chapter 3.

2.4. Security

It is clear that the exchange of technical documents of strategic importance between companies should be attempted only in a secure environment.

Two issues are particularly worth considering:

- taping;
- juridical protection.

Security issues are not specific to technical EDI and this is the reason why we did not dedicate a chapter to the subject. Nevertheless technical EDI users must be aware that, for some exchanges, security is paramount. Of course not all information requires high level security. Publicly available information, like catalogues, requires none or little security only.

2.4.1. Taping

Design documents exchanged amongst companies can be of vital importance for the competitiveness and survival of some or all of them. This is the case if two or more companies collaborate on a new design. If they achieve a new product that gives them a competitive edge over the concurrence, it is vital to ensure the secret is well kept.

It requires security measures both in the companies self and during the exchange which must be protected from taping by unauthorised third party. The issue is not specific to technical EDI but the importance of some interchanges makes it particularly relevant with technical EDI.

2.4.2. Juridical Protection

In life threatening application being able to trace design choices can become a matter of survival for a company. Once again this issue is not specific to technical EDI, in fact it has nothing to do with EDI but its importance means it should not be overlooked when preparing technical EDI exchanges.

When a company designs or manufactures a product, it engages its responsibility. If the product turns out to be faulty and causes an accident, the company survival (both due to legal compensation and loss in notoriety) may depend on whether the company is able to prove it met security requirements or to trace to a subcontractor who failed to deliver appropriate quality. Being able to trace the design, and to record design decisions and requirements can turn out to be a very important issue, perhaps the most important issue of the whole exchange. The records must be reliable enough to stood the test of time for the responsibility of the company can be questioned only years after it has delivered the product. Similarly the capacities of a company to meet its contractually imposed requirements can be questioned before a product actually reveals a fault.

Keith Blacker from Lucas Engineering and Systems Ltd expressed the opinion that this aspect was the most important one in technical EDI†.

We think this issue, mainly a juridical one, should permeate every technical choice and exchange agreement contracts.■

3. Reference

[1] V. Daniel Hunt
Enterprise Integration Sourcebook
The Integration of CALS, CE, TQM, PDES, RAMP, and CIM
Academic Press, San Diego (US), 1991

† Personal communications.

9 Conclusion

I can call spirits from the vasty deep.
Why so can I, or so can any man; but will they come when you do call for them?

Shakespeare

People communicate all day long. For pleasure and for business. Apart hermits, few humans are actually willing to avoid communication. Communication is particularly important for the success of most businesses.

> *Most companies must interact with other companies, and divisions within the same company must interact with each other. Although it is in principle easier to impose a single computing structure on all divisions within a company, in practice it often turns out that intracompany computer-mediated communication is nearly as difficult as intercompany communication, which is a major problem in business today. Indeed, while the computers of one company can usually exchange strings of letters and numbers with the computers of another company, fax rather than computer networking is the standard means of interchange for documents involving images, notes, and formatted text. [1]*

How come that faxes supplanted computer networks for business communications? The situation is even more surprising when one consider that most faxes actually come from computers. It is such that most modems currently on the market are actually modem/fax devices and turn PCs into fax machines. These systems are such ingenious that to send faxes, one only has to print onto the fax peripheral.

Sure E-mail is rapidly becoming a common way to conduct business and EDI acceptance is growing. But these are mostly (if not exclusively) used for text communications. On those days where communication highways gain such attention, one cannot help to notice that without a change in our business habits, electronic highways will be under-used.

Ian McEwan from General Motors summarises the situation when he says:

> *Data exchange is one of the fundamental building blocks on which industry and commerce are based. It is also one of the least understood of our basic requirements. Everybody understands how to send a letter or make a phone call; very few know how to send data. [2]*

and, concerning electronic highways:

> *However, we will all find, as GM found, that the highway itself, while essential, is not sufficient. We also need the open international protocols and standards that will allow us to use it. [2]*

1. Conclusion

We choose to end our study of technical EDI with some considerations on electronic highways because these highways seem to bring Starck's dream (a world where only ideas move) a little closer. Alas anybody who has ever tried to transfer a non trivial piece of data between two computers, even close to each other, knows it is not enough.

If the physical link is a prerequisite to any exchange between computers, it is hardly enough. Lest the computers exchange amorphous and meaningless bit streams, there is also a need for common convention on information representation. At present the only truly universal convention is ASCII, but ASCII carries only a low level semantic. To gain extra levels more expressive conventions are required. In this document we reviewed some of them for key aspects of documents interchange. We were mainly concerned with technical documents exchange in the context of EDIFACT but most of the techniques we reviewed can be successfully adapted to other contexts.

After an introduction on structured versus unstructured documents (chapter 1), we examined technical documents and concluded there was a need to exchange them with EDI (chapter 2). We then look at currently used techniques for technical document exchanges (chapters 4, 5, 6). In the process we reviewed the popular CALS initiative (chapter 3). We terminated our study with some suggestions on the inclusion of those techniques in EDIFACT (chapters 7, 8).

We hope you have enjoyed the reading.■

2. References

[1] Juris Hartmanis and Herbert Lin (Eds.)
Computing The Future
National Academy Press, Washington, DC, 1992

[2] Ian McEwan
Product Data Exchange in a Global Manufacturing Economy
CALS Journal, Winter 1993, p. 25-27

Appendices

Common Abbreviations and Acronyms

2D: 2 Dimensional
3D: 3 Dimensional
AECMA: Association Européenne des Constructeurs de Matériel Aérospatial
ANSI: American National Standards Institute
APLS: Advanced Procurement and Logistic Systems
ASCII: American Standard Code for Information Interchange
CAD: Computer-Aided Design
CAE: Computer-Aided Engineering
CALS: Continuous Acquisition and Life-cycle Support
CAM: Computer-Aided Manufacturing
CAx: stands for all kind of Computer-Aided systems (CAD, CAM, CAE,...)
CCITT: Consultative Committee on Internatioanl Telegraphy & Telephony
CE: Concurrent Engineering
CIM: Computer Integrated Manufacturing
CGM: Computer Graphics Metafile
CIM: Computer Integrated Manufacturing
CSG: Constructive Solid Geometry
DB/KB: database/knowledgebase
DIS: Draft International Standard
DNC: Direct Numerical Control
DoD: US Department of Defense
DSSSL: Document Style Semantics and Specification Language
DTD: Document Type Definition
DXF: Drawing Exchange Format
EDI: Electronic Data Interchange
EDIF: Electronic Design Interchange Format
EDIFACT: EDI For Administration, Commerce and Transport
EPSF: Encapsulated PostScript File
FOSI: Formatting Output Specification Instance
GIF: Graphic Interchange Format
GOSIP: Government Open Systems Interconnection Profile
IPC: Institute for Interconnection and Packaging Electronic Circuits
ISWDB: Integrated Weapon System Data Base
IT: Information Technologies
HyTime: Hypermedia/Time-Based Structuring Language
IGES: Initial Graphic Exchange Specification
ILS: Integrated Logistic Support
ISO: International Organisation for Standardisation
JIT: Just In Time
LAN: Local Area Network

MAP: Manufacturing Automation Protocol
NC: Numerical Control
ODA: Office Document Architecture
OSI: Open Systems Interconnection reference model
PDES: Product Data Exchange Using STEP
PDL: Page Description Language
RTF: Rich Text Format
SDAI: STEP Dara Access Interface
SET: Standard d'Echange et de Transfert
SGML: Standard Generalised Markup Language
SPDL: Standard Page Description Language
SQL: Standard Query Language for database
STEP: Standard for Exchange of Product Data
TI: Technical Information
TIFF: Tagged Interchange File Format
TOP: Technical and Office Protocols
VHDL: VHSIC Hardware Description Language
VHSIC: Very High Scale Integrated Circuit

Collected References

Dr. Donald P. D'Amato and Rex C. Klopfenstein
A Study of the CALS Standards for the Interchange of Patent Documents
CALS Journal, Summer 1993, p. 28-35

D. B. Arnold, P. R. Bono
CGM and CGI
Springer-Verlag, Germany, 1988

D. B. Arnold, D. A. Duce
ISO Standards for Computer Graphics
Butterworths, UK, 1990

Peter R. Bono, José L. Encanação, L. Miguel Encarnação, Wolfgang R. Herzner
PC Graphics with GKS
Prentice-Hall, UK, 1990

Department of Defense
MIL-STD-1840B Automated Interchange of Technical Information
Department of Defense, US, November 1992

Design Museum
Is Starck a Designer?
Design Museum, London, 1993

R. Doty
Product Data Sharing Using STEP Technologies
Digital Equipment Corporation, US, 1992

J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes
Computer Graphics — Principles and Practice, Second Edition
Addison-Wesley, Reading (Massachussets), 1990

Joseph J. Fuller, Theodore J. Post & Anne S. Mavor
Test and Evaluation of the Navy Technical Information Presentation System (NTIPS), F-14A Field Test Results, DTRC-88-036
US, September 1988

Joseph J. Fuller & Samuel C. Rainey
The Interactive Electronic Technical Manuals
CALS Journal, Winter 1992, p. 63-69

Frank Gilbane
Integrating New Technologies: Workflow Systems
CALS Journal, Summer 1993, p. 65

Charles F. Goldfarb
A Generalized Approach to Document Markup
ACM Sigplan Notices, Volume 16, Number 6, June 1981, p. 68-73

Charles F. Goldfarb
The SGML Handbook
Clarendon Press, UK, 1990

Charles F. Goldfarb
HyTime: A Standard for Structured Hypermedia Interchange
IEEE Computer, August 1991, p. 81-84

Joseph R. Goss and James Brunke
Rockwell Implements CALS
CALS Journal, Winter 1993, p. 42-45

Juris Hartmanis and Herbert Lin (Eds.)
Computing The Future
National Academy Press, Washington, DC, 1992

James H. Harvey
SGML Applied to Automotive Service Information
CALS Journal, Fall 1993, p. 27-31

Lofton R. Henderson, Anne M. Mumford
The CGM Handbook
Academic Press, London (UK), 1993

V. Daniel Hunt
Enterprise Integration Sourcebook
The Integration of CALS, CE, TQM, PDES, RAMP, and CIM
Academic Press, San Diego (US), 1991

Bryan Martin
SGML an Author's Guide
Addison-Wesley, UK, 1988

Ian McEwan
Product Data Exchange in a Global Manufacturing Economy
CALS Journal, Winter 1993, p. 25-27

Chris McMahon, Jimmie Browne
CADCAM — From Principles to Practice
Addison-Wesley, Wokingham (UK), 1993

A. J. Medland, Piers Burnett
CADCAM in Practice — A Manager's Guide to Understanding and Using CADCAM
Kogan Page, UK, 1986

Christian Monteix
Le Format TIFF et ses Modes de Compression
Eyrolles, France, 1993

Office of the Defense CALS Executive
CALS Definition and Vision Statement
September 21, 1993
in CALS Journal, Winter 1993, p. 11

David Peterson
MIL-STD-1840B: Beyond Nine-Track Tape
CALS Journal, Winter 1993, p.63-65

Yuri Rubinsky
Technical documentation in the World of STEP
CALS Journal, Spring 1993, p. 63-66

SITPRO & PFA
The UN/EDIFACT Workshop - Class Notes
SITPRO & PFA, UK, 1993

Joan M. Smith and Robert Stutely
SGML: The User's Guide to ISO 8879
Ellis Horwood, UK, 1988

Joan M. Smith
An Introduction to CALS:
The Strategy and the Standards
Technology Appraisals, UK, 1990

Joan M. Smith
SGML and Related Standards
Ellis Horwood, UK, 1992

J. Soelberg - W. Sorokine
Pratiquez l'électronique en 15 leçons
Editions Radio, Paris (France), 1986

Technology Appraisals
Open Information Interchange
Technology Appraisals Ltd, UK, 1993

Amjad Umar
Distributed Computing, a Practical Synthesis
Prentice Hall, New Jersey (US), 1993

Eric van Herwijnen
Practical SGML
Kluwer Academic Publishers, The Netherlands, 1990

Stanislas van Oost
The Use of CCITT X.400 Recommendations for EDI
FUNDP, Namur (Belgium), 1991