## Tutorials

These tutorials will hopefully give you an overview of some of the interactive TV standards that are being deployed today.

These tutorials assume that you either know something about digital broadcasting already, you've read my (or someone else's) introductory page, or that you're hoping things will get easier as they go along and that you don't really need to understand all these silly acronyms.

In the latter case, good luck - I've tried to keep these pages fairly acronym-free, but that's not always possible.

From this point on, I'll only be describing the highlights of the specification. After all, the MHP specification is over one thousand pages long, and any tutorial can only cover so much of that. The full specification is available for free [download](http://www.interactivetvweb.org/tutorials) from the [ETSI web site](http://pda.etsi.org/pda/home.asp?wki_id=12127.). Many other specifications relating to DVB services can also be downloaded from there.

The [JavaTV specification](http://java.sun.com/products/javatv/) is similarly available from the Sun web site Reading the specs themselves is the only way to grasp all of the subtleties of the specifications, and at the end of the day you'll need to do this anyway if you're going to be developing software using one of these standards.

If you're fairly familiar with the specifications, but want to learn how to use a specific feature, take a look at the [code samples](http://www.interactivetvweb.org/resources/code_samples). These commented examples will show you how to do some of the most common tasks that an MHP [application](http://www.interactivetvweb.org/tutorials) needs to do.

Please choose which [tutorial](http://www.interactivetvweb.org/tutorials) to view:

* [Digital TV Introduction](http://www.interactivetvweb.org/tutorials/dtv_intro)
* [Getting Started](http://www.interactivetvweb.org/tutorials/getting_started)
* [The MHP Tutorial](http://www.interactivetvweb.org/tutorials/mhp)
* [The OCAP Tutorial](http://www.interactivetvweb.org/tutorials/ocap)
* [The JavaTV Tutorial](http://www.interactivetvweb.org/tutorials/javatv)

## The JavaTV Tutorial

<http://www.interactivetvweb.org/tutorials/dtv_intro/broadcast_engineering_basics>

Sun have been working in digital TV standardisation for several years. Several open standards such as MHEG-6 and the DAVIC standard had used [Java](http://www.interactivetvweb.org/tutorials/javatv) in digital TV middleware, but none of these standards had used Java as the primary application model - Java was always treated as a way of adding more advanced scripting functionality to declarative technologies.

This had led to a complicated and clumsy [application](http://www.interactivetvweb.org/tutorials/javatv) model, and so Sun felt that it was time to build a pure Java solution. The result of this was the JavaTV specification. Since DVB was working towards the same end, there was a lot of cross-talk between Sun and DVB, and many companies were involved in both specifications. MHP uses the application model from JavaTV as a central piece of the MHP specification, and all of the JavaTV APIs are included in MHP.

Given this overlap, why have a separate JavaTV specification at all? Well, MHP is specific to the DVB family of standards. JavaTV, on the other hand, is not. JavaTV describes a set of digital TV concepts such as accessing service information, selecting a new service, and loading files from a carousel rather than a normal filesystem, but it does so in a way that's not tied to any digital TV standard. It's possible to implement JavaTV on an ATSC system, on a DVB system or on an ARIB system - in fact, this is now happening with the spread of open standards such as MHP, OCAP, ACAP and ARIB B23.

While JavaTV describes many of the core concepts that are needed, it is not a [complete](http://www.interactivetvweb.org/tutorials/javatv) specification of a digital TV platform. There are elements missing that must be defined before JavaTV can be used in the real world (e.g. JavaTV says nothing about what underlying Java platform is in use), and so JavaTV is used more as a component of other standards than as a middleware platform in its own right. For this reason, it may appear that learning JavaTV development is less useful than learning OCAP or MHP development.

Since it forms a common core for other standards, though, learning to develop JavaTV applications can be very valuable when it comes to improving portability: using the JavaTV APIs can help reduce the number of places where your application must be modified, and can help you to avoid making any platform-specific assumptions about your code.

This tutorial is designed for people who already have a knowledge of Java and of the basics of digital TV. If you're not sure that your digital TV knowledge is suitable for this tutorial, please take a few moments to review the [introduction to digital TV](http://www.interactivetvweb.org/tutorials/dtv_intro) elsewhere on this site.

**A Developer's Guide To Digital TV**

This [tutorial](http://www.interactivetvweb.org/tutorials/dtv_intro) is aimed at those of you who are moving to the interactive  
TV world, either from a background in PC software development, or as a  
manager who needs to know more about this funny [DTV](http://www.interactivetvweb.org/tutorials/dtv_intro) stuff.

This tutorial can be split into two main sections: the first section  
covers the basics of [digital TV](http://www.interactivetvweb.org/tutorials/dtv_intro) systems and how DTV signals are transmitted,  
while the second section takes a detailed look at the DSM-CC standard that  
is used by most digital TV systems for broadcasting data streams.

There's a lot of technical stuff in the next few pages, but we'll try  
and keep that to a minimum unless we explicitly have to. Some parts of  
this are more technical than others: only people who really need to know  
the gory details of DSM-CC should try to read past the introduction to  
DSM-CC, for instance.

At the moment, some sections are fairly specific to the DVB system of  
digital TV broadcasting. To stop anyone dealing with OCAP from feeling  
left out, however, we do have a tutorial on ATSC service information under  
development.

## A little More About DVB And MHP

The [Digital Video Broadcasting](http://www.dvb.org/) consortium is an industry consortium that standardizes various aspects of digital TV broadcasting. In the past, they've standardized issues such as how digital TV signals are transmitted over cable, satellite or terrestrial broadcasting networks, how information describing a digital TV transmission is encoded in the bitstream, and generally how to make the various parts of a digital TV system work together.

They've been doing this for several years now, and DVB standards are pretty widely used across Europe and Asia, with some take-up in the US. The DVB web site's page on [the use of DVB standards worldwide](http://www.dvb.org/about_dvb/dvb_worldwide/) gives you an idea of how many countries are now using DVB standards. A couple of years ago, they [started](http://www.interactivetvweb.org/tutorials/dtv_intro/dvb_background) to get the ball rolling on an open standard for interactive digital TV.

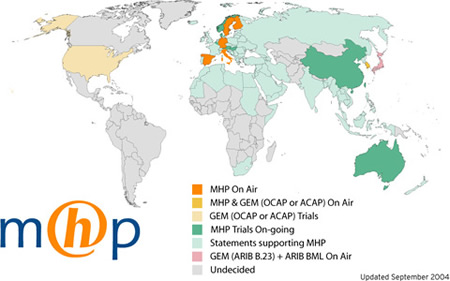
There have been attempts to do this before, by ISO with the [MHEG](http://www.km.giti.waseda.ac.jp/WG12/) standards, by [DAVIC](http://www.davic.org/), and by [ATSC](http://www.atsc.org) with the [DASE](http://www.itl.nist.gov/div895/cmr/dase/) standard. Unfortunately these were not very successful for a variety of reasons, but DVB's attempt seems to have been timed just about right. Previous standards had either been finalized just at the start of the [Java](http://www.interactivetvweb.org/tutorials/dtv_intro/dvb_background) wave, or as attempts to extend and adapt older standards to include Java. These timing issues meant that their chances of success were greatly reduced, but the world wasn't really ready for an open standard at that time.

These older open standards are still used in some cases (e.g. one of the UK digital broadcasters uses MHEG), but companies like OpenTV, NDS, and Microsoft (who have proprietary iTV solutions) still dominate the iTV market at the moment.

MHP is an attempt to change this. In it's simplest form, it's a set of Java APIs that let you write interoperable applications. These are broadcast as part of the MPEG-2 stream that makes up a digital TV signal, and an MHP-compliant receiver can run these applications on your TV. Since MHP uses Java (and some API extensions that are specific to digital TV), you can do almost anything you want to.

People have written MHP games, information services and electronic program guides, as well as news tickers, stock tickers and services related to TV shows such as sports broadcasts. I don't have the rights to show any of these applications here, but the [example applications page on the MHP site](http://www.mhp.org/content_creation/examp_app.html) shows a small selection of what's out there. There are lots of others that aren't shown there, but that's a pretty good selection. MHP is starting to become more widely adopted, at least by governments and public terrestrial broadcasters. It's still got a long way to go in the area of commercial broadcasting (e.g. cable and satellite operators), but it's making a few inroads. The [map](http://www.interactivetvweb.org/tutorials/dtv_intro/dvb_background) below (taken from the MHP web site) shows MHP adoption worldwide:

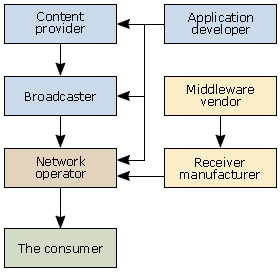
Adoption of MHP worldwide. Source: the [DVB web site](http://www.dvb.org/graphics/internal/WAM-DVB-MHP.png).

[](http://www.interactivetvweb.org/images/tutorials/dtv_intro/mhp_adoption_map.jpg)

OK, so you there's an open standard out there that's starting to compete with the existing proprietary solutions. So what? What does MHP have to offer over the current solutions?

Before we answer this, lets take a look at the content delivery chain for a digital TV signal. This particular diagram illustrates a situation that's fairly common at the moment:

A vertical (closed) market for digital television.



This a vertical market - the broadcasters rent the set-top boxes to the consumer as part of the subscription deal, which means that they control the specification for the receivers. So, all the receiver manufacturers can do is build boxes to the specification of the broadcaster, and all the consumer can do is take the box that the broadcaster provides to them.

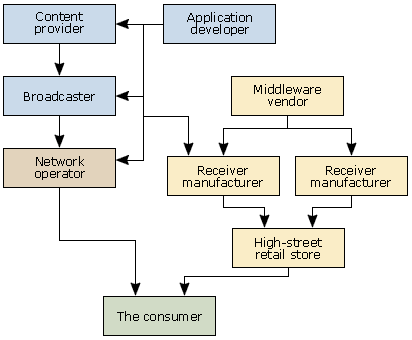
Since the broadcasters control the receiver specification, they also control the applications that will run on those receivers by their choice of middleware. Since different broadcasters can choose different middleware systems, a content provider may have to re-author any applications if they want to sell that content to other broadcasters.

So, in the current DTV markets, the broadcasters are pretty much full control , and everyone else has to go along with their decisions. Which is good for the broadcasters, right? Yes and no. The broadcasters get the control, but because they own the receivers, that can mean that a lot of capital is invested in hardware and the associated costs (maintaining inventory, technical support, upgrades and replacements, etc.)

For this reason, this situation isn't necessarily that desirable for the broadcasters. Ideally, they would like to dump these kinds of costs on to someone else. So the people who actually have the least risk in this situation is the middleware supplier, since there aren't many proprietary middleware vendors and no-one really wants to develop a new middleware stack on their own. The cost of competing with companies like NDS, OpenTV, and Canal+ is so high that very few companies can afford to do it, especially since existing middleware vendors have pre-existing relationships with broadcasters, STB vendors, CA companies and other players in the DTV business.

Now that we've seen the vertical market, let's look at the picture of how an open standard would work in an ideal  
world:

A horizontal (open) market for digital television.



In this case, the consumer can buy a receiver from any electronics store and expect it to work with any digital TV broadcaster, just like the current situation with analogue broadcasting. The subscription to the broadcaster simply gets the consumer a decryption module that allows you to watch the broadcasts. The user now no longer requires a separate set-top box for every broadcaster that he or she wishes to subscribe to, so it's possible to subscribe to different broadcasters for different packages in a much easier way than is currently possible.

Since the broadcaster no longer controls the specification of the receivers, receiver manufacturers are in a better position to add value and compete with each other, driving improvements for the customer. Similarly, since the middleware is now standardized and open, content developers can distribute their content to more broadcasters without having to re-author all their applications.

Middleware vendors are the people who suffer most in this picture, because the barrier to entry is now much lower and it's easier for new companies to implement middleware and compete with existing vendors. This drives down the cost of the middleware, but it is also likely to dramatically increase the size of the overall market as broadcasters who can't use proprietary middleware solutions (e.g. public broadcasters) start to deploy digital services.

Ideal worlds are nothing like the real world, of course, and so this situation isn't one that's likely to happen any time soon. The more likely scenario is a mixture of horizontal and vertical markets where broadcasters use open standards for the middleware. This enables the content developers to sell their content without having to re-author it, and makes life easier for receiver manufacturers, since they have a wider choice of middleware suppliers.

In the long run, this model would probably move closer to our ideal situation, but this will take some time.

So, having learned a little of the politics and reasons behind MHP, hopefully the rest of this site can give you some more information about the MHP standard. This site assumes that you know at least a little about digital TV broadcasting and in particular DVB. If you're unfamiliar with any of the following terms, it's probably best if you take a look at the [introduction to digital broadcasting](http://www.interactivetvweb.org/tutorials/dtv_intro) before exploring the rest of the site:

* DVB service
* Service information
* Section filter
* MPEG-2
* Transport stream
* DSM-CC
* Object carouse

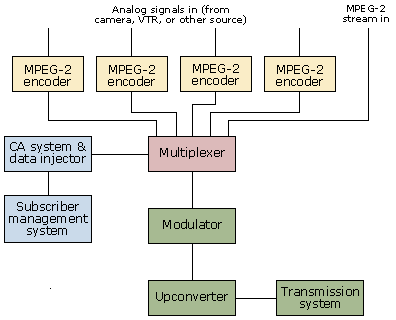
## Broadcast Engineering Basics

The purpose of this section is to provide an introduction to [digital TV](http://www.interactivetvweb.org/tutorials/dtv_intro/broadcast_engineering_basics) transmission issues for those of you who are interested. None of the material in here is really needed for an MHP developer, or even for someone developing MHP middleware. If you want that, take a look at the [introduction to MPEG and digital TV systems](http://www.interactivetvweb.org/tutorials/dtv_intro/dtv_intro).

Instead, this is an introduction to how data gets from the camera or tape to the viewer's screen. This is mainly about broadcast engineering, and the elements of the system that software developers don't usually see. This is by no means [complete](http://www.interactivetvweb.org/tutorials/dtv_intro/broadcast_engineering_basics), and is nothing more than a sketch to show you some of what goes on in the rest of the system.

A typical digital TV transmission setup looks something like this:

The components that make up a typical broadcast system.



This equipment is normally all connected together using high-speed connections like SDI (Serial Digital Interface) or ASI (Asynchronous Serial Interface) which are standard in the TV field. In addition to this, all of the equipment will be connected via ethernet to a control system and monitoring equipment to make sure that nothing goes wrong (or that if something does go wrong, the viewer doesn't see it). There will normally be a large number of some of these components, including some redundant spares in the event of problems. A typical head-end will contain many MPEG encoders and multiplexers, for instance. Now that we've seen how it's put together, let's examine each of these components in more detail.

## The encoder

The encoder is used to take an analog signal and convert it to MPEG-2. This is more commonly used in live shows - for other shows, we may have a selection of pre-encoded MPEG streams that we can [play](http://www.interactivetvweb.org/tutorials/dtv_intro/broadcast_engineering_basics) out from a dedicated playout system. This playout system is usually a highly customized PC or workstation with a large high-speed disk array and a number of digital interfaces for transmitting the data to the rest of the transmission system.

An encoder can generate two types of MPEG stream. Constant bit-rate streams always have the same bit-rate, no matter what the complexity of the scene they contain. If the signal is too complex to be coded at the specified bit-rate, the quality of the encoding will be reduced. If the scene takes less data to code than the specified bit-rate, it will be stuffed with null packets until the correct bit-rate is reached. This makes later parts of the processing easier, because the fact the bit-rate does not change makes things easier to predict later, but it does waste bandwidth.

Most encoders can now produce variable bit-rate MPEG streams as well. In this case, the bit-rate of the stream can be adjusted dynamically, as more or less bandwidth is needed to encode the images with a given picture quality. Since some scenes take significantly more bandwidth to encode than others, this lets the picture quality be maintained throughout a show while the bandwidth changes. The fact that the bit-rate of the stream can change doesn't mean that it will reach higher levels than a constant bit-rate encoding of the same stream of course: the operator can usually set the maximum bit-rate that the encoder can use, and the encoder will reduce the quality of the encoded output, if necessary to meet this.

Most broadcasters today use variable bit-rate encoding because it offers better quality while using lower bandwidth. In particular, variable bit-rate encoding lets us make maximum use of the available bandwidth at the multiplexing stage.

## The multiplexer

One MPEG stream on its own isn't much use to us as a TV broadcast. Even several MPEG streams aren't terribly useful, because we have no way of associating them with each other. What we really need is a single stream containing all the MPEG streams needed for a single service, or ideally multiple services. A transport stream, in other words.

The multiplexer takes one or more MPEG streams and converts them into a single transport stream. The input streams may be individual elementary streams, transport streams or even raw MPEG data - most multiplexers can handle a range of input types.

The multiplexer actually does a number of jobs - multiplexing the data is one of the more complex of these, for a variety of reasons. Each transport stream typically has a fixed bandwidth available to it, which depends on the transmission medium and the way the transmission network is set up. One of the jobs of the multiplexer is to fit a set of services in to this bandwidth. The easy way of doing this is to use constant bit-rate MPEG streams, because then the operator knows exactly how much bandwidth each stream will take, and setting up the multiplexer is easy. This gets pretty inefficient, though, since some streams may be using less than their share of the bandwidth, while others may need to reduce the picture quality in order to fit in their allocated share. This wasted space is a real problem, since the transmission costs are high enough (especially in a satellite environment) that you want to make maximum use of your bandwidth.

The way round this is to use variable bit-rate MPEG streams and a technique known as statistical multiplexing. This system takes advantage of the statistical properties of the multiplexed stream when compared to the properties of the several independent streams. While the bit-rate of each individual stream can vary considerably, these variations are smoothed out when we consider ten or fifteen streams (video plus audio for five to seven services) multiplexed together. Each stream will have different bit-rate needs at each point in time, and these differences will partially cancel one another out at any given time. Some streams will need a higher bit-rate than average at that time, but others will probably need less than average. This makes the bit-rate problems easier to handle, since they are now less severe. By maintaining a separate buffer model for each stream, the multiplexer can decide how to order packets in the most efficient way, while making sure that there are no glitches in any of the services.

At some points, the streams being multiplexed may have a bit-rate that is higher than the available bandwidth. A statistical multiplexer will use another one of the statistical features on MPEG streams to handle this situation. Since most MPEG streams only reach their peak bandwidths at fairly wide intervals for fairly short periods, delaying one or more of the streams will move the peak to a point where the bandwidth is available to accommodate it. This is another reason to maintain a buffer model for each stream - to ensure that these peaks are not moved to a point where they would cause a glitch in the service.

In some older statistical multiplexing systems, the multiplexer and encoders are connected and can communicate with one another. In particular, the multiplexer can provide feedback to the encoders and set the bit-rate that they encode their streams at. The feedback from the multiplexer means that if one stream needs more bandwidth than it's currently getting, the bandwidth for that stream can be increased temporarily at the expense of the others. This doesn't use true variable bit-rate encoding, since in many cases the streams are actually constant bit-rate streams, where the bit-rate used to encode them changes from time to time.

Despite appearances, this system is less flexible than true statistical multiplexing, because if the total bit-rate of the streams is higher than the available bandwidth, then the quality of one of the streams must be reduced. This isn't necessary in the case of the latest generation of statistical multiplexers, where these peaks can often be moved slightly to accommodate them. The other place where flexibility is lost is in the need for a connection between the encoder and the multiplexer. In practical terms, this means that the multiplexer and encoder have to be on the same site, or at least that the encoder feeds only one multiplexer at a time. In these days of remote processing, that can cause problems. Without this need, a network can handle streams where they have no control over the encoder, such as streams from remote sites, from other networks or from a playout system. This offers some big advantages in terms of bandwidth saving.

## Conditional access (CA)

Since we may not want to give our content away for free, we need some way of encrypting our services. This is handled by the conditional access (or CA) system. The algorithm that's used for this is proprietary to each CA vendor, although there are some open (but not publicly-known) algorithms such as the DVB Common Scrambling Algorithm. Manufacturers are understandably nervous about disclosing the algorithms they use, because the costs of having the algorithm cracked are huge - in some European markets, as much as 30% of subscribers were believed to be using hacked smart cards at one point. Even the DVB Common Scrambling Algorithm requires STB manufacturers to sign a non-disclosure agreement before they can use it.

In a DVB system, scrambling can work at either the level of the entire transport stream, or on the level of individual elementary streams. There's no provision for scrambling a service in its own right, but the same affect is achieved by scrambling all of the elementary streams in a service. In the case of scrambled elementary streams, not all of the data is actually scrambled - the packet headers are left unscrambled so that the decoder can work out their contents and handle them correctly. In the case of transport stream scrambling, only the headers of the transport packets are left unencrypted - everything else is scrambled.

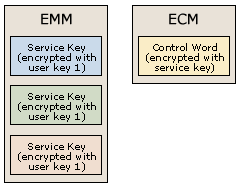
As well as encrypting the data that's supposed to be encrypted, the CA system adds two types of data to the stream. These are known as *CA messages*, and consist of *Entitlement Control Messages (ECM)* and *Entitlement management Messages (EMM)*. Together, these control the ability of individual users (or groups of users) to watch scrambled content. The scrambling (and descrambling) process relies on three pieces of information:

* The *control word*
* The *service key*
* The *user key*

The control word is encrypted using the service key, providing the first level of scrambling. This service key may be common to a group of users, and typically each encrypted service will have one service key. This encrypted control word is broadcast in an ECM approximately once every two seconds, and is what the decoder actually needs to descramble a service.

Next, we have to make sure that authorized users (i.e. those who have paid) can decrypt the control word, but that *only* authorized users can decrypt it. To do this, the service key is itself encrypted using the user key. Each user key is unique to a single user, and so the service key must be encrypted with the user key for each user that is authorized to view the content. Once we've encrypted the service key, it is broadcast as part of an EMM. Since there is a lot more information to be broadcast (the encrypted service key must be broadcast for each user), these are broadcast less frequently - each EMM is broadcast approximately every ten seconds.

Encapsulating code words and service keys in ECMs and EMMs.



One thing to note is that the encryption algorithms used may not be symmetrical. To make things easier to understand we're assuming that the same key is used for encryption and decryption in the case of the service and user keys, but this may not be the case.

When the receiver gets a CA message, it's passed to the CA system. In the case of an EMM, the receiver will check whether the EMM it intended for that receiver (usually by checking the CA serial number or smart card number), and if it is, it will use its copy of the user key to decrypt the service key.

The service key is then used to decrypt any ECMs that are received for that service and recover the control word. Once the receiver has the correct control word, it can use this to initialize the descrambling hardware and actually descramble the content.

While not all CA systems use the same algorithms (and it's impossible to know, because technical details of the CA algorithms aren't made public), they all work in basically the same way. There may be some differences, and the EMMs may or instance be used for other CA-related tasks besides decrypting service keys, such as controlling the pairing of a smart card and an STB so that the smart card will work correctly in that receiver.

In order to generate the EMMs correctly, the CA system needs to know some information about which subscribers are entitled to watch which shows. The Subscriber Management System, or SMS, is used to set which channels (or shows) an individual subscriber can watch. This is typically a large database of all the subscribers that is connected to the billing system and to the CA system, and is used to control the CA system and decide which entitlements should be generated for which users. The SMS and CA system are usually part of the same package from the CA vendor, and are tied together pretty closely.

The ECMs and EMMs are broadcast as part of the service (see the introduction to MPEG if you're unclear on the concept of a service). The PIDs for the CA data are listed in the Conditional Access Table (CAT), and different PIDs can be used for ECMs and EMMs. This makes it easier for remultiplexing, where some of the CA data (the ECMs) may be kept, while other data (the EMMs) may be replaced.

While [NDS](http://www.ndsuk.com) and [Nagravision](http://www.nagra.com) are the two most common CA systems out there, other CA systems are provided by [Conax](http://www.conax.com), [Irdeto Access](http://www.irdetoaccess.com), [Philips](http://www.philips.com) (the CryptoWorks system), and France telecom (the Viaccess system), for example. There are other systems from companies like [Motorola](http://www.motorola.com) and GI who make CA systems, but these are not often used in DVB systems. DVB systems can offer pluggable encryption modules using the DVB common interface (CI), which uses a PCMCIA card to contain the encryption hardware and software. This means that the user can switch encryption systems (for instance, if they change their cable company) without having to replace the entire STB. This is a big advantage for open standards, and really enables the move from a vertical market to a horizontal one.

Some companies (NDS for instance) are not convinced of the security of the DVB CI system, and so not all CA systems are available as CI modules. You have to remember, though, that these interfaces aren't necessary for a system to be DVB-compliant. They're a useful feature, but not required.

ATSC uses a similar, though slightly more secure, mechanism called the POD (Point Of Deployment) module, known as CableCARD in OpenCable systems. These are more widely deployed in US markets, and all OCAP receivers will include a CableCARD slot.

## Error correction and error prevention

Before we can transmit our signal we need to make sure that it will be received correctly. This means some way of identifying and correcting errors in the stream. To do this we add some extra error correction data to the MPEG packets, in order to allow us to correct data. The most common requirement in DTV systems is for an MPEG stream to be quasi-error free (QEF), which means a bit error rate of approximately 1x10-10, or one erroneous bit every 1 hour of video for a 30 Mbits/sec stream. Since we have to be able to correct the errors in real-time, the process is called Forward Error Correction (FEC)

Different transmission mechanisms (cable, satellite or terrestrial) all have different characteristics including different noise levels. A satellite signal for instance can have a lot of errors introduced by conditions in the atmosphere. A terrestrial signal may have errors introduced by reflections from buildings, or by the receiving aerial not being aligned correctly. These different conditions mean that very efficient error correction mechanisms are needed. DVB and ATSC systems all use Reed-Solomon encoding to add a first layer of protection. This adds a number of parity bytes to each packet. Typically, this 16 parity bytes are added to a 188-byte packet, which means that an 8-byte error can be corrected. Larger errors can be detected but not corrected.

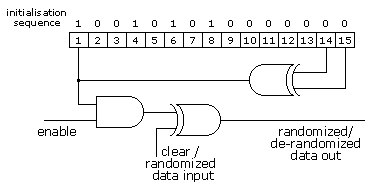
Once this is done, a further layer or error correction coding is added to improve things still further. Common coding mechanisms at this stage are trellis coding and viterbi coding. These exploit the fact that data is not sent one bit at a time, but is instead sent as 'symbols' that can carry several bits of data. In trellis coding, symbols are grouped together to form 'trellises.' For a group of three symbols, a modulation scheme that stores eight bits per symbol can store 512 separate values. By using a subset of these as 'valid' values, the network operator can introduce some extra redundancy into the signal. The effect of this is that each symbol may carry fewer bits of data, but for every group of three symbols, it's possible to correct one erroneous symbol by choosing the value for that symbol that gives a valid trellis. This is the approach used by US digital terrestrial systems. DVB systems use Viterbi coding instead, which is a modification of trellis coding that uses a slightly different algorithm to find the best matching trellis.

To strengthen the error correction, another technique called interleaving may be added. This helps avoid situations where a burst of noise (for example, a lightning strike causing electrical interference) can corrupt data past the point where FEC can fix it. After the data has FEC added, but before it is transmitted, the data is written to a RAM buffer and then read out in a different order. For instance, if we assume that our RAM buffer is a two-dimensional array with ten rows and ten columns, the data may be written to the buffer starting at row 1 and working down to row 10, then read from starting at the top of column 10 and working back to column 1. This means that bytes from the same packet (which will share error correction) are spread over a longer transmission period and are less vulnerable to burst noise.

At the receiver, the process is reversed, and the original order of the bytes can be restored. The interleaving scheme described here isn't the only possible one, and other (more memory-efficient) techniques will often be used instead.

Once we've added error correction, we need to do one more thing before it can be prepared for transmission. If the digital bitstream contains a large run of 1's, then there will be a (small) current flowing in the transmission and reception equipment. This is a Bad Thing, and so some randomization is needed to make sure that there is never a long run of 1's or 0's in the bitstream and to disperse the energy in the signal across all of its bandwidth. To do this, a simple randomizer is used, as shown in the diagram below. The process is symmetrical, so the same hardware is used to de-randomize the signal in the receiver.

A logical diagram of the DVB randomizer.



Every eight transport packets, the randomizer is reset and its register is loaded with the bit sequence 100101010000000. Of course, the randomizer and the de-randomizer must both reset themselves at the same point in the stream, or the input can't be recreated. This is done using the sync bytes from the transport packets. These are not scrambled, so the start of a packet can always be identified, and at every eighth packet, the value of the sync byte is inverted (from 0x47 to 0xB8). This is the signal for the de-randomizer to reset itself, making sure that both the randomizer and the de-randomizer are synchronized correctly.

By doing this, we make sure that the energy is dispersed across the signal spectrum. While it's not strictly necessary, DVB does require that a transmitter and receiver do this before transmitting the signal. The randomizer and its inputs are standardized by DVB in the standards for satellite (EN 300 421), cable (EN 300 429) and terrestrial transmission (EN 300 744). ATSC defines its own randomizer, which can be seen in the [ATSC digital TV specification (ATSC A/53c)](http://www.atsc.org/standards/a_53c_amend-1_corr-1.pdf).

## Modulating the signal

Now we have a digital stream that is almost ready for broadcast. However, we can't directly broadcast digital data - first we have to modulate it - convert it to an analog signal so that we can broadcast it using radio signals or electrical voltages in a cable.

As we've already seen, each of the different transmission mechanisms has different characteristics, and different strengths and limitations. So, each type of signal uses a different modulation scheme. The modulation scheme is just the way of converting digital information into an analog signal so that it can be transmitted. I'm not going to examine these in too much detail, because it's really not interesting to us as MHP developers. The table below describes which modulation scheme is used by each of the transmission mechanisms in a DVB environment.

|  |  |
| --- | --- |
| Modulation schemes in DVB networks | |
| **Transmission mechanism** | **Modulation** |
| Satellite | QPSK |
| Cable | QAM |
| Terrestrial | OFDM |

Cable and satellite use a similar modulation scheme (it's actually the same scheme, with different parameters). The main difference is that satellite signals are more prone to errors and so use a less efficient way of sending the data that provides a bigger difference between symbols, making correct demodulation easier. Terrestrial broadcasts use a different scheme in order to provide a much stronger resistance to errors caused by reflected signals.

The USA uses several different schemes for modulation, as shown in the table below. While the modulation schemes in use are relatively straightforward, the situation for other parts of the system is messy at best. Some satellite providers follow the DVB standards, while other satellite networks and many cable networks use proprietary systems such as DigiCipher II (both cable and satellite) and DSS (satellite only). In all cases they appear to use the same modulation scheme, but details of error correction and other parameters may be different. Cool.stf's [North American MPEG-2 information](http://www.coolstf.com/mpeg/index.html) page has more details about the modulation in DigiCipher II and DSS on satellite systems, as well as details about the use of DVB standards in the US.

|  |  |
| --- | --- |
| Modulation schemes in US digital TV networks | |
| **Transmission mechanism** | **Modulation** |
| Satellite | QPSK |
| Cable | QAM |
| Terrestrial | 8VSB |

The modulation is carried out by a device called, surprise, surprise, a **modulator**. This takes the digital transport stream as an input, and produces an analog output that can be passed onto the transmission equipment. The modulator is the last stage in the process that takes a digital input - after this, everything is analog and we're into the world of radio engineering.

Typically, signals are modulated to a lower frequency than they are broadcast at. Since the broadcast frequencies can be very high (up to 30GHz in the case of satellite transmissions, and up to 950MHz for cable signals), modulating the signals at these frequencies can be hard. So, what happens instead is that the frequencies are modulated at a lower frequency, which is then converted to a higher frequency before transmission. This is done using an upconverter. Basically, this does nothing else except convert the signal from one frequency to another, much higher, frequency. In this case, that other frequency is the one used by the network that you're broadcasting on. Each transport stream will be broadcast on a different frequency, and so the upconverter will have different settings for each transport stream that it handles.

If you want to know more details, [Digital Television -MPEG-1, MPEG-2 and principles of the DVB systems](http://www.amazon.com/exec/obidos/tg/detail/-/0240516958/qid=1094528776/sr=8-5/ref=sr_8_xs_ap_i5_xgl14/102-5225984-4263335?v=glance&s=books&n=507846) by Herve Benoit is a good but expensive read. A cheaper alternative is to read Agilent Application Note 1298, [Digital Modulation in Communication Systems - An Introduction](http://we.home.agilent.com/cgi-bin/bvpub/agilent/reuse/cp_ObservationLogRedirector.jsp?NAV_ID=-536885719.0.00&LANGUAGE_CODE=eng&CONTENT_KEY=1000000348%3aepsg%3aapn&COUNTRY_CODE=US&CONTENT_TYPE=AGILENT_EDITORIAL) which will provide you with all the detail you need.

Once you have a modulated signal, the signal is ready for transmission. All you need then is a transmitter, and antenna (in the case of terrestrial or satellite) or a cable network, and an audience...

## An Introduction To Digital TV Technology

<http://www.interactivetvweb.org/tutorials/dtv_intro/dtv_intro>

This page is for people who need to understand what happens in a digital TV system, but who don't know the details of how a digital TV signal is put together.

This is not an introduction to MPEG, and if you don't know what MPEG is, take a look at [the Tektronix guide to MPEG](http://www.tektronix.com/Measurement/App_Notes/mpegfund/25W_11418_3.pdf). Instead, this tutorial will concentrate on what makes a digital TV signal special, the terminology used to describe a DVB digital TV signal and generally helping you to understand what the hell people talking about digital TV actually mean.

It's also not an introduction to broadcast engineering. Another page describes [the basics of broadcast engineering](http://www.interactivetvweb.org/tutorials/dtv_intro/broadcast_engineering_basics), and shows you how an MPEG stream gets from the encoder to the viewer.

**Disclaimer: some of this information is specific to the DVB standard.** The [map](http://www.interactivetvweb.org/tutorials/dtv_intro/dtv_intro) below (originally from the DVB web site) shows which parts of the world currently use one of the DVB standards - if you're interested in the systems used in the US or Canada, the [ATSC (Advanced Television Systems Committee) web site](http://www.atsc.org/) is a good place to start. The [ATSC service information tutorial](http://www.interactivetvweb.org/tutorials/dtv_intro/atsc_psip/) else where on this site describes the major differences between DVB services and digital services in the USA or Canada, but does not include information about the basic details of the transmission mechanism.

## Anatomy of an MPEG-2 stream

OK, now that we've got that out of the way, lets start talking TV. A digital TV signal is transmitted as a stream of MPEG-2 data known as a transport stream. **Each transport stream has a data rate of up to 40 megabits/second for a** [**cable or satellite**](http://www.interactivetvweb.org/tutorials/dtv_intro/dtv_intro) **network**, which is enough for seven or eight separate [TV channels](http://www.interactivetvweb.org/tutorials/dtv_intro/dtv_intro), or approximately 25 megabits/second for a terrestrial network.

Each transport stream consists of a set of sub-streams (known as elementary streams), where each elementary stream can contain either MPEG-2 encoded audio, MPEG-2 encoded video, or data encapsulated in an MPEG-2 stream. Each of these elementary streams has a 'packet identifier' (usually known as a PID) that acts as a unique identifier for that stream within the transport stream.

**The only restriction on the number of elementary streams in any transport stream is that each elementary stream must have a unique PID value within its containing transport stream.** Since this is stored as a 13-bit value, this is not a major restriction. In practice, the number of elementary streams is limited by the total bitrate of the transport stream. Transmission issues mean that transport streams with bitrates much above 40 megabits/second can't usually be transmitted reliably.

**A transport stream consists of a number of audio and video streams that are multiplexed together**. First, each service in the transport stream will have its audio and video components encoded using MPEG-2 compression. The result of this process is a set of MPEG-2 elementary streams, each containing one video channel or one (mono or stereo) audio track. These streams are simply a continuous set of video frames or audio data, which is not really suitable for multiplexing. Therefore, we split these streams into packets in order to make the multiplexing process easier. The result of this is a packetized elementary stream, or PES.

To create a transport stream, each of these packetized elementary streams is packetized again and the data from the stream is stored in transport packets. Each transport packet has a length of 188 bytes, which is much smaller than a PES packet, and so a single PES packet will be split across several transport packets. **This extra level of packetization allows the stream to support much more powerful error correcting techniques - PES packets are used to provide a way of multiplexing several elementary streams into one bigger stream, and are more concerned with identifying the type of data contained in the packet and the time and which it should be decoded and displayed. Transport packets, on the other hand, are almost purely concerned with providing error correction.**

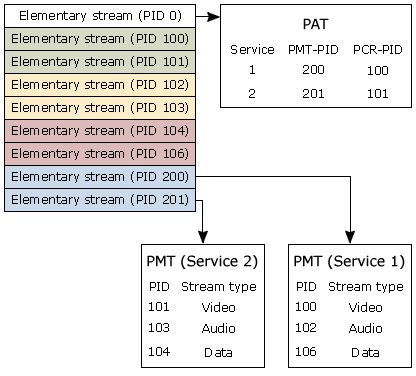
So far, we have just **considered audio and video data**. We may also want to include data streams as part of our service, for applications, **Teletext information or other reasons**. This is not too hard, because MPEG provides a well-defined way of carrying non-AV data inside transport packets. These are called private sections, and we will look at them in more detail little later. Since most of the equipment that generates this data will produce a stream of transport packets containing private sections, multiplexing them in to our transport stream is easy.

Once we have a complete set of transport packets for the different parts of our services, we can insert them into our final transport stream. When doing this, we have to be careful to insert packets in the correct order. This is not just a case of ensuring that all of the packets within the stream come in the right order - MPEG defines a strict buffering model for MPEG decoders, and so we have to take care that each elementary stream in our transport stream is given a data rate that is constant enough to ensure that the receiver can decode that stream smoothly, with no buffer underruns of overruns. We al so have to take care because video streams will use a much larger proportion of the final transport stream than audio streams, and so we can't simply insert a packet from each stream in turn. A ratio of ten video packets to every audio packet is fairly close to what we would likely see.

If we just multiplexed these transport packets together, we would have a transport stream that contains a number of elementary streams with no indication of what type of data is in these streams or how to reconstruct these streams into something that a receiver can present to the user. To solve this problem, MPEG and DVB both specify that other information should be added to the transport stream. This data is encoded in a number of elementary streams that are added to the transport stream during the multiplexing process, and is known as service information.

So what does this service information look like? Basically, it's a fairly simple database that describes the structure of the transport stream. We will examine this in more detail later, but at the simplest level it contains a number of tables that each describe one service in the transport stream. These tables list each stream in the service and give its PID and the type of data contained in the stream.

The anatomy of a DVB transport stream.



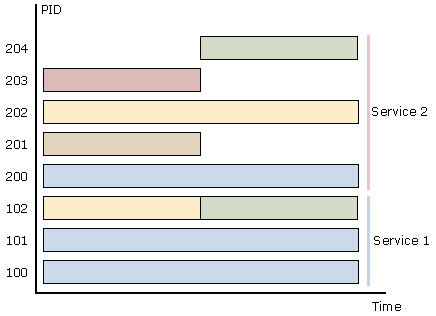
**Information about the types of stream in a service allow the receiver to not only identify which streams are audio and video, but also to identify different types of data stream - separating teletext information from service information from broadcast filesystems for instance. This makes it easy for the receive to know which streams it should pass on to different parts of its software stack for decoding.**

Describing the structure in this way, rather than embedding it into the elementary streams, means that we can re-use elementary streams across services. This is shown in the example [in the next section](http://www.interactivetvweb.org/tutorials/dtv_intro/dtv_intro#rtlmultiplex) of a real transport stream - two elementary streams (the streams with PID values of 32 and 1102) appear in more than one service. This allows efficient re-use of streams across services, and is most commonly used for data streams.

If our transport stream was to contain more than one service, we can simply multiplex all the audio, video and data streams for all the services together. The service information describes which elementary stream belonged to which service, as well as carrying some other information that is more for the benefit of the viewer than the receiver. This may include channel names and descriptions, information about the TV schedules, and parental ratings information.

So, if we take a look at this from a different perspective, we get this picture:

Elementary streams within a transport stream.



In this case, we have a transport stream containing eight elementary streams, split across two services. PIDs 100, 200 and 201 contain video, while the other elementary streams contain audio tracks in different languages. PID 204 contains an MHP application.

As we can see, for both services, the elementary streams containing the video and one audio track continue across services - this is typically done simply to make life easier for the broadcaster and is not required. When there are multiple audio tracks, (or multiple camera angles, as in the case of service 2), then there will be several different elementary streams on other PIDs.

This does not have to happen at an event boundary. As we can see from the case of the MHP application, this application is only available for part of the event. When it is no longer available, the elementary stream that contains it need not be broadcast any more. The ability to update the contents of a transport stream in this way offers a great deal of flexibility to the broadcasters.

A transport stream is different from the type of stream used in DVDs (which is known as a program stream). They are both MPEG-2 streams, and both of them contain multiplexed MPEG-2 audio and video data. However, there are two major differences between them.

The first difference is that a program stream does not contain as much service information as a transport stream. The reason for this is that a program stream has a simpler structure, and can't contain more than one service. Every elementary stream in an MPEG program stream belongs to the same service.

Secondly, transport streams are used in environments where there is much more chance of data corruption. Program streams don't have to worry about this since they are usually stored on optical disks or hard drives. Transport streams, on the other hand, may be transmitted to and from satellites, over terrestrial TV networks or over cable TV networks. This means that they have to be much more resilient, and so transport streams have extra levels of packetization and error-correcting information to help cope with the challenges of the environment that they are used in.

We've skipped a number of points in this discussion of MPEG and transport streams, especially issues such as synchronization and timing. For a more thorough discussion of these and the rest of the MPEG standard, take a look at the [guide to MPEG from Tektronix](http://www.tektronix.com/Measurement/App_Notes/mpegfund/25W_11418_3.pdf).

## Networks, bouquets, services, events & multiplexes

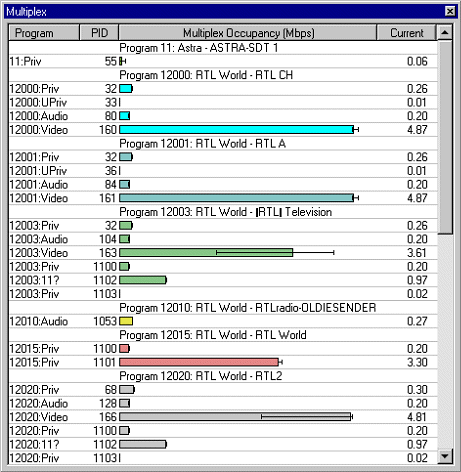
In DTV-speak, **each transport stream is also known as a multiplex**, because it consists of a number of services multiplexed together. Every multiplex is broadcast on a single frequency, and only one multiplex can be broadcast on each frequency. As I said earlier, each multiplex typically has a data rate of around 40 megabits/second for satellite or cable systems, and around 25 megabits/second for terrestrial networks.

Within a multiplex, each group of elementary streams that makes up a single TV channel is called a service. The number of elementary streams in a service doesn't have to stay constant. This can vary between TV shows on that service (for instance, some shows may be broadcast in multiple languages or with multiple camera angles), or it may even change within an TV show. These changes are all perfectly legal in MPEG - not common, but legal.

In digital TV systems, each TV show is known as an event. Thus, from one point of view each service consists of a number of elementary streams that are transmitted simultaneously, but from another point of view the service consists of a series of individual events broadcast one after another.

The image below should give you an idea of what a real transport stream looks like. This is a screen grab taken from a transport stream analyzer, showing one of the multiplexes being broadcast on the Astra satellite:

An example of a real transport stream.



One thing that you will notice is that in this screenshot, services are referred to as programs. This is an MPEG term, and basically means the same thing as a service. One of the reasons an MPEG-2 program stream is so named is the fact that it only contains a single program.

As you can see from this image, the multiplex contains a number of different services, where each service contains at least one audio stream, at least one video stream and usually several data streams. The first column indicates the type of the stream (some can't be identified by the software in this case) - the number that prefixes the type is the number of the service (or program).

The second column shows the PID value for each elementary stream. Although it's not obvious in this screenshot, it's normally good practise to give elementary streams belonging to the same service similar PID values. That way, it's easy to identify which service a given stream belongs to. The final two columns show (graphically and numerically) the bit-rate of each elementary stream in megabits/sec.

As the screenshot shows, digital TV video signals are usually coded at 3-5 megabits/sec for standard-definition video , with the total data rate of a service being 4-6 megabits/sec. This is a little less than DVD-quality, but it does allow the broadcaster to fit a reasonable number of services in each multiplex. Broadcasters like to fit as many services as possible into a multiplex, since this means that they can fit more channels into every frequency band (remember, every multiplex is broadcast on a single frequency band) and so they can use fewer transponders on a satellite to broadcast the same number of services.

OK, so now we know what goes into a transport stream. There are some other things that are worth knowing, however. The transport stream physically groups a set of services together, but services in DVB systems can also be logically grouped as well. A logical group of services is called a bouquet. Why do we need this? Assume for a minute that you work at a large broadcaster, where you broadcast 50 channels. You broadcast over satellite, so each of your transport streams is limited to 40 megabits/sec (around 8 channels). So, you need 7 transport streams to contain all your services.

You sell access to these services in packages, so that a consumer can choose to buy your basic package (which contains 13 channels) sports package (which contains 8 sports channels) or your movie package (which contains 5 movie channels). How can you identify in a machine-readable way which channels are part of which package? You could group them in transport streams, but your basic package is too big to fit in one TS. Instead, by assigning a bouquet to each package, you can group the services into transport streams in the most efficient way while still having a mechanism for grouping the services in a logical way.

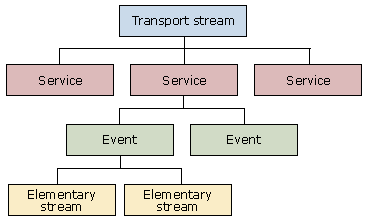
Digital TV systems also have the concept of a network. This is not a computer network: instead, it is a set of transport streams that share some common service information. These transport streams will often be broadcast by the same company (e.g. your satellite or cable operator), and more than one network may be available at any time. This is especially true in terrestrial systems, where there may be several networks operating at the same time in the same area (e.g. several national networks, plus one or more regional operators depending on coverage). In this case, the receiver will normally use automatic channel scanning to find all of the available channels, rather than relying on service information.

The company that owns the network may or may not own the actual delivery medium - in the case of a cable TV system, for instance, the owner of the cable infrastructure is usually also the network operator. In the case of a satellite TV system, however, the network operator (e.g. SES Astra) is typically does not run any of the TV networks that are broadcast over that satellite. Similarly in terrestrial systems, a network (e.g. Freeview) may not own any of the transmitters or distribution mechanism that are used to get its signals to the viewer's home.

So, what we have is:

* A network consists of one or more transport streams that are broadcast by the same entity
* A transport stream is an MPEG-2 stream containing several services
* Each service is a TV channel, and consists of a series of events one after the other
* Each event is a single TV show, and consists of a number of elementary streams
* Each elementary stream is a packetized MPEG-2 stream containing MPEG-2 encoded audio, video or binary data.
* Several services (possibly from several different transport streams) can be grouped together logically in a bouquet.

The structure of a DVB transport stream.



Every service in a DVB network can be uniquely identified by three values. These values are the original network ID (the ID of the network that originally broadcast the service), the transport stream ID (to identify a particular transport stream from that network) and a service ID to identify a service within that transport stream. We can actually go further than this. Each elementary stream in a service may have a component tag, that allows the unique identification of a given elementary stream. This is used by the receivers to decide what service to play, and by interactive applications that want to switch the receiver to decode a different audio or video stream.

DVB transport streams will have both an original network ID and a network ID, which identify the network that originally produced the transport stream (e.g. the BBC) and the one that is currently transmitting it (e.g. BSkyB).

ATSC systems use a combination of the transport stream ID and the source ID to identify a particular service. Depending on the value of the source ID, it may be unique only within the current transport stream or it may be unique at a regional level (or at the network level for satellite signals). This allows some co-ordination of source IDs across different terrestrial networks.

## Service information

So, you have a transport stream, which contains several services, where each service contains several elementary streams. How do we tell what services we're broadcasting? How do we tell which elementary stream belongs to which service? How do we even tell what types of elementary stream we're broadcasting?

As we saw briefly above, the answer is a special set of elementary streams that contain a set of database tables describing the structure of transport stream, the services within it and some useful information that digital TV receivers can show the user, such as the name of the service and schedule information for the services. These tables are collectively known as service information (SI). Every transport stream (DVB or not) has some service information that the MPEG standard declares mandatory, but DVB defines several extra SI tables in addition to the standard ones.

These tables are broadcast as elementary streams within the transport stream. Some of them are tied to specific services within the transport stream, while some are more general and describe either the structure of the transport stream itself or properties of the network. In some cases, elementary streams containing SI are broadcast on a fixed PID to make it easier for decoders to find it, while in other cases the PID on which an SI table is broadcast is stored in another SI table.

[ATSC service information](http://www.interactivetvweb.org/tutorials/dtv_intro/atsc_psip) is covered in a separate tutorial, and so we won't consider it here. The SI tables that are commonly found in a DVB transport stream are:

* Program Association table (PAT) - defined by the MPEG standard
* Program Map Table (PMT) - defined by the MPEG standard
* Network Information Table (NIT)
* Service Description Table (SDT)
* Event Information Table (EIT)
* Conditional Access Table (CAT)
* Bouquet Association Table (BAT)
* Time and Date Table (TDT)
* Time Offset Table (TOT)

The Program Association Table is the fundamental table for service information. It describes which PID contains the Program Map Table for each service (see below) as well as the Network Information Table for the transport stream in those networks that use it.

The Network Information Table describes how transport streams are organized on the current network, and also describes some of the physical properties of the network itself. The NIT also contains the name of the network, and the network ID. This is a value that uniquely identifies the network that is currently broadcasting the transport stream, and may be different from the original network ID that we discussed earlier, if the transport stream is being rebroadcast.

The Conditional Access Table describes the CA systems that are in use in the transport stream, and provides information about how do decode them.

The Program Map Table is the table that actually describes how a service is put together. This table describes all the streams in a service, and tells the receiver which stream contains the MPEG Program Clock Reference for the service. The PMT is not broadcast on a fixed PID, and a transport stream will contain one PMT for each service it contains.

Together, the PAT, PMT, and CAT are known as Program Specific Information (PSI) and are defined by MPEG. All other tables are specific to DVB systems.

The Service Description Table gives more user-oriented information about services in a transport stream. Unlike the PMTs, there is only one SDT in a transport stream, and that contains the information for every service. The SDT typically contains information such as the name of the service, the service ID, the status of the service (e.g. running/not running/starting in a few seconds) and whether the service is scrambled or not.

The Event Information Table provides schedule information about events on a service. This includes the event name, start time duration and the status of the event. This table is actually split in to two separate tables: the EIT-present/following, which contains information about the current and next events, and the EIT-schedule, which contains other schedule information. These in turn can be split into tables describing the current (actual) transport stream and other transport streams. It is mandatory for the transport stream to contain the EIT-present/following for the actual transport stream, while other EIT tables are optional.

The Bouquet Association Table lists and describes the services in a bouquet. This does not provide very detailed information, since this can be gained from other SI tables. Instead, it just provides a list of the services contained in a bouquet.

The Time and Date Table and the Time Offset Table provide a time reference for the stream. The TDT contains the current UTC (Universal/GMT) time, while the TOT contains both this and the offset from UTC for local time. This can be used to calculate schedule information accurately, if needed.

Some of these tables may contain information about other transport streams, as well as information about the current transport streams. The NIT, SDT and EIT must contain information about the current transport stream (and these tables are known as the NIT-actual, SDT-actual and EIT-actual respectively), but the transport stream may also contain versions of these tables that refer to other transport streams. These are known as the NIT-other, SDT-other and EIT-other.

The table below describes which of these tables in mandatory and which is optional.

|  |  |  |  |
| --- | --- | --- | --- |
| Service information tables in a DVB system. | | | |
| **Mandatory (MPEG)** | **Mandatory (DVB)** | **Optional (DVB)** | **Reserved PID** |
| PAT |  |  | 0x0000 |
| PMT (one per service) |  |  |  |
| CAT |  |  | 0x0001 |
|  | NIT-actual | NIT-other | 0x0010 |
|  | SDT-actual | SDT-other | 0x0011 |
|  | EIT-present/following (actual) | EIT-schedule (actual & other)   EIT-present/following (other) | 0x0012 |
|  | TDT |  | 0x0014 |
|  |  | TOT | 0x0014 |
|  |  | BAT | 0x0011 |

Of course, life is not as simple as it appears. Although DVB requires that the mandatory tables are present, it doesn't require that they are populated. That would be too easy. So, it's not entirely unusual to see transport streams containing empty SI tables, either because the broadcaster was too lazy to add the information, or because they are squeezing every last drop of bandwidth from a broadcast. The contents of an SI table may change over time - for instance, the contents of the EIT will change when the current event changes. Each SI table has an associated version number, which is incremented with every change. The receiver will track these version numbers, and load new versions of the tables when they are broadcast.

## Structure of an SI table

Each SI table is fairly similar in structure. They all consist of a header, followed by zero or more descriptor loops. Each descriptor loop contains one or more descriptors that provides the information for one row in the table. Each descriptor may only contain some information for a given row in the table, and some descriptors that contain more generic information may appear in different types of table. This re-use of descriptors makes SI parsers slightly simpler, since each descriptor has its own header (which includes the descriptor ID), as well as the actual information it contains.

A full list of descriptors can be found in the MPEG-2 systems specification, the DVB SI specification, and the ATSC PSIP specification, depending on the system in use, but some of the most common and useful DVB descriptors are:

* The service\_descriptor, which is found in the SDT and gives the name and type of a service.
* The linkage\_descriptor, which is found in several of the tables and provides a reference to a source of more information about an element of the service information, as well as a reference to a replacement service if the current service is not running or is scrambled.
* The component\_descriptor, which is found in the EIT and gives information about an elementary stream such as the content type and format and in some cases (e.g. audio streams) the language of the stream.
* The data\_broadcast\_id\_descriptor is found in the PMT and gives information about the type of encoding used for a data stream.
* The stream\_identifier\_descriptor is also found in the PMT and is used to attach a component tag to an elementary stream so that individual streams may be uniquely identified.
* The CA\_identifier\_descriptor appears in several tables and identifies the scrambling system (if any) that is used for a given service or event

## MPEG Sections

Since these tables can sometimes be quite large, there is a need to split them to fit inside a transport packet. Each chunk of data is knows as a section, and these can be used to hold any type of binary data, not just SI tables.

Sections that contain data and not audio or video streams are typically known as private sections, even when the data format is publicly known. These sections mostly follow a standard format:

|  |  |  |
| --- | --- | --- |
| The MPEG-2 private section format. Source: ISO 13818-1:2000 (MPEG-2 systems specification). | | |
| **Syntax** | **No. of bits** | **Identifier** |
| private\_section() { |  |  |
| table\_id | 8 | uimsbf |
| section\_syntax\_indicator | 1 | bslbf |
| private\_indicator | 1 | bslbf |
| Reserved | 2 | bslbf |
| private\_section\_length | 12 | uimsbf |
| if(section\_syntax\_indicator == '0') { |  |  |
| for(i=0; i<N; i++) { |  |  |
| private\_data\_byte | 8 | uimsbf |
| } |  |  |
| } |  |  |
| else { |  |  |
| table\_id\_extension | 16 | uimsbf |
| Reserved | 2 | bslbf |
| version\_number | 5 | bslbf |
| current\_next\_indicator | 1 | bslbf |
| section\_number | 8 | uimsbf |
| last\_section\_number | 8 | uimsbf |
| for(i=0; i<private\_section\_length-9; i++) { |  |  |
| private\_data\_byte | 8 | uimsbf |
| } |  |  |
| CRC\_32 | 32 | rpchof |
| } |  |  |
| } |  |  |

The table ID is used to identify the contents of the elementary stream (whereas the stream type listed in the PMT only describes the type of stream - the PMT will tell you it's a data stream, but the table ID may help tell you what the data actually is). In the case of SI tables, this identifies which table is being broadcast (as you'd expect) but it is also used for identifying other private data streams.

The table ID extension is used to sub-class this information, where the table ID may identify a given class of data,and the table ID extension may identify a particular data stream within that class. In the case of SI tables, this is used to identify those tables which can contain information about the current transport stream and others. For instance, the table ID for the SDT-actual is different from the table ID for the SDT-other, and in both cases, the table ID extension contains the transport stream ID, identifying which transport stream the SDT refers to.

## Presenting video - decoder format conversion

MPEG-2 video will typically be encoded at PAL or NTSC resolutions (depending on the country of origin). In either case, the encoded video may have an aspect ratio of 4:3, 16:9, or some other aspect ratio, (with the former being more common at the time of writing). This aspect ratio, together with other information, is broadcast in the MPEG stream as part of the data. This data is known as the active format description.

There is no guarantee, however, that a TV used to view the signal will have the same aspect ratio as the encoded video. This is where decoder format conversion comes in. Basically, this is a conversion operation carried out within the receiver to convert the video signal from the original aspect ratio to the aspect ratio of the display. Usually, this is carried out in the analogue stage of the receiver, after the signal has been generated but before it is sent to the TV.

Although the conversion from 16:9 to 4:3 (or vice versa) is most common, there are other common conversion operations to support the different types of conversion such as letterboxing, shoot-and-protect and pillarboxing. The UK Digital Terrestrial Group has published a set of [implementation guidelines in PDF format](http://www.dtg.org.uk/publications/books/afd.pdf) detailing how receivers can handle these conversions. These conversions are also covered in the [EACEM E-Book](http://www.ueberall-tv.de/download/AG_DVBT2/MinAnfor/A1_E-Book2-02.pdf).

[This article](http://www.tvtechnology.com/features/Tech-Corner/f-rh-aspect-ratio.shtml) at TVTechnology.com describes a little more about decoder format conversion and active format descriptions from a US perspective - it's not quite identical to the DVB situation, but it does discuss DVB and the similarities are far more important than the differences.

## DSM-CC

DSM-CC is a standard for data broadcasting (amongst other things) based on MPEG streams. It provides a great deal of functionality that is used in MHP and OCAP - as well as being one of the main methods for transmitting applications to a receiver, it can provide data streaming, timecode information for MPEG streams and a host of other functionality.

DSM-CC is a complex topic, and so it is covered in a separate [DSM-CC tutorial](http://www.interactivetvweb.org/tutorials/dtv_intro/dsmcc). While only middleware developers need to know the gory details of DSM-CC, application developers and head-end manufacturers can also benefit from knowing something about its inner workings.

<http://www.interactivetvweb.org/content/crash-course-designing-tv>

<http://www.interactivetvweb.org/tutorials/dtv_intro/atsc_psip>

<http://www.interactivetvweb.org/tutorials/dtv_intro/dsmcc>

<http://www.interactivetvweb.org/tutorials/mhp>

<http://www.interactivetvweb.org/tutorials/ocap>

|  |  |  |
| --- | --- | --- |
| Java ME and Java Card Technology |  | Java Logo |

Java Platform, Micro Edition (Java ME) provides a robust, flexible environment for applications running on embedded and mobile: microcontrollers, sensors, gateways, mobile phones, personal digital assistants (PDAs), TV set-top boxes, printers and more. Java ME includes flexible user interfaces, robust security, built-in network protocols, and support for networked and offline applications that can be downloaded dynamically. Applications based on Java ME are portable across many devices, yet leverage each device's native capabilities. Red Arrow [Read More](http://www.oracle.com/technetwork/java/javame/about-java-me-395899.html)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| http://www.oracleimg.com/us/assets/spacer.gif |  | http://www.oracleimg.com/us/assets/spacer.gif |

|  |  |  |
| --- | --- | --- |
| [[Phone](http://www.oracle.com/technetwork/java/javame/javamobile/index.html)](http://www.oracle.com/technetwork/java/javame/javamobile/index.html)  **Java for Mobile Devices** | What is Java for Mobile Devices? Java for Mobile Devices relies on Java Platform, Micro Edition (Java ME) to deliver applications and services to all types of mobile handsets, ranging from price-efficient feature phones to the latest smartphones. Java is currently running on over 3 billion phones and offers unrivaled potential for mobile developers worldwide. Red Arrow [Read More](http://www.oracle.com/technetwork/java/javame/javamobile/overview/about/index.html) |  |
| [[Embedded Client](http://www.oracle.com/technetwork/java/javame/embedded/index.html)](http://www.oracle.com/technetwork/java/javame/embedded/index.html)  **Java Embedded** | What is Java for Embedded? Using Java Technology for Embedded enables you to develop highly functional, reliable, portable and secure applications for today's more powerful embedded systems. Oracle offers a full range of products, services, and support that makes it easy for you to develop using Java Technology in your embedded projects.  Red Arrow[Read More](http://www.oracle.com/technetwork/java/javame/embedded/overview/index.html) |  |
| [Java TV](http://www.oracle.com/technetwork/java/javame/javatv/index.html)  **Java TV** | What is Java for TV? Java TV refers to JSR-927, the Java Community Process (JCP) specification providing API's for digital TV-related capabilities for set-top boxes, Blu-ray Disc players, and other digital media devices. Java TV is an optional package which sits atop the Connected Device Configuration, Foundation Profile, and Personal Basis Profile (CDC/FP and PBP).  Red Arrow[Read More](http://www.oracle.com/technetwork/java/javame/javatv/overview/about/index.html) |  |
| [[Java Card](http://www.oracle.com/technetwork/java/javame/javacard/index.html)](http://www.oracle.com/technetwork/java/javame/javacard/index.html)  **Java Card Technology** | What is Java Card Technology? Java Card technology provides a secure environment for applications that run on smart cards and other devices with very limited memory and processing capabilities. Multiple applications can be deployed on a single card, and new ones can be added to it even after it has been issued to the end user. Java Card also includes a set of unique tools for developing new products. Red Arrow[Read More](http://www.oracle.com/technetwork/java/javame/javacard/overview/about/index.html) | Red Arrow |

|  |  |
| --- | --- |
| JAVA FOR MOBILE DEVICES |  |
|  | |  | | --- | | Java for Mobile Devices is a set of technologies that let developers deliver applications and services to all types of mobile handsets, ranging from price efficient feature-phones to the latest smartphones. Java is is currently running on over 3 billion phones worldwide, and growing. It offers unrivaled potential for the distribution and monetization of mobile applications.  At the core of the Java Mobile Platform is Java Platform, Micro Edition (Java ME). Java ME provides a robust, flexible environment for applications running on mobile and other embedded devices: mobile phones, TV set-top boxes, e-readers, Blu-Ray readers, printers and more. For over a decade, Oracle has been working along with leading mobile and embedded companies to develop the Java ME Platform through the [Java Community Process](http://www.jcp.org) (JCP). A key achievement has been the definition of the Mobile Services Architecture (MSA), setting a baseline of mobile APIs that developer can target within their applications. In 2011, Oracle and partners will be working within JCP to drive Java ME.next - a proposal for the modernization of Java ME .  In addition to its role within JCP, Oracle is also a provider of high performance Java ME implementations and developer technologies being used to deploy tens of thousands of applications worldwidein the mobile and embedded markets, including:   * [Oracle Java Wireless Client](http://www.oracle.com/us/technologies/java/ojwc-170429.html): a multitasking Java ME runtime optimized for the leading mobile phone platforms. * [Java ME SDK](http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk/index.html): a state-of-the-art toolbox for developing and testing mobile applications. * [Light Weight UI Toolkit (LWUIT)](http://www.oracle.com/technetwork/java/javame/javamobile/download/lwuit/index.html): a compact library for the creation of rich user interfaces. * [Oracle Java ME Embedded](http://www.oracle.com/technetwork/java/embedded/downloads/javame/index.html): designed and optimized to meet the unique requirements of small, low power devices. | |

|  |  |
| --- | --- |
| JAVA TV | Java Logo |

Java TV is a Java ME-based technology that provides a performant, secure, and easy to implement solution for developing Java applications that run on TV and set top box devices. Using the Java TV runtime, a developers can easily create applications, such as Electronic Program Guides (EPG's), Video-on-Demand (VOD) clients, games and educational applications, applications for accessing internet data (e.g. weather, news tickers, social networking), and, on most Blu-ray Disc titles, the user interface and bonus content.

http://www.oracle.com/technetwork/java/javame/index.html

http://www.oracle.com/technetwork/java/javame/javatv/download/index.html

http://www.oracle.com/technetwork/java/javame/javatv/overview/getstarted/index.html

http://www.oracle.com/us/technologies/java/mobile/wireless-client/overview/index.html

http://www.oracle.com/technetwork/java/javame/about-java-me-395899.html

http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk/index.html

http://www.oracle.com/technetwork/java/javame/javamobile/download/overview/index.html

http://www.oracle.com/technetwork/java/javame/javamobile/download/lwuit/index.html

Java Tutorial: Java TV and Java ME (Version 8)

<http://www.interactivetvweb.org/tutorials/javatv>

<https://today.java.net/pub/a/today/2005/02/09/j2me1.html>

<http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javame-419430.html#sun_java_me_sdk-3.0-rr-oth-JPR>

<http://docs.oracle.com/javame/config/cdc/cdc-opt-impl/ojmeec/1.0/reference/html/z4000c841293984.html>

<http://www.code4tv.com/c/downloads>

<http://www.oracle.com/technetwork/java/javame/javatv/documentation/index.html>

<http://www.interactivetvweb.org/tutorials/javatv>

<http://www.interactivetvweb.org/content/getting-started-xletview>

<http://www.interactivetvweb.org/tutorials/getting_started/application_development>

<http://xletview.sourceforge.net/>

<http://docs.oracle.com/javame/config/cdc/cdc-opt-impl/ojmeec/1.0/reference/html/z4000c841293984.html>