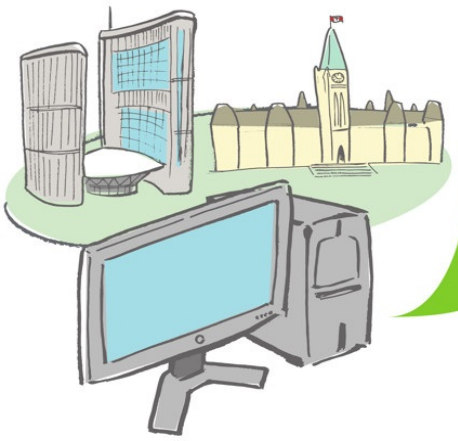
RAHUL BHAGAT

HL7 FOR BUSY PROFESSIONALS

Your No Sweat Guide to Understanding HL7



MSH|^~\&|SENDER_APP|SENT_BY|RECEIVER_APP|RCVD_BY|201310201500||ADT^A04|MSG_ID001|P|2.5|||AL
EVN|A04|201310201500|||ID221^Dude@Terminal
PID|1||PAT416^^^HEALTH_ID||SEBELUS^KANSAS||19480115060O|M|||123 SESAME ST^^TORONTO^ON^A1A2B2 ^CANADA||(416) 888-8088||ENGLISH|M||PAT_AC_721914

MSH|^~\&| RECEIVER_APP|RCVD_BY| SENDER_APP|SENT_BY|201310201501||ADT^A04|R_MSG_ID279|P|2.5|||AL
MSA|AA| MSG_ID001| Got your message

ILLUSTRATED BY: CALVIN HUI

HL7 for Busy Professionals

Your No Sweat Guide to Understanding HL7

Rahul Bhagat
Illustration By Calvin Hui

Anchiove
2015

Table of Contents

Preface

After university, I got a job with a busy, Toronto based, healthcare consulting company. On day two at work, I was handed a printout with cryptic text on it, and a document called interface spec, to read and understand.

This was my introduction to HL7. At that time, I did not realize that this obscure messaging protocol would become my ticket to far off places, and the reason to meet and work with a lot of people.

It didn't take me long to learn HL7, my programming background helped. Later, I realized that my skill is in high demand and I became a consultant. I traveled to different cities and worked on various HL7 projects.

I also started running into people from non-technical background who wanted me to explain HL7 in the elevator or while chatting in their cubicle. There wasn’t any introductory book I could suggest, so the idea of writing one myself.

I'm glad I collaborated with Calvin Hui in writing this book. He not only took care of illustration and design but also nudged me when I was slacking after the first draft. My friend Erik Westermann was a great sounding board and helped me refine my ideas. And thanks to many colleagues who helped me develop my skills. In particular, Derrick Leung, who mentored me when I was just starting out.

So here it is. My idea of an introductory book on HL7. I hope you enjoy reading it.

Part I

Scratching the surface

1. Introduction

A technical book usually implies a dry subject. So its no surprise authors have a hard time figuring out ways to make the book interesting to the reader. HL7 is one such subject. It is a subject that is so high on the scale of dryness and no one comes to it willingly. The only reason someone would read a book on HL7 is because of his or her job. And if you are here, reading this book, then I assume you work in healthcare IT or intend to join the industry soon.

I have made every effort to take out the dryness of the subject and make this book interesting. There are no needless jargons or esoteric concepts thrown casually to trip you. In fact, you will see a heavy reliance on everyday examples and inclusion of background information to paint a complete picture. But HL7 and healthcare system integration are complex subjects so there will be topics that don't make sense right away. Please persevere. Tie a knot and hang in there. Gradually things will make sense.

This introductory book on HL7 goes in detail to explain what HL7 is. It gives you the basic concepts, tells you about the organization behind it and helps you create a mental map of the voluminous HL7 specification document. And, it takes you through a whirlwind tour of some of the most commonly used HL7 messages, all in a short span of time.

Early Railroads

HL7 was created to solve the problems of clinical system integration. But to truly understand the problems of system integration, let’s start with another integration problem we solved centuries ago.

The 1800’s were a time when railways were coming of age in America – just like battery driven cars, drones and other new technologies are coming of age today.

There were literally hundreds of companies competing for a piece of the railway pie. Enterprising companies would buy up land, lay down tracks and run a transport service between cities which had no other means of transportation except for horse-drawn wagons or, if one was fortunate, steamships.

By the time American civil war started (1861), vast stretches of the continent were already connected through rail and work was well underway on the construction of the transcontinental railroad to connect California with the rest of the country.

However, there was one problem. You could not just hop on a train and get off at your destination, like you can today. Because these railroads were built and run by different companies, they used different track gauges (horizontal distance between two rails of the track). This meant you had to get off and change trains whenever you hit a junction with two different gauge widths. There were well over twenty different track gauges being used at the time of the civil war. The army had to constantly load and unload cargo in its effort to get supplies to the troops. This was a serious problem!

And it was the reason that finally made the American government to push for the conversion of all railway tracks to a standard gauge—4 feet and 8.5 inches, the most commonly used gauge width. More than half of the existing tracks were built to this width so it was easiest to convert the remaining tracks to this width and achieve standardization.

Standardization of rail tracks was the first step towards creating an integrated system where goods and people could move freely across the whole network. It was followed by the development of a common signal system, time zones, harmonized train schedule, fixed coach height, a standard coal and water supply system and on and on.

It was evident that an integrated system needed a standard way of doing things.

Evolution of Healthcare IT Systems

Today, we are in a (somewhat) similar situation with the movement of healthcare information. It cannot seamlessly flow from one system to the next. Each organization has its own way of storing and sharing information. Whenever health information needs to move across organization boundaries, it hits the incompatible standards roadblock. Someone has to unload and reload the information.

Healthcare IT systems have evolved similar to railroads. Initially, hardware costs (think multi-million dollar mainframes) were the biggest factor, so only a few teaching hospitals with deep pockets had the means to build a system. These were primarily stand-alone systems meant to serve a specific purpose. For example, to manage patient population in a large hospital.

Then the hardware cost came down and minicomputers arrived on the scene. A computer could be had for less than $25,000 and didn’t need a

room to house it. This allowed smaller players and even departments within a hospital to purchase systems of their own. Pharmacies installed systems to track prescriptions and dispensed medication while laboratories set up systems to track requests for tests and their results.

This led to dramatic improvement in productivity for these organizations but there was no free flow of information between the clinical systems. The problem was lack of standardization. Information from one system had to be unloaded to paper and transported to where the other system was. Then a human operator would reload the information to the other system by manually typing it in.

Of course this was the worse case scenario. Improvements were made. Information was loaded on floppy disks and electronically moved to the other system. Still, there was no free flow of information between systems. This prevented us from realizing the true potential of electronic systems.

Then some IT vendors came up with a solution. Replace stand-alone systems with an integrated product - an EHR (electronic health record). If you are familiar with Cerner, Epic or Meditech then you know what I am talking about. A large system with modules for every department.

This eliminated the need for health information to cross system boundaries. Within the system, the modules would use a standard way of storing and sharing information and this would allow the information to flow seamlessly within the organization.

This approach worked well. EHRs have been very successful in eliminating the problem of integrating systems within an organization and they continue to be one of the cornerstones of the healthcare IT structure.

But what about sharing information outside the organization? Healthcare organizations don't work in isolation. They need to share information with insurance companies and send patient care information to the government. They have to constantly communicate with the outside world.

To use our railway analogy, this was similar to the situation where each state could set its own standard gauge. You could travel all over a state without the need to switch trains but when you wanted to cross the state boundary, you would need to disembark and get on a train that ran on the other state's standard gauge.

Clearly, EHRs were only a limited solution.

There was also the question of what to do with existing standalone clinical systems. These systems were built over many years through substantial monetary investment. An organization would be loath to scrap all that investment & hard work and replace it with an EHR.

Healthcare needed a better solution. It needed a standard gauge to

connect these EHRs, standalone systems, external systems and systems that were yet to be built. It needed to move away from constantly loading and unloading information.

The solution was HL7.

2. What is HL7?

HL7 is an ANSI accredited, OSI level 7, application layer protocol for exchanging clinical and administrative data between healthcare systems.

Chances are, if you are not a network engineer or did not study computer science, then "OSI level 7, application layer protocol" probably means nothing to you.

In lay terms, you can say that HL7 is a language that clinical systems use to exchange information with each other. But even that doesn't tell you anything. When I was learning HL7, the definition raised its own questions and left me with a vague sense of unease. It took a fair bit of research to figure out what HL7 is.

So instead of leaving with a sense of unease, why don't we take the time and figure out what HL7 really is?

Application Layer Protocol

HL7 is an application layer protocol. This means that it defines the rules for exchanging data (clinical and administrative) between applications.

We often use the word system and application in an informal way, which clouds the distinction between the two. Historically an application was the same as a system. An old accounting system, with its hardware and printers and monitors had only one job or application– preparing and maintaining financial records.

Things changed when systems became more powerful and started taking on multiple roles. A great example is your smartphone. It's not just a phone anymore. Making a phone call is just one of the many functions of the device. It has numerous "apps" or applications for all sorts of things.

Similarly, modern computer systems or servers run multiple applications, including clinical applications. When applications communicate with each other, they have to do so through their system. Basically, applications create a message in a language that is understood by their counterpart applications – in our case HL7 – and hand it over to their system for delivery. The system doesn't understand the message. Its job is to reliably deliver the message to the destination system.

HL7 is one such specialized application-to-application language/messaging rule book/protocol – whatever you call it – for communication between clinical applications.

OSI Level 7

HL7 is also an OSI (Open System Interconnection) Level 7 protocol. This is just a formal way of saying that it is an application layer protocol.

Now, we are going to discuss OSI and its levels and that means splashing through packet based, network communication. If you are not interested in it, I would suggest skipping over to the next chapter.

OSI is a reference model that networking guys use to make sense of the network communication model and how things really happen at the bit and byte level.

It is not difficult to understand the OSI model. The secret is proper background knowledge and an understanding of the key concepts. Let's see if we can do that in a few short pages here.

Historical Background

Using electricity for communication started with Samuel Morse, the inventor of the telegraph. He created a simple circuit with a battery, a bowl of mercury and two long wires grounded at ends.

If he dipped a wire in the bowl of mercury, it completed the circuit and current flowed through it. To send a short burst of electricity, he would dip the wire and pull it out quickly. This was like sending electric "smoke puffs" to the other end.

This basic idea was refined into the telegraph and Morse code. The code had two letters – a dot and a dash. A dot was a short puff of electricity and a dash was a longer puff (about 3 times the duration of a dot). Dots and dashes were combined to represent letters and voila! We had electronic communication.

4. Evolution of HL7

We live in a world of standards. As new technologies emerge, a necessary condition for their wide adoption is standardization. We notice it when there is a problem with the standard. As must be evident to anyone in a foreign country not able to plug in a laptop because the power outlet is different.

Standards emerge from different sources. A standard could be imposed by the government, as was the case with the conversion from the imperial to the metric system of measurement. Both the USA and Canada started the conversion but in the States, the government defunded the Metric Board, stalling their conversion. In Canada however, the conversion was completed and we started measuring distance in kilometers and temperature in Celsius.

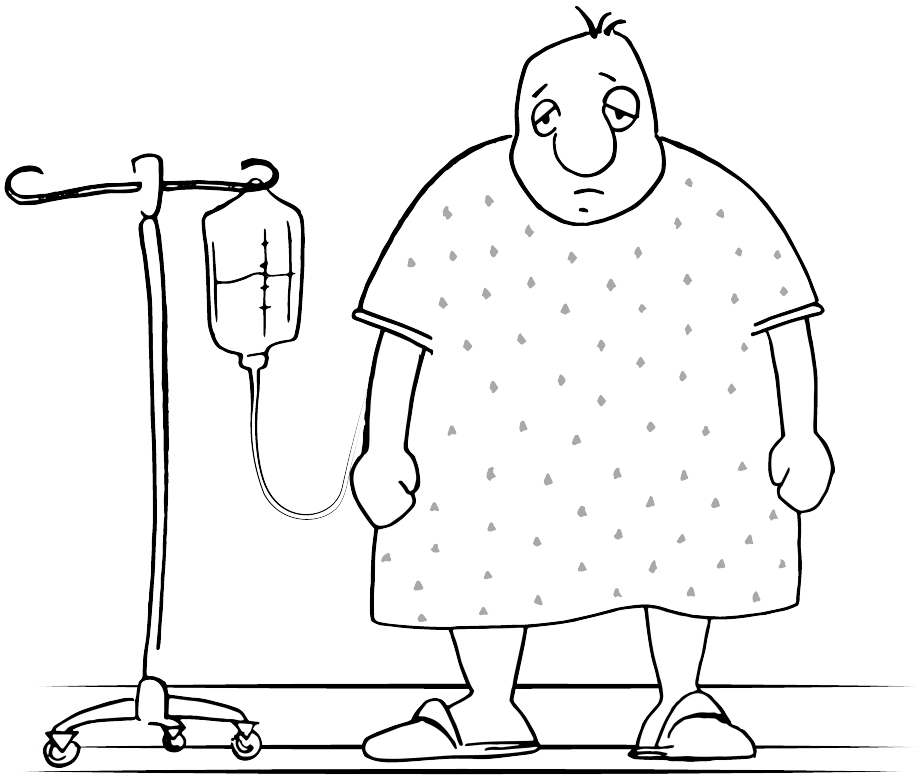
Another source of standard is the industry itself. Companies at the forefront of a new technology vie for competitive advantage by promoting adoption of their technology. We have all heard about the famous videotape standards battle between Betamax and VHS. Sony's Betamax was a superior technology but VHS became the standard because JVC was relentless in promoting it to electronics manufacturers.

A third source of standard is market forces, which leads to a de-facto standard. This was the case with TCP/IP. It became the dominant, and ultimately standard, network-communication protocol as a result of gradual adoption by universities and businesses. By the time the ISO model was developed, it was too late for a switchover. TCP/IP was already baked in.

And finally, there is the deliberate approach where experts get together with the specific intent of creating a standard. This is how the HL7 standard came into being.

PART II

Digging Deeper



6. Message Building Blocks

To the uninitiated, the sight of an HL7 message is often intimidating. A brew of symbols and characters, it looks like something out of the Matrix that is beyond the comprehension of mere mortals. But to be honest, HL7 really is quiet simple and straightforward, once you know how to read it. And for that, you will need to learn about the building blocks of an HL7 message.

Let's take the example of registering a new patient. When the staff at the front desk completes the patient registration and hits enter, it triggers an event: A04 (Register patient). This causes the system to generate a new ADT^A04 HL7 message, which looks something like this.

MSH|^~\&|SENDER_APP|SENT_BY|RECEIVER_APP|RCVD_BY|201310201500||ADT^A04|MSG_

EVN|A04|201310201500|||ID221^Dude@Terminal

PID|1||PAT416^^^HEALTH_ID||SEBELUS^KANSAS||194801150600|M|||123SESAME.ST^^TORON
(416)888-8088||ENGLISH |M||PAT_AC_721914

NK1|1|SEBELUS^MARY|SPOUSE|||(416)888-9999|(647)123-1234|C|20131020

PV1|1|O|ROOM10^BED12^OUTPATIENT|ELECTIVE|||S21195^DRIKOFF^FRANCIS^^^DR^MD||(

PV2|||DAY SURGERY

AL1|1|FA^PEANUT||PRODUCES MILD RASH

See what I mean? Makes no sense. But soon it will.

Segment

The primary building block of a message is a segment. A segment is simply a row of data in the message. So, for the message above, the first segment starts with MSH and ends on line two with AL. It is actually just one row of data, which was wrapped over to the second line. There is a line break after AL and that means end of the segment. The second segment starts with EVN on line three and ends at "Terminal" on the same line, followed by a line break and so on. A new segment always starts on a new line.

The first three characters of each segment is the segment ID. The segment ID is an acronym or the nametag of the segment.

Once we know the segment name, we know the information in that

segment. This is because the main purpose of a segment is to group related information together.

In our example here, there are seven segments (IDs bolded). MSH is the Message Header segment, EVN is the Event segment, PID is the Patient Identification segment and so on. Without even looking at the content of the PID segment, I can tell you it contains the name of the patient, his health ID, date of birth, phone number, address - basically all the information that can be used to identify the patient. Hence the name of the segment - Patient Identification.

MSH|^~\&|SENDER_APP|SENT_BY|RECEIVER_APP|RCVD_BY|201310201500||ADT^A04|MSG_

EVN|A04|201310201500|||ID221^Dude@Terminal

PID|1||PAT416^^^HEALTH_ID||SEBELUS^KANSAS||194801150600|M|||123SESAME.ST^^TORON
(416)888-8088||ENGLISH |M||PAT_AC_721914

NK1|1|SEBELUS^MARY|SPOUSE|||(416)888-9999|(647)123-1234|C|20131020

PV1|1|O|ROOM10^BED12^OUTPATIENT|ELECTIVE|||S21195^DRIKOFF^FRANCIS^^^DR^MD||(
S21195^DRIKOFF^FRANCIS^^^DR^MD||37323|SELF||||||||||||||||||||||201310201500

PV2|||DAY SURGERY

AL1|1|FA^PEANUT||PRODUCES MILD RASH

Message Structure

Segments in a message are always organized in a specific order. This order is called the message structure. Different message types have different message structures but some things are always the same. For example, every message starts with an MSH segment.

If the order of segments in a message is not exactly like its message structure, then that message will become invalid. It will be rejected by the receiving system.

You can get the abstract message structure of any message in the HL7 specification document. The abstract message structure of an ADT^A04 message is in Chapter 3 of the HL7 specification document where event A04 is discussed.

Here is a partial abstract message structure of an ADT^A04 message. It is just a table with three columns: segment ID, segment name and the chapter where that segment is explained.

Partial Message Structure of ADT^A04

MSH	Message Header	2
[{ SFT }]	Software Segment	2
EVN	Event Type Segment	3
PID	Patient Identification	3
[PD1]	Additional Demographics	3
[{ ROL }]	Role Segment	15
[{ NK1 }]	Next of Kin	3
PV1	Patient Visit Segment	3
[PV2]	Patient Visit – Additional Info.	3
….	….	

If you compare the example message to its abstract message structure, the segment order does not match between the two. In the example message, the SFT segment is missing after MSH; PD1 & ROL are missing too.

Does that mean the example message is invalid? No, it's not because the [ ] and {} brackets around those segments make them either optional or repeatable.

Optional / Repeatable / Mandatory

There are two kinds of brackets: square [ ] and curly {}. If a segment ID is enclosed within [square brackets], it means the segment is optional. We can choose whether to include that segment in the message or not. These segments are generally for optional information, such as PD1 (additional patient information).

If the segment ID is enclosed within {curly brackets}, then that segment is repeatable. We can have more than one instance of that segment in a real message. Curly brackets are for segments like NK1 (Next of Kin). If a patient has given contact information for two next of kin (spouse and sister), then the information for each next of kin will need a separate NK1 segment in the message.

If a segment ID is enclosed in both [{square and curly}] brackets then that means the segment is both optional and repeatable. If a segment ID is not surrounded by any bracket then that means it is a mandatory segment. That segment has to be present in the message. So segments like MSH, EVN, PID, PV1, with no brackets, have to be present in a real message.

Based on this knowledge we can see why the example is a valid message. All the missing segments, SFT, PD1 and ROL are surrounded by square brackets. And that means those segments are optional. We can choose to leave them out.

8. Control Segments

From the last chapter, if you remember the discussion about the anatomy of a message, control segments are the segments in the head of a message. They only carry meta-data information about a message.

There are about a dozen control segments defined by HL7. They are all explained in chapter 2 of the HL7 spec. Fortunately, we only need to know about a few of them to account for the vast majority of cases. For example, there are control segments for breaking a very large message into smaller pieces and control segments for batching together a large number of messages. These control segments are not used that frequently and for a general understanding, you can skip them.

There are five control segments that you really should know about – MSH, EVN, NTE, MSA & ERR. We will start with MSH, the ubiquitous control segment that every message begins with. It is the most important control segment. If you decide five is too many for your precious time and you are only going to read about one, then let this be the one.

Message Header Segment (MSH)

The message header segment (MSH) is the most important control segment. Every HL7 message starts with this segment. When an HL7 message is received by a system, it is the MSH that tells the receiving system where this message came from, the information it contains and how it is supposed to be acknowledged.

This is a segment you want to know well.

To get a better understanding of the contents of this segment, let's use the MSH segment from our example A04 message and explore its contents.

MSH|^~\&|SENDER_APP|SENT_BY|RECEIVER_APP|RCVD_BY|201310201500||ADT^A04|MSG_ID

If you break the segment down into its separate fields, it gets easier to figure out the content. Remember | is used to separate fields.

MSH-1: | (Field Separator)

MSH-2: ^~**&** (Encoding Characters)

MSH-3: ***SENDER_APP*** (Sending Application)

MSH-4: ***SENT_BY*** (Sending Facility)

MSH-5: ***RECEIVER_APP*** (Receiving Application)

MSH-6: ***RCVD_BY*** (Receiving Facility)

MSH-7: ***201310201500*** (Date/Time of Message)

MSH-8:

MSH-9: ***ADT^A04*** (Message Type)

MSH-10: ***MSG_ID001*** (Message Control ID)

MSH-11: ***P*** (Processing ID)

MSH-12: ***2.5*** (Version ID)

MSH-13:

MSH-14:

MSH-15: ***AL*** (Accept Acknowledgement Type)

Note that some of the fields are empty (e.g. MSH-8). This is perfectly fine. Remember, not every field in a segment is required to have a value. If you refer to the segment attribute table of MSH, you can confirm that all missing fields are optional.

Now, here is a little insider information. There are only a few fields in each segment that are really important and regularly used. That is why you see the usual pipe pattern (||||) in HL7 messages. The consecutive pipes are nothing but empty fields.

So keeping with our tradition, and saving you precious time, we are going to discuss only the most important fields in a segment.

In the MSH segment, owing to the fact that it contains most of the meta-data information, there are many important fields. It is the heaviest control segment. Some of these important fields are required and others are optional, but they almost always have a value.

If you refer to the segment attribute table of the MSH segment, HL7 requires that the following fields always have a value.

MSH-1: Field Separator

MSH-2: Encoding Characters

MSH-7: Date/Time Of Message

MSH-9: Message Type

MSH-10: Message Control ID

MSH-11: Processing ID

MSH-12: Version ID

It is easy to find out which fields are required in a segment. Just go to the segment attribute table and look for the letter R in the optionality (OPT) column.

Besides these required fields, there are some other fields (below) in MSH, which are optional but regularly used. They are important and I think you should know about them.

MSH-3: Sending Application

MSH-4: Sending Facility

MSH-5: Receiving Application

MSH-6: Receiving Facility

MSH-15: Accept Acknowledgement Type

MSH-16: Application Acknowledgement Type

Keep in mind though that this is only my personal opinion. Others can argue that there are other optional fields that are important and some here are not. I'm not denying it. But from my experience, I believe these are the important fields in the MSH segment.

Now let's get familiar with these fields because the name of the field doesn't tell you even one tenth of the story.

MSH-1: Field Separator

Usually, the first field in a segment is the field that follows the segment ID. So technically "encoding characters" should be the first field of MSH segment. But with MSH, there is an anomaly. The first field (MSH-1) always defines the symbol that will be the field separator (delimiter) for the entire message. If you remember the discussion about pipe delimiters in Chapter 6, | is the field separator in HL7 messages and therefore, the first field of MSH. But it doesn't have to be. You can choose to have a comma (,) or any other symbol as the separator. If you choose to use a comma, the segment will look something like this.

MSH,^~\&,SENDER_APP,SENT_BY,RECEIVER_APP,RCVD_BY,201310201500,,ADT^A04,MSG

This would be a perfectly legitimate HL7 segment. However, | has become such a de facto standard that no one really uses anything but | as the field delimiter. But it's good to know that we have the power to change it.

MSH-2: Encoding Characters

Encoding Characters are the four symbols ^ ~ \ **&** that HL7 reserves for message construction.

These characters have special meaning, which allows applications reading an HL7 message to distinguish between components and subcomponents of a field, read repeating fields, and translate symbols.

The encoding characters, in order, are - Component Separator (^), Repetition Separator (~), Escape Character (\) and Sub Component Separator (&). The position of each character is fixed in the field. First the component separator then the repetition separator and so on.

By having these symbols in MSH-2, we are basically saying that in this HL7 message, ^ will be used to separate components, ~ will be used to separate multiple occurrences of a field, \ will be used for special characters and & will be used to separate sub components.

But shouldn't this be hardcoded in systems that read HL7 messages, instead of including it in every message?

Good point. The reason encoding characters are included in every HL7 message is because these characters are customizable too, just like the field separator |.

HL7 gives you the option of selecting your own encoding characters. If you don't like ^ and would rather have # as the component separator in your messages then all you have to do is replace ^ with # in MSH-2. As a result, your encoding characters would be #~\&. The # symbol will now be the component separator.

But this whole discussion is pointless! Over the years, these symbols have become a de facto standard. I'll bet, many folks who have been working with HL7 for years, don't know that you can change these symbols. I have never come across a message where a different set of symbols were used.

Before we move to the next field, you need to know more about the other two encoding characters – the Repetition Separator (~) and the Escape Character (\).

Repetition Separator (~): This is the symbol that separates multiple values in a field. Remember the section on segment attribute table in chapter 7? Some fields are repeatable and they can have multiple values. ~ is the symbol that is used to separate those values in a field.

In the MSH segment, field MSH-18 and field MSH-21 are repeatable. This means, whenever those fields have two or more values, the values will be separated by the ~ symbol. If a system reading the message comes across the ~ symbol, it will know right away that what follows is the next value of the field.

Escape Character (\): HL7 reserves encoding characters for message construction and they have a special meaning in the message. What happens if you need to use one of those special characters as part of the data? The

application reading the message is going to be all confused!!

In real world applications, the most troublesome of these special characters is the ampersand symbol (&). It is used for “and” (as in Ben & Jerry’s) and is also a commonly used symbol in programming languages like HTML (which could be embedded in an HL7 message). So, sooner or later, you are bound to come across the & symbol in the body of an HL7 message.

What happens if these characters are part of the data? Let’s consider an example.

Ben & Jerry’s Diagnostic Center sends the result of a test as an HL7 message to the ordering hospital. The hospital system receives the message and starts reading it to parse the data (pull out field values) and save it to a database. When the system gets to the Sending Facility field (MSH-4), it will read “Ben” and then run into the & symbol. At that point, the system is being told that the name of the facility has a sub-component. Facility names don’t have sub-components (if you check the segment attribute table), so in all likelihood the system doesn’t have a corresponding field in the database to save the value “Jerry’s Diagnostic Center”.

This is a recipe for application failure. Let’s assume this is a futuristic, can-handle-anything kind of system, but even then the system is only saving “Ben” as the name of the sending facility, which is incorrect. The doctor reading the lab report will see that “Ben” sent the test result. I don’t know how much faith she will have in the report.

So what do we do? We can’t ask Ben & Jerry’s Diagnostic Center to change its name.

This is where the escape character comes to the rescue. If characters, which have special meaning in HL7, need to be transmitted as part of the data, then all one needs to do is replace the character with its corresponding escape sequence. The system reading it will read the escape sequence and replace it with that special character before saving it.

An escape sequence is nothing more than one or more characters surrounded by the escape character (\). Every special character in HL7 has a corresponding escape sequence. There are many escape sequences for formatting and highlighting text, and you can even create custom escape sequences.

Here are the escape sequences for the symbols we have already discussed:

Escape Sequences

Use	Special Character	Escape Sequence
Field Separator	|	\F\
Component Separator	^	\S\
Sub-Component Separator	&	\T\
Repetition Separator	~	\R\
Escape Character	\	\E\

Source: HL7 Specification Document v2.5, Chapter 2

Guess how Ben & Jerry's Diagnostic Center will be encoded in an HL7 message. You will replace the encoding character & with its corresponding escape sequence so that you will have "Ben \T\ Jerry's Diagnostic Center" encoded in the message. The receiving system will recognize \T\ as an escape sequence and replace it with the & symbol when the data is saved locally.