



Lobster



THE COMPACT GUIDE TO EDI.

**EVERYTHING YOU
ALWAYS WANTED TO KNOW
ABOUT ELECTRONIC
DATA INTERCHANGE.**

DESIGNED BY
GIFFEN, INC.
HURON, CALIF.
100 LBS. NET WEIGHT
PRODUCE OF U.S.A.

THE COMPACT GUIDE TO EDI.

**EVERYTHING YOU
ALWAYS WANTED TO KNOW
ABOUT ELECTRONIC
DATA INTERCHANGE.**

CONTENTS.

GENERAL INFORMATION _____ 4

E-what?? A quick game of buzzword bingo _____ 4

Is this something I need? _____ 7

EDI, EAI and beyond _____ 11

CONCEPTS _____ 15

EAI _____ 16

EDI _____ 20

Operational variants _____ 25

 In-house _____ 26

 SaaS _____ 32

 EDIaaS _____ 34

 Open Internet vs. VAN _____ 36

ESB _____ 39

ETL/ELT _____ 40

SOA _____ 43

DATA FORMATS & MESSAGING STANDARDS _____ 45

Introduction _____ 46

UN/EDIFACT _____ 51

FIXED RECORD _____ 65

 VDA _____ 68

 Fortras _____ 72

XML _____ 75

 openTRANS _____ 86

 BMEcat _____ 88

CSV _____ 92

IDoc _____ 96

ANSI ASC X12 _____ 101

ENGDAT _____ 105

Miscellaneous _____ 108

COMMUNICATION PATHS, DATA SOURCES & DATA SINKS ___111

Introduction	112
Public paths	115
Email	115
FTP	117
HTTP	119
AS2	125
SFTP/SCP	131
X.400	132
OFTP	135
WebDAV	138
Internal paths/sources/sinks	140
File system	140
SAP ALE and RFC	144
Databases	146
Message services	150
AS/400 DataQueue	153
Miscellaneous	156

SOFTWARE SOLUTIONS _____ 157

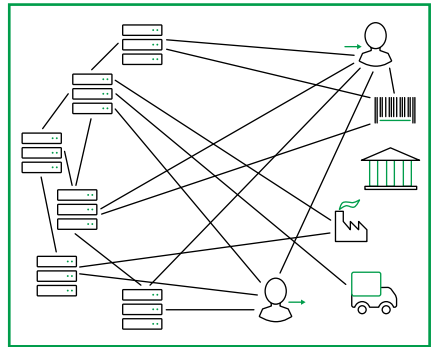
GENERAL INFORMATION.

→ E-WHAT??

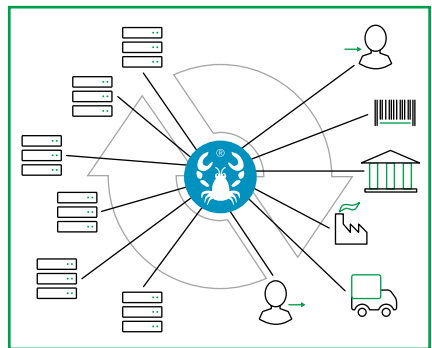
In this manual, we will introduce you to concepts that relate to the interactions between different parts of the IT landscape of an individual company, or of multiple related companies. By doing so, we hope to shed some light on the countless acronyms, keywords and marketing terms you will encounter. You can find more information on all these concepts on the following pages.

EAI

EAI stands for **Enterprise Application Integration**. In this context, the word “Enterprise” is referring to a business. EAI therefore involves integrating all the different applications within a business or organisation – or in other words, joining them together in a network. This is not done by arranging for each individual application to be able to talk to every other application but by creating an intermediate layer – a shared communications platform – which the different applications are connected to via communications interfaces or adapters, so that almost all these applications can communicate and collaborate with each other. This is also where the processes – or the rules of collaboration – are defined.



EAI: interface before



EAI: interface after

E-WHAT?? A QUICK GAME OF BUZZWORD BINGO

EDI

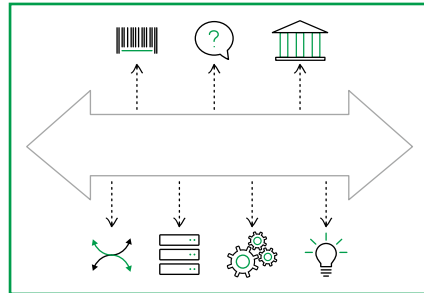
EDI stands for **E**lectronic **D**ata **I**nterchange. It may sound straight forward, but it's highly complex.

In a sense, even as basic as sending a fax is a form of electronic data interchange – after all, the fax contains data and is transmitted electronically. However, this example has little in common with the EDI we are talking about here. EDI is intended to allow applications (by which we mean programs) to exchange data with each other largely automatically, and across the boundaries of the enterprise. This is very similar to EAI, but it takes place between two or more companies instead of within a single company.

For example, when a customer's software sends an order by way of a file directly to a supplier and that order lands in the supplier's ordering system without human intervention, that's EDI.

ESB

ESB stands for **E**nterprise **S**ervice **B**us, which is a somewhat vague term. An ESB can be a concept, a technology, an infrastructure or a piece of software that supports the integration of different applications within an enterprise, a department or a project. In other words, the ESB can form the foundation for EAI. It connects different applications via adapters, transports data between them, and helps to transform the data so that the systems can understand each other. However, it has no understanding of more complicated relationships and business processes, as these things once again fall under the category of EAI.



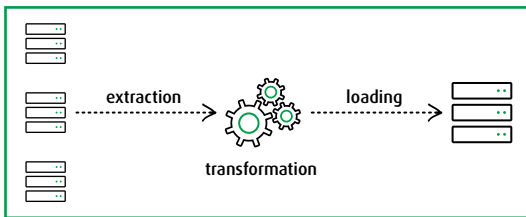
ESB: Enterprise Service Bus

E-WHAT?? A QUICK GAME OF BUZZWORD BINGO

ETL/ELT

The acronym ETL stands for **Extract, Transform, Load** and is synonymous with retrieving (or extracting) data from one or more systems, transforming them, and loading (or strictly speaking, saving them) to a target system – typically a database. ELT means the same thing, just in a different order where the data are first pumped into the database before they are transformed. All this falls under the scope of the data warehouse, the point of which is to process data from different sources and systems in a standardised manner in order to make them available for statistics and analyses (among other things).

The necessary standardisation is achieved through transformation, which not only



converts the different formats used in the source systems into a standard format for the database, but also improves the quality of the data (by eliminating duplicates and so on).

ETL: process

SOA

SOA stands for **Service-Oriented Architecture**. This refers to many different applications – which can even be spread throughout the world – providing an interface through which their services can be requested. Generally speaking, this might involve calculating the distance between two coordinates, or, more specifically within an individual company network, it might be asking when a particular lorry is available for a delivery. Both of these services may also be needed in combination in order to deliver goods, so there may also be an additional service that calls up both as a ‘two in one’.

However, nobody needs to know how the service in question is implemented; they only need to know how to use it. For that reason, only the interface to the service needs to be known to the outside world. An example of this is web services for data exchange.

→ IS THIS SOMETHING I NEED?

If you have read through the buzzword bingo, and ideally the detailed descriptions of the individual concepts too, then you will now have a better idea of what terms such as EDI, EAI and SOA mean. The question now is whether you would need something like this yourself, and if so, what exactly you need.

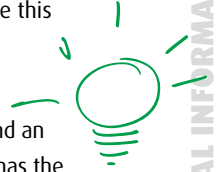
QUESTION 1: WHAT DOES YOUR IT LANDSCAPE CURRENTLY LOOK LIKE?

There are still companies whose IT systems consist of a few PCs with MS Office and an email program. Orders arrive over the phone or via fax, the warehouse manager has the inventory in his head, and delivery notes are filled out by hand. Larger-scale IT systems are nowhere to be found. If this setup is working well for you and none of your business partners are demanding EDI processes, you can park this for now and put this manual to one side.

However, most modern companies no longer work like this, and once they reach a certain size or work with big enough business partners, they begin to see the benefits of digitalising their businesses processes. But, as mentioned in the description of EAI, this often happens bit by bit. Somewhere along the way, the business ends up with a diverse assortment of mismatched IT systems, all of which have different data formats and interfaces and are unable to communicate with each other.

Perhaps you've already gone through this phase and have spent the last few years cobbling together an IT landscape of C programs, shell scripts or even Excel macros that you use to get data from one system to another. Not to forget your major customers or suppliers who have taken to lumbering you with data in their chosen format and now those same customers are threatening you with penalties for not sending their data back to them in the same format. Your programmer is usually on hand to rustle up a number of new programs and scripts to solve the problem – each of which is designed to somehow carry out a particular task. If they still remember how six months later, it will be nothing short of a miracle. And as soon as the next business partner comes along with slightly different requirements, it will be time for the next few thousand lines of code.

Please, stop what you're doing immediately! If you carry on in this way, there will be hell to pay. Not to mention what would happen if your programmer quit. In that case, you would be left with all manner of code and absolutely no idea what is going on in the background or how you could ever make any changes to it.



IS THIS SOMETHING I NEED?



QUESTION 2: HOW PRECISE ARE YOUR FUTURE REQUIREMENTS?

Until now, what has been happening so far in the space between your individual IT systems, and between your business and the outside world? You should start by analysing this in depth. Has communication with the outside world virtually ground to a halt? Do data need to be transported and reconciled between your own systems in order for that to change? In that case, you are looking more at EAI, and possibly also ETL.

Do orders come in from outside your organisation, and do order confirmations, delivery notes and invoices need to be sent? Do you regularly send electronic catalogues to your customers? Do you have lorries on the roads whose locations can be tracked (via GPS services)? Whenever data flows to you from outside your company or vice versa, that falls under the category of EDI.

What systems do your organisation operate? Databases? SAP? Older ERP systems that can only import and export proprietary file formats? Your setup determines how and with whom a possible future EAI or EDI system needs to be able to communicate. And how does the outside world talk to you? Do you only receive Excel files? Is that likely to remain unchanged? Shouldn't your system also be able to handle general standards in order to be future-proof? Standards such as EDIFACT, X12, XML, and so on? Does all your communication take place via email, or are secure transfer paths such as AS2 or SSH-based communication also necessary?

These questions are also relevant when determining what your future system needs to be capable of. And now you're taking a closer look at things, it becomes clear that there's quite a lot to consider, isn't there? Perhaps a dedicated EDI or EAI system won't be sufficient, as the tasks of both areas need to be covered. Oh, and let's not forget the regular reconciliation between multiple internal databases, which falls under the scope of ETL. And then there are your customers, who would like to send you queries about your current inventory, for example. A problem that can be brilliantly solved using web services. You can either deploy multiple specialised tools for each specific area, or one single tool that may not match the optimum performance of the specialist tools, but excels at bringing everything together under one roof.

QUESTION 3: WHAT SHOULD EACH TOOL OFFER YOU?

Aside from EAI, EDI or ETL-related abilities, there are a number of additional aspects to keep in mind:

Are you open to the tool requiring a specific system (Linux/UNIX, Windows, or even AS400/iSeries)? Or do you expect it to be platform-independent? Or better yet: should it be installable in just a few clicks via Docker containers, with no programming required? What's the operability like? Do users need to learn a specific programming language before using the software? Or should the focus be on professional matters, with the software operated more intuitively? With the latter, do bear in mind that EDI and EAI involve highly complex processes that even trained professionals struggle to patch together in just a few clicks. So please: be sure to manage your expectations as to the intuitiveness of the solution.

What does the monitoring look like? Do you always have a clear overview of whether everything is running smoothly, or whether there are any errors that need fixing? Does the system actively notify you of any problems?

How important is transparency to you? Why not simply open your browser to display all important system information in a quick and straightforward way in modern HTML5. All under the motto 'any place, any time, any device' – whether laptop, tablet or mobile phone.

Data security is also a key consideration for business-critical data. Can invoices be archived for the period required by law, for example? Are all important data and logs kept available for inquiries at a later date?

Additional specialised requirements may also be relevant, particularly when it comes to EDI. Some countries, for example, still require digital signatures for certain types of data (especially invoices). Although this obligation has long since been abolished in Germany, you may need to be able to integrate a signature system when doing business with partners in certain countries.

What's more, a standard tool often provides you with an endless list of functions. And yet, as luck would have it, you may well end up with a highly specific requirement which still can't be met using an 'out-of-the-box' solution. Does the solution allow for you to extend the system yourself? Not everyone has their own in-house programmer. But if you did, then

IS THIS SOMETHING I NEED?

GENERAL INFORMATION

the software you deploy should offer interfaces that make it possible to integrate and add your own extensions. Or the manufacturer should at least offer to develop these extensions for you!

Last but not least: What's the situation with support? How quickly does the manufacturer respond to problems? Do they also offer help with more straightforward questions? What do people say about them? Spending hours on hold for the customer hotline is a thing of the past! Why not receive direct support in just one click with the help of detailed explanatory videos or contextual documentation?

When reading question 1, did you come to the conclusion that a tool like that wouldn't actually be such a bad idea? Then you should also carefully consider the points listed under questions 2 and 3. If you know what features your possible software should have then see what's on the market. Have your key scenarios evaluated by different manufacturers (how much time will it take to implement, how expensive are they, and will they even work?), and weigh up which product meets the majority of your requirements. It might not be the cheapest one, but it's better to make the right investment up-front than to regret buying the wrong program after endless rounds of breakdowns, extra work and lost time.

The EDI experts at **Lobster** will also be happy to help answer any of your questions about EDI and EAI (info@lobster-uk.com).

 **EDI, EAI AND BEYOND****WHAT DO WE MEAN BY 'BEYOND'?**

At the start of this manual, we explained the terms EDI and EAI (Electronic Data Interchange and Enterprise Application Integration). Put very simply, these terms relate to the transportation of data between different software systems and the transformation of those data into different formats, with the difference between EDI and EAI being more organisational than technical in nature.

In classic EDI/EAI, the transformation is limited to syntactical changes. In other words, a file in XML, CSV or EDIFACT format is converted into CSV, fixed record, Excel, IDOC, etc. The field contents are either copied across unchanged or subject to formal transformation. Examples of formal transformations include different date formats, decimal points, fill characters, units of quantity, and so on.

However, as the interfaces grow more complex, so do the requirements placed on the internal logic of the interfaces. **As a result, more and more internal functionality is shifted to the interfaces.** This internal functionality might relate to typical business logic (item numbers, delivery blocks, checks on delivery dates, and so on). But in other cases, a separate logic may be required that is used solely to return data to the person submitting the query. In addition, data may also be generated during conversion that add to or expand upon the logic previously available in the company.

We have listed a few examples below that go beyond the classic terminology of EDI/EAI and help clarify the explanation provided above:

- **Checking reference data**

Master data such as item numbers and custom numbers are checked for validity during conversion. If any values are invalid, the data might be rejected with an error message, or the incorrect data might be replaced with dummy values. During replacement with dummy values, the value received is either transported to the target system in a text field or forwarded via email. This solution is used when the target system is unable to process errors in the master data as conveniently as required.

- **Notifying specialist departments**

During conversion, a check is carried out to gauge whether the order is urgent. If it is, the relevant department is notified via email during conversion to save time. This solution is used when the target system is unable to display urgent orders individually, quickly or clearly enough to users.

- **Building a tracking system**

When order data are received by a company with a heterogeneous software landscape, the orders pass through various different programs before the goods eventually leave the company. If the status of an order needs to be verified during processing, company employees have to search various systems to find the answer. This effort can be reduced if the order status is written to a database during conversion between the different programs. This limits enquiries to a single system. It also gives external partners the opportunity to carry out their own enquiries via a web interface.

- **Temporary storage of partner reference values**

When sending their data, a partner will include reference values that need to be included in any responses sent back to them. Yet these reference values aren't used in the target system, and there are no available fields for these values. So, during conversion, the reference values are temporarily stored in a database from which they are then read when the response is sent. This solution allows you to avoid improper use of fields in the interface and the target system. It also sidesteps the issue of having to adapt the target system.

- **Aggregating order items**

During conversion, multiples of an item in an order or repeat deliveries to the same customer, for example, are aggregated within a single transfer. This is typical business logic that should only be implemented during conversion if the target system does not have the necessary features.

- **Recording data for statistical purposes**

During conversion, additional data relating to volumes, insured values, numbers of items etc. are written to a database. These data can then be statistically analysed at a later date using external programs.

- **Generating supporting documents for EDI processes**

During conversion, the data are used to generate additional human-readable documents in PDF or Excel format, which are sent the partner. Documents of this kind are mandatory for some companies, e.g. for summary invoices.



CONCEPTS.

EAI	_____	16
EDI	_____	20
Operational variants	_____	25
In-house	_____	26
SaaS	_____	32
EDlaaS	_____	34
Open Internet vs. VAN	_____	36
ESB	_____	39
ETL/ELT	_____	40
SOA	_____	43

CONCEPTS.

→ EAI



WHAT IS EAI?

Over the course of time, a given company will tend to accumulate an enormous amount of software: warehouse management, order processing, an ERP system, and much more. Often, the individual programs are introduced one by one and the application seemingly most suited to the relevant task is chosen each time. This almost inevitably results in each program coming from a different manufacturer. While each solution works with its own file formats, databases, and so on.

Now, it would make a lot of sense to connect these different programs with each other, since – at least in the business logic – they work on the same data. For example, it would be great if a program for registering orders could check in with the warehouse management software to see whether a particular item that has just been ordered is in stock, or if it needs to be reordered. Then the customer would immediately receive a reliable estimated delivery time. And the account manager would no longer need to phone the warehouse manager to check. The customer is notified automatically.

That's where EAI comes into play. After all, warehouse management and order entry aren't the only two systems that need to be able to talk to each other. At the end of the day, every single IT system will most likely need to be able to talk to at least half the others in the company. So, rather than teach each of them the dialects (or interfaces) used by the other systems, communication is handled centrally via a single system acting as an interpreter of sorts. This is the EAI system.

A simple example:

Of course, the example we looked at just now was a quite basic, but it will serve as an introduction. With that in mind, let's assume that an account manager wants to input a new order into the order entry system. This system is capable of generating an order confirmation that can be sent to the customer. The fact that both of these processes should run in a largely automated manner falls under a different topic (EDI).

For now, however, let's say this order confirmation should also display the delivery time – or at least an approximation of it. To do that, it is important to know whether all the items ordered are in stock, or whether any of them need to be ordered in. And if they do need to be ordered in, the delivery times should also be determined. But one thing at a time.

First things first, the order comes in and the order entry system is given a list of items. This list includes item number 4711. Twice. The question is: do we have two of that item in stock? The EAI system goes and checks.

Conveniently, the warehouse management system works with a normal SQL database. So, it doesn't matter whether it's an Oracle, a MySQL or even an MSSQL database, all we need to do is send a statement over to the database, which then returns the current inventory level of the item in question. Anyone who knows SQL will understand it:

```
SELECT stock FROM inventories WHERE itemnumber = "4711"
```

For everyone else, here is the translation: from the table "inventories", give me the value of "stock" in the row with item number "4711". The database now returns the necessary information - let's say the number 543. That means we have more than enough items in stock. As a result, we can offer a very quick delivery time, as we can send out two today.

That's EAI. Simple, right?

The key points are: The order entry system doesn't know anything about the warehouse management system or its database. Either it informs the EAI system or the EAI system finds out for itself that there is a new order. The EAI system then runs a check against the warehouse management system and hands the information back to the order entry system so that it can issue an order confirmation with realistic delivery times.

OK, we admit, it's almost too simple! It's perfectly possible that the inventory data can't be retrieved from the database so easily. Or the database might report that there are no longer enough items in stock. Let's say that item 4711 has run out and needs to be ordered in. Now, things get more complicated.

A more complex example:

So we're out of item number 4711. This means we need to order it from our supplier, who unfortunately markets that product under its own item number, which our order entry system doesn't recognise. As a result, after getting the red light from our warehouse management system, the order entry system needs to find out the supplier's item number for this particular product - as well as who will deliver it to us, of course.

Let's keep things simple again and take a database that contains both the internal item number and the supplier's number. The database is different this time, but the principle is the same. We'll spare you the statement – this isn't an SQL course, after all. The answer comes back: item number "A08/15" and supplier "Hardware Provider", since this particular item is a piece of hardware. This information could just as easily be obtained from SAP, in which case the request would be made via an RFC call.

However, we now find ourselves in an awkward situation: we really wanted to write about EAI – i.e. the integration of internal systems – but we now need to make a request to the company Hardware Provider which will cross the boundaries of the business, and thus falls under the category of EDI. That's the problem with these two concepts – they aren't always so easy to keep separate. Of course, we can now do the following things: Let's assume that we have an EDI package to go with our EAI software, so we can now make a request to that. We want the EDI system to ask Hardware Provider when item 4711 – aka A08/15 – can be delivered to us. Our EDI software is equipped with a SOAP interface for this purpose, which can be called up as a web service. So let's put together a quick request:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header />
  <soapenv:Body>
    <requestdeliverytime>
      <supplier>Hardware Provider</supplier>
      <ItemNo>A08/15</ItemNo>
    </requestdeliverytime>
  </soapenv:Body>
</soapenv:Envelope>
```



The EDI software forwards the query – if possible as a web service call – to Hardware Provider's system, and then delivers the result back to us. But first it needs to check what kind of business Hardware Provider is, how to get in touch with it, and so on. It might retrieve the information from a database, which... we could just have easily gone and done ourselves!

If you're now thinking, "What rubbish. Why have two systems when the work could easily be done by one?" then you're quite right. So now our EDI system should be able to query databases in the same way as the EAI system? And the EAI system should query web services in the same way as an EDI system? If that's the case, then having two different systems would be completely pointless.

This is precisely why everything should really be combined within one single system. However, our aim is to provide an initial explanation of the different concepts, which distinguish between the integration of systems within a single company and communication between systems owned by different companies. The fact that these two things are closely related and a separation between different systems is usually nonsense, is – in theory – irrelevant.

To quickly finish up our example: once stock levels have been queried in our own warehouse management system or in the supplier's system, the order can be processed further by adding the appropriate delivery time and sending an order confirmation with the expected delivery time to the customer. Oh, but wait – now we're back to EDI!

IN SUMMARY

EAI can be defined as the interconnection of multiple internal IT systems – not in a number of 1:1 relationships, but in a star-shaped or daisy-chain configuration. In this context, people often talk of a 'hub and spoke' architecture, whereby the EAI system forms a hub that communicates with the different systems in its IT landscape via a series of spokes.

**EDI****WHAT IS EDI?**

Whereas EAI refers to communication between IT systems within a single company, EDI refers to the exchange of data between multiple companies. Here too, the IT systems ‘talk’ to each other directly – humans should ideally play only a monitoring role.

Individual companies often deploy software developed by different manufacturers – and when that’s the case, things get a lot more complicated when different companies do business with each other. Not only are there many different standard software systems rattling through the processors, but entire in-house applications too – many of which were written when IT was still in its infancy. As you doubtless already know, nothing lasts longer than a temporary solution.

So how can we get cobbled-together software solutions used by different companies to work together effectively? Often, the closest thing there is to a communications interface is the text files that are generated and inputted. Given that each program has its own data format, this could result in linguistic confusion of near-Babylonian proportions.

To remedy this, there are specific data formats developed as standards by major organisations (including departments of the UN), as well as software that is capable of converting data from one format into another. You might even call them ‘interpreters’, capable of translating between the many different languages and dialects. And bear in mind that interpreters at the UN generally don’t translate directly from Afrikaans into Thai, but work into one of the standard languages (such as English or French), and then from that into the target language. In the world of EDI, those standard languages are EDIFACT, Fortras and BMEcat. The equivalents to the regional languages are generally CSV files or texts in fixed record format, all of which generally fall under the category of ‘in-house’. In other words, this is the format used within a given company – and possibly nowhere else on earth.

If both of the companies involved in the communication have access to an ‘interpreter’ software of this kind, then they can convert data from their in-house format to one of the standard formats before sending them to their partner via their chosen digital route. The partner then converts the standard into their own in-house format before feeding the data into their systems. This makes communication only the third stage that rounds off the EDI process.

Example 1 – Quick query:

In our more complex EAI example, we had to ask a supplier how long it would take them to deliver a particular item to us, since that item appeared in a customer order and we no longer had it in stock. Requests such as this falls into the remit of an EDI software (although, as already established, it is rather impractical to make such a strict distinction between EAI and EDI- after all, this is just a quick external query in the middle of the EAI process).

Therefore, the EDI software is given the task – possibly in the form a SOAP request (here via a web service) – of asking the supplier ‘Hardware Provider’ how quickly they can deliver two copies of the item numbered ‘A08/15’ to us. That item number is the one used by the supplier – in our system, the item is registered under the number ‘4711’.

A web service offers the optimum solution for short requests of this kind. Web services are called up via HTTP, by means of a predefined method. The key point here is that the request is sent to the web service as an XML file, and the answer comes back in the same format. The request/answer itself is packaged inside a so-called SOAP envelope filled with additional information – and that’s it. The answer comes back very quickly, in any case – after all, speed is what we’re aiming for.

Below, you can see the example of how our EDI software receives the request from the EAI system:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header />
  <soapenv:Body>
    <requestdeliverytime>
      <supplier>Hardware Provider</supplier>
      <ItemNo>A08/15</ItemNo>
    </requestdeliverytime>
  </soapenv:Body>
</soapenv:Envelope>
```



The EDI system now needs to find out how to contact the company 'Hardware Provider'. This information might come from a database or from an SAP system (strictly speaking that would take us back to EAI territory), or it might be stored directly in the EDI system somehow. Whatever the case, it turns out that we can – once again – contact the supplier via a web service. As already mentioned, this is the best option for requests of this kind as it is simply the fastest.

Let's spare ourselves another XML example and say that the supplier has guaranteed us a delivery period of three days. Or more accurately, its IT systems have made the guarantee. This is because an answer received via an HTTP connection certainly won't come from an employee, but will be generated completely automatically by the supplier's IT landscape. After all, EDI and EAI processes are also at work on the supplier's side. Our EDI system now reports the three-day delivery period back to our EAI software, which continues processing the order.

Example 2 – Customer order:

As mentioned in the EAI example, a customer order should really no longer be placed by an employee. That would be, well, almost medieval. Orders nowadays should be sent digitally, e.g. as an email or via a website order form. Or if we stay with the B2B sector, in the form of an EDIFACT file delivered directly from our customer's EDI system.

So let's say that an email has come in with an attachment in EDIFACT format.

This message type is called – you'll never guess! – ORDERS. You can find a more detailed description of what something like this would look like in the section on EDIFACT.

The EDI software gets to work on the email and retrieves the attachment containing the order. It recognises what is going on based on the file name, or perhaps the subject line. If need be, it will also look at the contents if it can't make sense of the other identifying features. Then it reads the data and inputs them into the order entry system. At this point, our EAI software takes over, since we are making a clear distinction here between EAI and EDI for the purpose of defining both terms. Eventually, the EAI system finishes its processing, leaving the order in the order entry system along with all information regarding delivery times, predicted delivery times for the customer, any special conditions, etc. An order confirmation with the whole works can now be sent to the customer. Then the EDI system takes over again.

One possibility would be for the EAI software to take all the data for the order confirmation from the order entry system and store them in a directory as a small CSV file, from which the EDI system can once again read the data in order to build them into an EDIFACT document. In that case, the relevant message type would be ORDRSP – all message type names in EDIFACT are made up of six characters, and this one stands for Order Response. Once the ORDRSP has been issued, the customer receives an email from us containing the EDIFACT message as an attachment, which will probably be instantly processed by their own EDI system in turn. Long live modern technology!

Let's just take a quick look at B2C – i.e. communications and trade between companies and end customers (including private individuals).

In this case, there definitely wouldn't be any EDIFACT files flying back and forth. The customer would be more likely to place their order via a web form – e.g. in a web shop with a basket, checkout and all the usual bells and whistles. Yet they too will receive an order confirmation, which would tend to be along the following lines:

Dear Ms. Bloggs,

Thank you very much for your order. Please find your confirmation below.

Item number	Name	Quantity	Price per unit	Total price
4711	Pleasant surprise	2	€ 9.99	€ 19.98

Total: € 19.98

incl. VAT: € 2.76

Predicted delivery date: 30/02/2099

With best wishes,

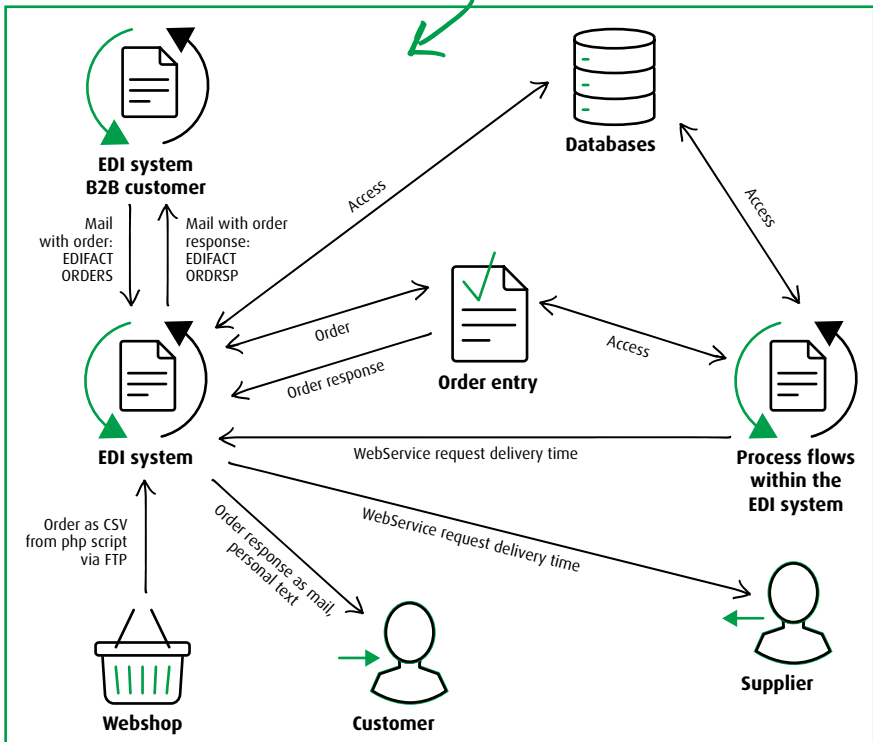
Your Webshop Team

This external communication in this case is very different: to start with, we might have a CSV file that supplies us with a PHP script from the web shop via FTP, while the response email will contain running text instead of an EDIFACT attachment. Yet the process behind it all is still the same.

As such, it makes sense for the different aspects of that process to be built in a modular fashion, as far as possible:

- Communication via different channels
- Reading data from different formats
- Processing in conjunction with internal systems
- Output of response data in various formats
- External communication of the results via various paths

The entire procedure is set out in a diagram below:



On the one hand, it makes a lot of sense to separate the processes that fall under EDI and EAI. Smooth EAI processes do not rely on how the orders reach the order entry system or the confirmations leave it. Yet on the other hand, we often have the same technical processes on both sides – as seen just now in the example of communication with the order entry system, or with databases too. To make matters more complex, we find ourselves sending a web service query from the EAI to the EDI system – which the EDI system then forwards to our supplier – simply in order to keep things strictly separate. It would be far better to have one system that can handle everything, and to establish any modularity within the processes of that system instead.

SUMMARY:

EDI stands for all processes relating to automated communication between the company's own IT systems and the outside world. This involves not just pure data transfer (email, FTP, web service, AS2, etc.), but also converting between different data formats (often via standard formats as an intermediate step), enriching data with additional information, reconciling data and inputting it into or reading it from internal systems. Ideally, humans should play only a supervisory role here, intervening in the event of faults (e.g. unclear data or network problems) and then relaunching the processing once the problems have been remedied. Both EDI and EAI systems should therefore offer comprehensive monitoring, as well as extensive opportunities to respond to errors quickly and efficiently.

→ OPERATIONAL VARIANTS

While EAI and ETL generally only make sense as in-house installations, the situation is far from clear-cut when it comes to EDI. After all, a great deal of communication goes on with the outside world in this area. And unlike EAI, internal systems don't necessarily need to be directly accessible for EDI. It's often enough if files can be exported and imported (data management). That is why we have a whole host of options when it comes to configuring our EDI:

In-house

Here, there are two ways in which the configuration can take place. We can distinguish between self-managed in-house, and in-house with service provision.

SaaS – Software as a Service

With SaaS, also known as cloud computing, the installation itself is outsourced. That means a provider makes hardware and infrastructure available, and takes care of maintenance matters such as the installation of updates. You can then control everything and set up your own processes via the Internet.

EDIIaaS

With EDIIaaS, it is – in principle – possible to outsource not only the system operation and administration, but all the work too. A service provider runs the software in their own data centre, connects your business partners to it, and sets up conversions and other processes. All you have to do is send the data for your partners to the provider's system, and the provider then sends your partners' data back to you.

Open Internet vs. VAN

Communication also forms part of EDI – and here we have two alternatives. You can send data either via the standard Internet, or via a VAN (value-added network). The latter comes with a few additional features beyond pure data transfer, such as secure transfers.

Let's take a closer look at the alternatives:

→ IN-HOUSE

The word 'in-house' simply means that the EDI system runs on your premises, on your network, and on hardware provided by you (whether real or virtual).

This comes with several pros and cons:

Pros:

- All your systems – such as databases, SAP, and everything else besides – are directly accessible, allowing the EDI system to make use of its full capacity.
- The program is simultaneously also able to carry out EAI tasks. Of course, the previous point is of course an essential prerequisite in this regard.
- Everything is fully under your control – nothing leaves your network beyond the data that were intended for the outside world anyway.
- You only need to make a single investment when buying the system, followed at most by a manageable and unchanging support fee. That fee won't go up even if your data volumes increase exponentially.



Cons:

- You need to supply your own hardware and infrastructure, and of course there are certain administrative costs involved too – although those costs will depend on which additional service you decide to purchase.
- The initial investment is somewhat more substantial.

Deciding where to install the software is only half the challenge. You also need to think about who will then work with the system and implement your processes. We can split that decision into two general alternatives (although the boundaries between them are somewhat fluid, of course).

1. Self-managed in-house

This gives you full control over everything. The installation will probably still be carried out by the manufacturer or a service provider, but the day-to-day work will be done by your employees. That means your IT unit (and most larger companies generally have a dedicated EDI team too) will configure the communication with your partners, establish all the processes, conversions, workflows and so on, and of course connect your internal systems too. The manufacturer's support team would naturally stand ready to assist you with this, but it would largely be up to you.

That sounds like a lot of work – and it is – but it's also worthwhile, since everything you do yourself will also be fully understandable to you afterwards. If any changes are necessary, this can be done quickly without having to provide complex explanations of what you need to a service provider, and without having to wait until the provider has time for your project.


Of course, changes to processes (such as switching to a newer version of an EDIFACT message) generally involve a certain lead time, which means this might not be so important. But what if your supplier calls on Friday afternoon and tells you that the IP and name of their FTP server to which you send all of your orders will be changing in two hours? Someone from your IT unit will need to be able to stay on-site to make the changes, since a service provider might close early on Friday or be so busy that they can only get round to it halfway through the following week.

You can also manage extensions to your software in-house, insofar as this is technically possible. And provided you have the right people for the job, of course. But a word of caution here: Even if the manufacturer provides you with the interfaces (or APIs), they

will absolve themselves of responsibility for any lines of code that you have written or introduced into the system yourself. On the other hand, that would still be true if you had written everything for yourself without using any standard software.

In any case, this scenario calls for comprehensive training for your employees. That should start with administration, data backup and logging, but it should also include day-to-day operation, set-up of processes and communication, as well as using the API for in-house extensions (if this is available and relevant to you). Yet because training can generally only cover the fundamentals and just a small proportion of the countless functions found in a complex system of this kind – and because your employees will only retain a small part of the training anyway – quick, reliable and helpful support from the manufacturer is absolutely essential in cases like this. You should therefore research the company in advance and look carefully at its reputation for support. There should also be comprehensive documentation available, as it's generally quicker to look at the manual than to send an email to the support team.

And make sure you don't reinvent the wheel in the process. Connections to standard communication paths (from AS2 to X.400) and the use of standard data formats (EDIFACT, Fortras, VDA, etc.) should be supported by the system through simple templates and pre-set structure definitions.



Pros of this solution (in addition to the general 'in-house' component):

- You know exactly what is going on in your system as you've set it all up yourself.
- That means you can make changes quickly.
- You can respond instantly to errors and problems.
- Updates can be installed over the weekend when the system can be put out of action temporarily (and a service provider would only be available for a higher fee than usual).
- Your internal data will never fall into the hands of third parties. This is particularly important in the healthcare sector with its strictly confidential patient data, as well as among banks and insurance companies.
- If extensions are permitted by the system, they can be implemented quickly.
- Your employees know how everything works. You understand your business processes and don't need to explain them at length to a service provider.

Cons:

- Your employees will need comprehensive training in order to be able to operate and administer the system properly. It will also take them a while to gain enough experience to properly get to grips with it.
- That means you're dependent on quick and effective support.
- You need human resources to administer and operate the system.

2. In-house with service provision

If you chose this option, everything will run on your premises, but the software manufacturer (or an independent service provider) sets up the various connections for you. You monitor the whole thing, and the service provider comes back into play whenever there are problems or you want to make changes.

So what kinds of things can a service provider help you with?

- Administration, including installing updates and bug fixes
- Configuring communication with partners
- Connecting other systems (ERP, database, etc.) in-house
- Applying conversions between data formats
- Mapping out the entire EDI process
- Monitoring ongoing operations and responding to errors

Of course, in each case you can decide what you want to outsource and what you would prefer to manage yourself. But for now, let's take a look at an extreme example in which everything is outsourced – although there is a hard limit to this when it comes to monitoring, since that at least would have to remain under your control in an in-house system. It would also be difficult for a service provider to continuously monitor multiple customer systems that could be distributed around the world.

In any case, you can definitely dispense with any comprehensive training in this scenario. Although you should still make sure you understand the fundamental operating principles of your system – since it is unwise to entrust your business-critical data to a black box. However, in this case it isn't necessary to have detailed knowledge of configuration, administration and process mapping. Whenever you need something, for example you're wanting to connect a new partner, broach a new data format or similar, you can hand it

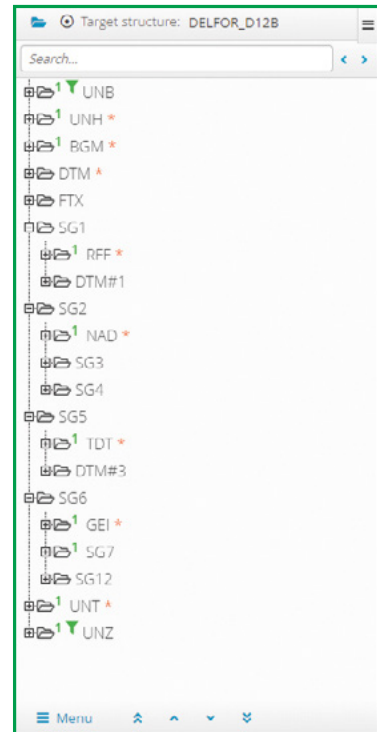
over to the service provider. The provider will have specialists who do nothing else all day and know every inch of the software like the back of their hand.

Yet what they don't know about in detail is the nature of your business. EDI is no field for identikit solutions, since even two processes that seem identical on the surface can look completely different from one company to the next. As an example, let's look at a simple conversion of delivery call-off data from VDA 4905 format into an EDIFACT DELFOR. The VDA data format offers a huge level of freedom, since not all the elements available to the standard necessarily have to be made mandatory. In that regard, the EDIFACT format is somewhat more comprehensive.

The figure below shows just a part of this structure:



Structure of VDA4905 format in Lobster_data



Structure of DELFOR_D12B format in Lobster_data

For example, on the right-hand side of the figure on page 30, you can see an NAD segment under segment group 2 (SG2), where NAD stands for 'name and address'. You can also see another segment of the same type in SG7 (nested under SG6), which relates to processing instructions. But what you can't see here is that the NAD appears again on the bottom of the structure, at the level of the individual items. In other words, there is no such thing as a definitive conversion from VDA4905 to DELFOR – rather, there is a whole host of different possibilities.

And this is where you have a huge advantage over your service provider in terms of knowledge. After all, you know which of the many NAD segments that appear in the DELFOR should be used in your particular case. Or failing that, you can ask the partner directly for whom the data are intended. On top of that, the VDA format only has enough room for the supplier's and the customer's numbers, with no detailed address information at all. All the same, you will know whether it's enough to simply enter the customer and supplier numbers as they appear in VDA format, whether they need to be converted from one number range to another (ILN), or whether the full address details are needed. And more importantly, you will know which of your systems you can obtain this information from, and how you can get it.

Of course, you can explain all this to the service provider who will set the conversion up for you; however, experience suggests that this will result in more time spent on communication, in the form of a lengthy chain of questions and answers. And it all only gets more entertaining when it comes to complex processes that need to take account of particular conditions within your company.

The situation is equally tricky when it comes to administration, communication and connecting systems. On the one hand, of course, the software manufacturer or a specialist service provider will have extensive experience not only with their own systems, but also in relation to interactions with certain third-party systems (e.g. databases). On the other hand, you are still the one who needs to gather the necessary information and pass it on to the manufacturer or provider in order for them to carry out the implementation. To put it concisely: by that point you might as well have done it all yourself.

**Pros:**

- You don't need to build any extensive knowledge in-house about how your system works.
- Conversions and other processes will benefit from the service provider's experience in using the software.
- You save yourself a good deal of personal effort.

Cons:

- Time-consuming communication to set up every new connection and process, and to make every change.
- You still need to do some of the work yourself in order to supply the service provider with the information they need.
- The service provider can't always respond straight away – you are dependent on their resources.
- You may need to give information to the service provider that you would prefer to keep in-house.

Summary:

If the system is already on-premise, you should use its location to have full control over everything. You can certainly outsource some jobs to service providers, as this may reduce the initial training costs and deliver positive results more quickly. However, you shouldn't expect to be able to simply give the service provider a few keywords and sit back and look forward to the final result. You may be able to save your employees' time, but some effort will still be required.

 **SAAS**

SaaS stands for 'software as a service'. This usually means that a piece of software, which is centrally hosted by a provider, is 'purchased' by way of a subscription. It is accessed via the internet, e.g. via a browser or dedicated clients.

So what does this service look like in reality?

The EDI system is installed somewhere in a data centre – theoretically on the other side of the world. The data centre operator ensures that the system is highly available, takes care of data backups, installs updates and bug fixes, and – of course – provides the entire

communications infrastructure. You make use of all this from your business premises via the Internet, a browser or a special client. You can set up communications channels from the data centre to both your partners and yourself, since the data need to be able to travel back and forth between your systems and the EDI system in the data centre. You can also map out data conversions and process chains exactly as you need to – this is completely up to you. And of course, you can keep an eye on whether all processes are running smoothly. So far, so good. It means you save money on hardware and infrastructure, and the entire system administration is also left up to the operator.

And because the system isn't installed on your premises, you just pay for usage rather than for an entire system up-front. Various pricing structures are available. You can be charged per active interface, by the number of messages/conversions, by volume of data, and so on. This naturally saves you having to make substantial initial investments.

Yet the concept quickly reaches its limits once your requirements go beyond simple format conversions and communication. For instance, things get complicated if you enrich data – e.g. by obtaining name and address data to go with a customer number. This is because data of this kind are stored in databases and in other systems on your own network, and not in the data centre. Of course, you could build a VPN link to allow the data centre to access your network, but that will have a big impact on performance – not to mention the accompanying security issues. It's also an additional point of weakness, since while the data centre might be able to guarantee high availability, the VPN link can quickly collapse under the weight of DSL problems, for example.

If all you have are simply designed EDI processes – just data transport and conversion – then SaaS can be a very good option. It may also be possible to tack on simple enrichment and implementation tasks (e.g. country codes) relating to small files. Yet if a closer connection to the rest of your IT landscape is needed, SaaS will quickly get you into difficulties. If you need a single system that can also carry out EAI or ETL tasks, then you will have to turn to an in-house solution.

**Pros:**

- No need for either hardware or infrastructure – aside from an Internet connection.
- A data centre can guarantee you a whole other level of availability.
- You don't need to worry about purely administrative tasks, such as updates.
- There are no major initial investments involved.

Cons:

- You are dependent on the reliability of the operator.
- The rest of your IT landscape is inaccessible, or accessible only by making special arrangements.
- This means there is no chance of using the same system to also handle EAI or ETL matters.
- Business-critical data and processes are stored with a third-party.
- The running costs can quickly mount up to become more expensive than an in-house solution would have been, depending on the intensity of use and the tariff model.

Tip:

If you are currently only looking into EDI because your business partners have asked you to, and you don't have more than a handful of simple connections to make, then SaaS can look very enticing. However, if you can foresee that this whole area will grow for you over the next few years (and there is a very good chance that it will), then you should look for a provider who will let you adopt both options. You could start with SaaS (or even EDIaaS), and then simply move the interfaces to your own premises when the time comes for an in-house installation. If you are using the same software then a transfer of this kind shouldn't prove too difficult.

 **EDIAAS**

When you use EDIaaS, you have very, very little to do with the entire EDI process, since EDI as a Service means that everything is handled by a service provider – from operating the hardware the configuring the communication paths and data conversions, and even monitoring, too.

Of course, that all sounds amazingly convenient, as your work then essentially consists of passing data intended for your partners to the service provider, waiting for them to supply the data to the recipients in the right format and subsequently having them return your partners' processed data to you. All your systems need to do is generate the right data and input it again.

Yet in practice, your work doesn't really consist of just these few steps, briefly outlined above. Although your service provider may fine-tune your communications with your partners for you, and may even take charge of finding out how accurate their data need to be – yet you will still need to explain to the provider how your own data are structured and exactly how you want them to process your partners' data for you. That said, of all the available solutions, this is definitely the one that involves the least amount of effort on your part.

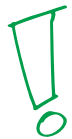
What's more, EDIaaS also leaves you with virtually no control over what happens to your business-critical data. At best, you will be able to check whether your provider thinks everything is running smoothly, or if any processes have crashed. In the worst case scenario, you will only find out about any problems when your partners start complaining about unclean data or data that fails to arrive altogether, or when you find data going missing yourself.

On top of that, every time you want to make a new connection or even the smallest change, you will need to submit an order to your service provider, make an additional payment for the service, and wait until the provider has enough available manpower to fulfil your order.

And last but not least, there is the disadvantage we encountered with SaaS that your systems are inaccessible, or accessible only under very restricted conditions. What if you just want to quickly retrieve the address data for a particular customer number from your database? No can do.

Pros:

- No in-house hardware or infrastructure required – instead, you get the reliability of a data centre.
- Service providers generally offer a wide range of possible communication paths.
- The service provider has a lot of experience in dealing with software and data formats, as they work with these on a daily basis.
- There are no major initial investments involved.
- Relatively little work required on your part.



Cons:

- No control whatsoever over business-critical data and processes.
- Complete dependence on the service provider (trustworthiness, human resources, etc.).
- The rest of your IT landscape is inaccessible, or accessible only by making special arrangements.
- That means there is no chance of using the same system to handle EAI or ETL matters.
- Depending on the usage intensity and the pricing structure, the running costs can quickly mount up to be more expensive than if you were using an in-house solution.

Conclusion:

Even more so than SaaS, EDIaaS is suitable only for a handful of very simple EDI scenarios. When things get significantly more complex, this solution becomes ineffective. And it also leaves you with less control over your own data and business-critical processes than was the case with SaaS. The same advice applies here: If you want to use this method as a starting point for EDI, you should choose your provider intelligently so as to give yourself the option of switching to a more flexible alternative at a later date without having to completely reconfigure everything.

→ **OPEN INTERNET VS. VAN**

EDI communication is the alpha and omega – quite literally! After all, the beginning and the ending – the alpha and omega – of every EDI process consists of communication. Both your data and the customers somehow need to get into the EDI system, and then come back to the customer or to you afterwards. Whereas this was previously done by fax, through the post or over the phone, nowadays the Internet offers a whole host of protocols for communication and remote data transmission. And new ones are being developed all the time. Here is a short list of the most common:

- **FTP(S)**: File Transfer Protocol (with optional SSL encryption).
- **SMTP/POP3/IMAP**: Classic email.
- **SFTP and SCP**: Secure FTP und Secure Copy – both methods of transfer involving SSH tunnels.
- **HTTP(S)**: The well-known Hypertext Transfer Protocol (with optional SSL encryption).
- **OFTP**: Odette File Transfer Protocol. Only its name has anything to do with the widely used FTP. This was actually originally a temporary solution that has remained with us, and is already in its second version.

- **AS2:** Applicability Statement 2. The actual data transfer itself takes place via HTTP(S), but comes with a wide range of security layers, including multiple options for encrypting and signing the data. The sender also receives an immediate notification to tell them whether their data were received cleanly.
- **WebDAV:** Web-based Distributed Authoring and Versioning – an extension of HTTP that offers more possibilities for data transfer (entire directories), as well as user rights and versioning.

We will describe all of these communication paths in more detail later on.

There are also other ways to exchange data with partners beyond the general Internet, and these (and a few other) EDI communication options. For example, OFTP initially only worked via direct ISDN connections between the communication partners – which is naturally more secure than a normal FTP transfer or simply sending emails over the Internet. One service that provides (mostly) secure transfers – often coupled with a few additional services – is the Value Added Network, or VAN for short.

This concept comes from the pre-Internet era, when it was used to achieve two key objectives:

- VAN customers didn't need to negotiate their own communications with every single business partner, and if their partners were connected to the same VAN, they could use the VAN to hand the data over directly along with instructions for where the data needed to go. In the ideal scenario (i.e. every partner using the same VAN), only one connection was needed to get through to everybody.
- And because a VAN of this kind wasn't publicly accessible (unlike the Internet today), the operator could also make sure that nobody was secretly listening in or disrupting traffic. In other words, they could guarantee secure transfers.

In certain respects, these points are still valid today – although nowadays, the value is added through processes such as data conversion, which means that VANs tend to behave a little like EDIaaS services. The question here is: What should you use when you want to set up an EDI system? Should you connect to a VAN, or simply use the Internet, since you have access to anyway? Of course, this question only comes up if you want to operate an EDI system of

your own (including one that is hosted under SaaS) instead of just using EDIaaS. Let's take another look at the two main advantages of the VAN:

One connection for everybody

This might sound fantastic at first, but it's only an ideal scenario. Because for this to be possible, every single communication partner needs to be connected to the same VAN – which is a rather unlikely proposition. Of course, it's possible that the VAN might offer you communication with 'the outside world' – but then you're back on the normal Internet, where advantage 2 no longer applies. Configuring the various communication channels shouldn't prove too difficult with decent EDI software, since we've come a long way in that regard over the last few years.

Closed user group & secure transfer

Leaving aside the fact that there is no such thing as an unhackable network, there is now a huge range of widely available standards specifically for secure transfers. Whether HTTPS, FTPS, SFTP/SCP or AS2, encryption is used everywhere. With AS2, there are many different ways for you to work with encryption and data signatures, and receipt confirmations are mandatory here too. So if you use modern transfer methods, you'll find that this benefit has long since been matched online.

In other words, as the Internet evolved, it largely caught up with the advantages VANs used to enjoy. The disadvantage remains unchanged, however: it costs more money. Whether you're charged monthly or annual fees or an additional sum for each file you send or receive, you will be paying extra for a service that you could just as well obtain via another method.

As mentioned above, we are discussing a situation where the extra work involved in EDI (e.g. conversion, reconciliation, enrichment, and so on) is already covered by a specialist system. If all you need is simple EDIaaS, the offer of a suitable VAN may be sufficient for your needs (although in this day and age, it is barely imaginable that anyone would willingly restrict themselves solely to communication partners who are connected to the same system...).

 **ESB****IF ONLY WE COULD SAY FOR CERTAIN...**

So what actually is an ESB? Some people call their software an ESB; others use it as a name for certain infrastructure that they have built in their company (or in their department); and still others think of it as an architecture concept or a particular technology. It's all extremely vague.

The term was probably invented to provide a (more or less) simple way of referring to various pre-existing software products and their functions. But that doesn't help us much. What does an ESB actually do?

EVERYTHING TO DO WITH COMMUNICATION

The ESB facilitates communication between different systems without mapping out any complex logic in the process. That's what makes it different from EAI and EDI. It understands the languages (i.e. data formats and interfaces) of all the connected systems, and it can transfer information from one system to another – but that's about the extent of its functionality if it is limited solely to the tasks of a true ESB.

In other words, a system can hand a CSV file over to the ESB, which the ESB then passes on and also converts into an XML format (for example) that the target system will understand. Yet business logic is a foreign concept to it. It offers a good way of simplifying communications between different systems. There is no need to teach every single system how to communicate with other systems (including conversion of data formats); instead, you can manage all this centrally with the ESB using an adapter.

That means the ESB only needs to know how to convert a CSV into a particular XML format, irrespective of how many systems will then use those formats to communicate.

This all sounds very similar to the description of EAI above. And indeed, we can view the work of an ESB as part of the puzzle pieces that make up EAI. The difference is that EAI also maps out and implements the logic and interdependencies in the business processes, whereas the ESB solely deals with translation and transmission. At any rate, the whole topic is genuinely very vague – and some experts even doubt that there is any reason to use the term at all – so we won't spend any more time on it.

 **ETL/ELT****WHAT DOES ETL MEAN?**

ETL stands for Extract, Transform, Load, and means reading (or extracting) data from one or more data sources before processing (or transforming) them and then loading them into a sink – normally a database. The sources are also typically databases in the majority of cases. ELT does the same thing, but in a slightly different order.

With ELT, we are usually dealing with mass data – so, for example, the task might be to regularly push the inventory in the ERP system into the database of the web shop. This can quickly add up to several thousand records – or even several tens of thousands. And these massive volumes of data pose a special challenge to an ETL tool.

Incidentally, we are talking specifically about ETL tools here, since we shouldn't waste any time on custom-developed programs. Even if these are tailored precisely to perform a specific task or to be ever so slightly faster, standard professional tools have so many other advantages (flexible configurations, monitoring, faster implementation of new tasks) that the DIY method simply isn't worthwhile. The same goes for all the other concepts we have discussed, by the way – EDI, EAI, and so on.

 **WHAT COUNTS WHEN IT COMES TO ETL?**

Speed, speed, and did we mention speed? No in all seriousness: of course there are other important factors at play, but don't forget that we are dealing with mass data. To explain this, let's take another look at the example of transferring the stock list from the ERP system into the web shop database.

In the simplest case, this can be done by simply reading the entire ERP database and pushing it into that of the web shop. Yet the tables at least will probably look different, so we will need to modify the structure of the data slightly. Then the ERP system might have special codes for the different countries, their respective tax rates, and so on, so the web shop will need country codes such as DE, AT, CH as well as direct percentage values for tax. Those are just a few simple examples of possible transformations.

Now we have a problem: we want to have a clean data set – or in other words, a snapshot of the stock list in the ERP where everything matches up precisely. That means we can't simply fire off a select statement to retrieve a few tens or hundreds of thousands of items from the source database over the course of a few minutes, since that database may be constantly changing. The result would be inconsistent data. A simple solution to this would be to run our process in the middle of the night when the inventory won't change. But the problem there would be that the data behind the web shop would become less accurate as the day went on. So maybe we should make the reconciliation multiple times per day instead – perhaps every three hours. Yet that would mean we wouldn't be able to make any changes to the ERP system while it was being read, as the tables would be locked. That wouldn't be ideal either, if it took a few minutes and happened multiple times per day. This is why our motto here is 'need for speed'!

We also have the same problem in the target database. If we replace the entire database each time, the tables will have to be emptied and refilled for each reconciliation, leaving the web shop unusable while the process is running.

There are a few different possible approaches here. For example, we could use a time stamp for the most recent change to ensure that we only retrieve records from the source that have changed since the last reconciliation. That would reduce the quantity of data, but could also slow down the reading process due to the comparison contained in the where-clause in the statement. Alternatively, we could tell the ERP system to track what has changed since the last transfer and to automatically make these data available at regular intervals. Finally, if the data in both databases need to be synchronised at all times, we could immediately hand every change over to the ETL tool, which would then pass it on to the web shop database. That would spare lengthy shutdowns on both sides, but would result in more work. For instance, if an item changed multiple times between two full reconciliations, it would only be transferred once; whereas with instant handovers, each change would be passed on every single time it happened.

We should make sure we don't forget about the 'T' part of the process – the transformation. Speed plays a role here too, albeit a less important one than when we are reading and writing the source and target databases. After all, we aren't blocking any other systems

during this phase. Yet it also shouldn't take a whole hour to process hundreds of thousands of records, since otherwise the data will be obsolete by the time they reach their destination.

In the simplest case, as mentioned above, a transformation can involve replacing country or tax codes. Yet it might also involve eliminating duplicates or otherwise cleaning and enriching the data. As such, we often need to access files with conversion tables, additional databases and other third-party systems. We may also need to call up web services or other SOA services. As you can see, there are similarities here with EAI processes.

This brings us to a very important point. As mentioned above, specialist ELT tools are generally better than generic branded solutions (and for more reasons and examples of this kind of software, head over to Wikipedia). However, as this kind of process often involves steps that appear in EAI (and EDI) processes, it is worth considering whether you really need to buy a dedicated ETL software, or whether it would be better to acquire an EAI tool that can carry out the same tasks. Although it certainly won't reach the extreme speeds of specialist ETL tools, but the performance of the EAI software might well be more than enough for your requirements, and you can then go on to use the system for other purposes too.

Not to forget that you will be able to keep all your important processes in one central location and monitor them too, instead of staying on top of two different systems. Or on three, if you also want to deploy an EDI system – which is unavoidable for almost every company nowadays. In other words, the ideal solution in many cases is a single system that is at home in all three worlds and can meet your requirements across the board – even if it may not be able to produce the same quality of results in extreme situations as specialist tools such as ETL (and SOA-type functionality might come into play here too).

 **SOA****FOCUS ON SERVICE**

SOA stands for Service-Oriented Architecture. In other words, an architecture of software products or IT systems designed specifically to provide services. This all sounds very nice, but what kinds of services are we talking about here?

In fact, it could be any type of service. The main thing is that every system involved – whether on the intranet or the Internet – offers an interface via which users can access its services. There are no general rules governing exactly how the service is accessed, but the way in which each individual system communicates needs to be clearly defined. And of course, the great thing about SOA is that there can be a service that exists solely to call up multiple other services – irrespective of whether those services are made available by the same system, or by a completely different company on the other side of the world.

Examples:

In principle, a service might simply perform a particular calculation – e.g. adding together $a+b$, to take the most basic example. We feed in the two values and the service returns the total. A sensible example would be a service that calculates the distance and remaining travel time between two postcodes. Another service might deliver the geo-coordinates for a particular address. Yet another could also take an address as an input and run a credit check based on the surrounding neighbourhood.

These are examples of what would be public services – but services of this kind can naturally also be deployed within a company to check the stock levels of a particular item (for example). Then we would no longer need to know which database we needed to query – we could simply call up the relevant service and obtain the inventory data.

When working with several such services, we can also bolt a new, more complex service on top that calls up a whole sequence of individual services and delivers a final result at the end. This gives rise to a whole network of services in which complex services call up other, simpler ones, and are themselves used by even more complex services. This network can be distributed across the entire world, all while end users entering a query somewhere on their monitor remain oblivious.



Requirements and benefits:

A service that is part of an SOA is subject to certain requirements. It must:

- provide a certain level of functionality
- be accessible externally (i.e. via a network)
- be usable as an independent unit – or in other words, it needs to be stateless
- be accessible via a clearly defined interface that is independent of the type of user
- have a separate implementation that is encapsulated from the outside world, so that this plays no role in using the service
- and, at least in theory, it must meet a few other criteria that aren't necessarily important in practice.

These requirements – especially the encapsulation and the clearly defined interfaces – offer a few benefits. Let's assume (for example) that there is a service within the company which returns the stock level for a particular item number via a web service interface (one of the many ways in which SOA services can be called up). At the moment, we simply have a small wrapper around a database query. It takes the item number as an input, runs a select statement on a particular database, and then returns the inventory value. Now we have a change to the warehouse management process – SAP has been introduced, so we can't simply look at the tables anymore. Instead, we are now able to go through the RFC interface on SAP and find the inventory value for the item number.

If we formerly connected directly to the database and fired off a statement every time we ran a query of this kind, there would now be a great deal of work involved in converting everything to SAP/RFC. Instead, all we need to do is to reimplement the service. The code gets completely replaced, but the external interface remains unchanged. And lo and behold, by making a single centralised change that nobody would even notice, we can neatly switch all our queries from the old warehouse management system to the new SAP system.

That's exactly how it looks from the perspective of the user of the service. If the it was previously accessed using old, cobbled-together C++ programs but the company switches to a modern standardised software package, that package only needs to be capable of calling up a web service in order for the user to continue using the same interface as before. Because services in an SOA typically use standard protocols such as SOAP, they can be accessed and used via a wide range of software products.

DATA FORMATS & MESSAGING STANDARDS.

Introduction	46
UN/EDIFACT	51
FIXED RECORD	65
VDA	68
Fortras	72
XML	75
openTRANS	86
BMEcat	88
CSV	92
IDoc	96
ANSI ASC X12	101
ENGDAT	105
Miscellaneous	108

DATA FORMATS & MESSAGING STANDARDS

→ INTRODUCTION

BASIC INFORMATION ABOUT DATA FORMATS AND MESSAGING STANDARDS

When we talk about data formats, we are referring to the lowest level of the file structure. In other words, we are looking at the question of how individual values are represented in a file.

There are a few basic types of data formats here:

CSV: Comma-separated values.

Fixed record: Each value has a precise number of characters, allowing it to be located precisely by virtue of its position within the file.

XML: Extensible markup language; a format in which the values (and even groups of values) are given their own names, tags and attributes.

There are also more complex variations on these formats – e.g. the values in EDIFACT and X12 messages are separated by several, different characters, while the BWA format used in the book trade is based on a mixture of fixed record and CSV. Separating the individual values is just one aspect of the issue, however, since for each more complex format, the values are arranged into logical groups. With CSV and fixed record, these groups are typically called records, while EDIFACT or X12 refers to segments and loops or segment groups for next-level groupings. XML calls them complex elements.

Simple examples:

CSV:

```
OH;4711;K0815;20210530  
LIN;1;S123;5;9.99  
LIN;2;H456;3;17.95
```



Fixed record:

```
OH 4711 K0815 20210530  
LIN0001S123 0005000009990  
LIN0002H456 0003000017950
```


XML:

```
<Order>
  <Header>
    <OrderNo>4711</OrderNo>
    <CustomerNo>K0815</CustomerNo>
    <OrderDate>2021-05-30T00:00:00+2</OrderDate>
  </Header>
  <LineItems>
    <LineItem No="1">
      <ItemNo>S123</ItemNo>
      <Quantity>5</Quantity>
      <PricePerUnit>9.99</PricePerUnit>
    </LineItem>
    <LineItem No="2">
      <ItemNo>H456</ItemNo>
      <Quantity>3</Quantity>
      <PricePerUnit>17.95</PricePerUnit>
    </LineItem>
  </LineItems>
</Order>
```



Here, we see the same information three times, in three different formats. As you can see, the order header has the identifier “OH” in CSV, while the line items have the identifier “LIN”. After all, we need some way of identifying the information involved. XML is clearly much more detailed and significantly easier for humans to read – but it also takes up a lot more space. Yet with a decent compression algorithm, this isn’t a serious problem – at least during transfer.

We will look at these formats in more detail over the following pages, but these basic examples will suffice for now.

INTRODUCTION

At its most basic, a messaging standard can simply define more complex data formats – but it can also extend as far as laying down procedural rules concerning transfer paths (for example). Examples of standards that concentrate on the data format (or in other words, on defining particular message types and their exact structure) are EDIFACT and BMEcat. The latter of these in turn uses XML as a data format.

Conversely, ENGDAT is a highly complex industrial standard that contains data formats, but also defines the content of the files (e.g. the metadata – cf. the article on ENGDAT) and sets out rules concerning transfers and even file names.

Example of EDIFACT format:

```
UNA:+.? '
UNB+UNOC:3+Sender ILN+Recipient ILN+210230:1025+1++98765'
UNH+1+ORDERS:D:96A:UN'
BGM+220+9'
DTM+4:20210230:102'
NAD+SU+++Hardware Provider+1 Sample Street+Nowhere+NRW+54321+DE'
NAD+BY+++Lobster: DATA GmbH+Hindenburgstr. 15+Pöcking+BAY+82343+DE'
LIN+1++4711:SA'
IMD+++::USB Stick'
QTY+1:100'
UNS+S'
CNT+2:1'
UNT+10+1'
UNZ+1+98765'
```



In this example, the company Lobster (the identifier BY stands for buyer) ordered 100 USB sticks (item number 4711) from the company Hardware Provider (SU = supplier) on 30 (sic!) February 2021.

As an aside: In 2013 in Fulda, Germany, an individual really was arrested for having multiple fake IDs on which they claimed that their birthday was the February 30.

The example above shows an order ORDERS in EDIFACT format version D 96 A, the earliest of two versions, which was issued in 1996. As the managing body for EDIFACT, the UN Economic Commission for Europe (UNECE) typically issues two versions A and B each year, and sometimes even a third (which is called – surprise, surprise – C).

Generally speaking, not every message changes in each new version, and the changes are often backwards-compatible. The defined message types cover almost everything you will ever need in electronic data interchange – from AUTHOR (authorisation message) to WKGRR (work grant request message). We will go into more detail in the chapter on EDIFACT.

THE ENG DAT PROCEDURE

ENG DAT is a defined workflow for exchanging technical documents, used primarily in the automotive sector. The files involved are generally CAD files, so it is impossible to integrate information about the people involved possible – unlike an EDIFACT message. Instead, the meta-information is packaged in separate files and sent alongside the CAD file. This even includes address and contact information. These files are called ENGPART messages. EDIFACT and XML can both be used as the format for all this additional information, although XML is the more legible option (just compare the examples above!) and the one used in the latest versions ENG DAT v3, ENGPART v4. Transfers take place via OFTP. This is not mandatory, but it has become standard in practice.

OTHER MESSAGING STANDARDS

X12 works in a similar way to EDIFACT in that it provides its own unique format as a more complicated form of CSV with multiple separating characters, each with a different meaning – see above. This standard is a precursor to EDIFACT and is still in use, primarily abroad. And much like EDIFACT, it also defines large numbers of different message types.

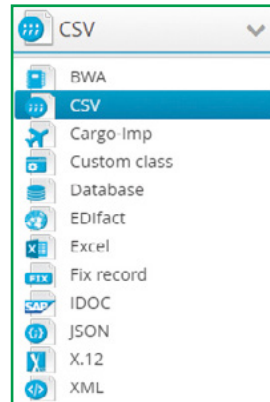
By contrast, Fortras and older VDA messages use a pure fixed record format, though the number of individual messages is restricted. You can find out more about these standards in their respective articles.

INTRODUCTION

CONCLUSION:

As you can see, an EDI program needs to be able to deal with many different messaging standards and data formats. The bigger the selection it can handle, the better. Even if you only have simple CSV files in your queue, this can quickly change, for example when dealing with new business partners. As such, an EDI system should offer the widest possible selection of formats.

By way of example, here is the list of options included with the data management software Lobster_data:



As we have already said, behind CSV, XML and fixed record there lies a whole host of different formats and messaging standards, all of which rely on one of these basic formats. And yes, Excel is (unfortunately) still also frequently misused in data interchange. Incidentally, SAP IDocs are also fixed record files in their older form, though they have a very special structure, which is why we have treated them separately. More recently, these documents have also become available in XML format. You can find out more in the article on IDoc.

One good solution for integrating all these data formats is Lobster_data (see www.lobster-world.com/en/lobster-data).

→ UN/EDIFACT

WHAT DOES IT STAND FOR?

EDIFACT stands for Electronic Data Interchange for Administration, Commerce and Transport. Because this messaging standard was developed by a division of the United Nations – the UNECE (United Nations Economic Commission for Europe) – and is maintained by its CEFAC department, this standard is often referred to as UN/EDIFACT. And because the final E in UNECE stands for Europe, organisations overseas (in the USA, Australia etc.) prefer to use X12, which is similar in principle, but fundamentally different. You'll find more information on X12 in a separate section.

EDIFACT is a data format, i.e. a method of constructing messages. And there are an awful lot of types of message: Version D12B defines almost 200 different message types. What do we mean by version D12B? D12B stands for the Directory of message types as of the year 2012, edition 2. Generally speaking, two editions, A and B, are issued each year in which new messages are added and existing ones are modified (which usually means expanded). In 2001, however, there was also a third edition (called C). Conveniently, the message specifications in each version have been made generally available for download from the UNECE website. Fine, so they come in a dreadful text format, but all the same, they're free of charge.

ALTERNATIVES – AKA SUBSETS

The UNECE issues the standard for EDIFACT, in which it defines messages for all kinds of different sectors and fields of application. That's all well and good, but – as is often the case – everyone just wants to do their own thing. They do so by means of subsets, which draw on the same elements as EDIFACT, but focus exclusively on those that are relevant to particular industries or user groups. The selection of message types used – as well as the structure of the messages – is stripped back to what specific users consider to be sufficient for their needs. Examples of these subsets include EANCOM (used in the consumer goods industry), EDIFOR (haulage), EDIWHEEL (the tyre industry) and Odette (created by the trade association of the same name and used in the automotive sector). At least in theory, the messages in these subsets should be structured in such a way that they can just as easily take their place within the structures of the UN standard. In other words, segments, segment groups and even fields can be omitted – but only if they are not mandatory within the standard. That said, it is possible to define your own qualifiers. But more on that later.

GENERAL STRUCTURE

How are EDIFACT messages structured? Here is an example:

```

UNA:+.? '
UNB+UNOC:3+Sender ILN+Recipient ILN+210230:1025+98765'
UNH+1+ORDERS:D:96A :UN'
BGM+220+9'
DTM+4:20210230:102'
NAD+SU+++Hardware Provider+1 Sample Street+Nowhere+NRW+54321+DE'
NAD+BY+++Lobster: DATA GmbH+Hindenburgstr. 15+Pöcking+BAY+82343+DE'
LIN+1++4711:SA'
IMD+++::USB Stick'
QTY+1:100'
UNS+S'
CNT+2:1'
UNT+11+1'
UNZ+1+98765'

```

**THE BOTTOM LEVEL**

Let's unpick the whole thing from the bottom up. Here, you can see individual lines, each of which contains a "segment" – though the lines only serve to aid legibility, and are not compulsory. The entire message could just as easily be written in a single line, or in blocks of text 80 characters wide (which is very popular among AS/400 or IBM iSeries users). In that case, the message would appear as follows:

```

UNA:+.? 'UNB+UNOC:3+Sender ILN+Recipient ILN+210230:1025+98765'UNH+1+ORDERS:D
:96A:UN'BGM+220+9'DTM+4:20210230:102'NAD+SU+++Hardware Provider+1 Sample Street+N
owhere+NRW+54321+DE'NAD+BY+++Lobster: DATA GmbH+Hindenburgstr. 15+Pöcking+BAY+82343+D
E'LIN+1++4711:SA'IMD+++::USB'QTY+1:100'UNS+S'CNT+2:1'UNT+11+1'UNZ+1+98765'

```

Looks ugly, doesn't it? But it's just as valid. This is because an apostrophe (') appears at the end of each segment, which is an agreed end-marker (and part of the standard). That makes line breaks completely unnecessary. As you can see from the example above, it's even possible to split values. It makes no difference at all. Any software capable of reading EDIFACT will simply put the values back together again (or at least, it should...).



So what is a segment? Each segment consists of a segment identifier (UNA, UNB, UNH, and so on) and values. The identifier is always made up of three letters, and tells you what the value is. For example, QTY stands for **QUANTITY**, while NAD is short for **NAME AND ADDRESS**. Segment identifiers beginning with UN have a special meaning, and do not depend on the message type. The most important of these general segments are: UNA, UNB, UNH, UNT and UNZ. We will explain these individually later on.

The values within a segment are separated by two different characters; in the example, these are the plus sign and the colon. Why two different characters? Because segments contain not only fields, but also things called **Composite elements**, or **Composites** for short. These are made up of multiple fields that belong together logically. For example, in the UNB segment, the date and the time of the message are combined into a composite: 210230:1025 - or February 30, 2021 (yes, that isn't a typo) at 10.25am. Standalone fields are separated from each other and from the composites using a plus sign, while fields within a composite are separated with a colon. If a field has no value, but is followed by another field or composite, then the necessary separators are included in order to preserve the order. A good example of this can be seen above in the IMD segment: IMD+++::USB Stick. Here, only the third field of the composite contains a value, and the two fields before the composite are also empty. If nothing else appears in the segment or composite after a particular value, then the remaining separators can be omitted. An example of this would be:
NAD+SU+12345'

If one of the separators itself happens to appear within a value, it needs to be escaped in order to pass as a normal character. In the standard, this is done with the help of a question mark, so that a company called "Reibach + Sons" would become Reibach ?+ Sons. And if the question mark itself appears within a value, it is also escaped using itself: "Understood??"

So much for the lowest level of the data format.

THE GENERAL STRUCTURE OF AN EDIFACT FILE

Let's now go back to the very top and look at the structure of each EDIFACT file. Certain segments will appear in every file, regardless of what that file relates to. These general segments are also known as service segments, and they form the wrapper for every EDIFACT file or message:

UNA (optional; in case other special characters are used than those specified in the standard – see below)

UNB (mandatory; appears once at the beginning of the file)

UNG (optional; wraps around a group containing multiple messages of the same type)

UNH (mandatory; once per message, n times within a group or file; header information)

Within an individual message

UNT (mandatory; end segment to UNH, therefore used once per UNH)

UNE (optional; end segment to UNG, therefore used once per UNG and mandatory whenever UNG is used)

UNZ (mandatory; end segment to UNB, appears just once at the end of the file)

STRUCTURE OF INDIVIDUAL MESSAGES

And now we come to the structure of an individual message. Unfortunately, this isn't entirely straightforward.

The message type determines which segments are used and what order they appear in; however, the same kind of segment can appear in multiple places within a single message, and can take on different meaning depending on the context. As an example of this, we will look at the APERAK message in version D96A. APERAK stands for 'Application Error and Acknowledgement'. What can we say? It isn't always easy when you have to reduce message names to exactly six characters...



Structure of the APERAK EDIFACT message, version D96A, in Lobster_data

We have chosen this message purely because it has a very simple structure that fits into one small diagram. It's used to tell the sender of the data if there are any problems with the content or technical information in their file, or to confirm that everything is OK.

You can see the service segments UNB, UNH, UNT and UNZ (UNG/UNE aren't really relevant to the content and have therefore been omitted), as well as the actual message between those segments (from BGM onwards). BGM stands for 'Beginning of Message' and is mandatory in every message. The red star in the box flags it as a mandatory segment. Optional segments or segment groups (we'll get to those soon) don't have a red star.

We just mentioned ‘segment groups’. The image shows SG1, SG2, SG3, and inside SG3, SG4 (SG standing for segment group). The tree structure lets you see clearly which group contains which segments or sub-groups. Much like an individual segment, a segment group of this kind can, under certain circumstances, appear multiple times within a file. In fact, this is always the case for segment groups – otherwise there wouldn’t be any need for them. In the picture you can see from the green one which segment appear only once and which segments or groups can appear multiple times in the file (no green one). What makes EDIFACT somewhat complicated is the rule stipulating that the individual segments can appear in particular positions in each file. Let’s take the RFF segment as an example. This contains references, e.g. to other documents. In principle, an RFF can appear immediately after the BGM; this is the RFF that appears in SG1. If this is followed by an NAD then we know that we have moved on to SG2. However, if another RFF appears after that, it can’t be the one in SG1 – rather, it has to be the one in SG4. And that in turn means we need another ERC between the NAD and the RFF, as otherwise the file will be broken. Complicated, isn’t it? Let’s take another specific example, but strip it back to the segment identifiers and add a few new segments:

UNB

UNH

BGM

DTM Global date and time information (DTM = Date and Time) for this message

RFF This is the RFF in SG1, containing references from this message to other documents (e.g. an order number)

DTM The DTM in SG1, e.g. date and time information from the external document to which the reference is made (an order placed on...)

RFF The RFF in SG1 once again – in other words, SG1 is repeated

DTM The DTM in the second iteration of SG1

NAD With that, we’ve arrived at SG2, containing the address details and contact information

NAD And this is also repeated, albeit only the NAD segment

ERC Now we’re in SG3 (error information)

FTX Free text containing information about the error



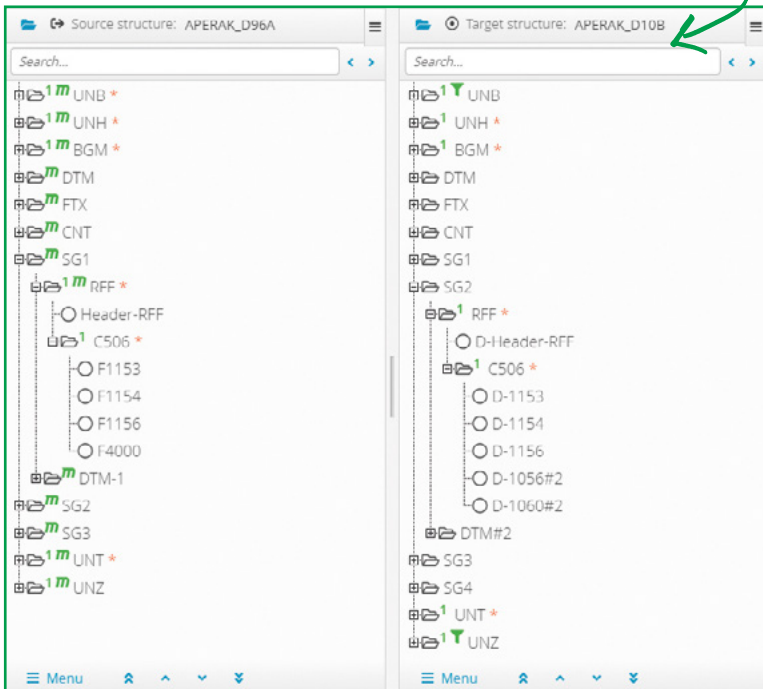
RFF And this RFF is now the one in SG4
FTX This free text belongs to the reference in SG4
FTX And so does this one
UNT
UNZ

If we took out the ERC segment, then none of the segments after it would be able to appear either – we would skip straight to the UNT. We can see this in the screenshot too: SG3 is optional, but if it is used, then the ERC segment has to appear too. As a general rule, the first segment in each segment group is mandatory when the group is being used. The reason for this is simple, because without an ERC, no one would know that SG3 was up next.

CONSISTENCY WITHIN VERSIONS AND DEVELOPMENTS BETWEEN THEM

Although EDIFACT files have a very complicated structure, there is some good news: the individual elements (the segments and composites) are always built in the same way. In other words, an RFF segment will always look the same, irrespective of what type of message it appears in. And even a field with a specific number (such as 1153 – fields are numbered in EDIFACT) will always have the same type, the same maximum length and the same meaning. Of course, the structure of segments and composites is expanded from version to version, and a field can e.g. become longer in the process for example. However, within a single version, an RFF or an NAD will always look the same. Even a C506 used in multiple segments will look the same everywhere it appears.

Here is a direct comparison between the APERAK message in versions D96A and D10B, each showing the structure of the first RFF segment:



APERAK D96A and D10B and the differences between the RFF segments - as displayed in Lobster_data

As you can see, another segment group has now been shifted to before the references, which means that RFF now introduces SG2. In addition, field 4000 (reference version number) has been replaced by fields 1056 (version identifier) and 1060 (revision identifier). Incidentally, these also appear in the BGM segment and in the newly added DOC segment in the later version of SG1, and carry the same meaning, albeit in a different context. This explains the additional #2 numbering - a unique feature of this converter. The 'F' and 'D' letters that appear before the actual field names are also connected with this special software.



GENERAL SERVICE SEGMENTS

UNA: Defines the special characters

In exactly this order:

- 1 Separators between values within a composite.
- 2 Separators between standalone values and whole composites.
- 3 Decimal separators.
- 4 Escape characters for special characters that appear within values themselves (example: for a company called “Reibach + Sons”, a ? would need to appear before the +, giving Reibach ?+ Sons. Otherwise, the + would be interpreted as a separator).
- 5 In EDIFACT syntax V3, the space character was still reserved. With EDIFACT syntax V4, however, it can be replaced with a repetition character to signify a new feature, which fortunately is not being used.
- 6 The end-of-segment character.

The characters :+.? ' are standard; if these are used in a file then the UNA segment can be omitted.

UNB: Interchange header (file)

The most important global information about the file as a whole, such as the character set (see below), sender, recipient, date and message reference. In the example, all mandatory fields have been filled:

```
UNB+UNOC:3+Sender ILN+Recipient ILN+210230:1025+98765'
```

- 1 UNOC specifies the character set, or in other words, the characters that can be used in the document.
- 2 The 3 refers to EDIFACT syntax version 3, though there is already a version 4 by now (which we don't need to go into here).
- 3 Then come the identifiers for the sender and the recipient. These could also simply be company names.
- 4 Next, the date and time of the message.
- 5 At the end, we have a reference number for the transfer, which is also repeated in the UNZ segment. It uniquely identifies the file to the parties involved, so it should only be used once between those parties.

UNH: Individual message header

Serial number and basic information about the individual message.

```
UNH+1+ORDERS:D:96A:UN'
```

- 1 This is the first message in the file or group (the 1 is repeated in the corresponding UNT).
- 2 The message is an order (ORDERS)
- 3 The message is written in the format version D96A and complies with the UNECE standard rather than a subset such as EANCOM

UNT: End of individual message

The total number of all the segments in this message, including UNH and UNT as well as the serial number of the message that was given in the corresponding UNH.

UNZ: End of the entire file

The total number of messages in this file and the transfer reference number, which is also given in the UNB.

The totals and reference numbers in the UNT and UNZ segments are for monitoring purposes. They allow you to quickly spot whether individual segments or entire messages are missing, or have been incorrectly added by mistake.

BGM: Beginning of a message

Here once again, we find details about the message type (220 = order) as well as some additional information (e.g. whether the order should be withdrawn).

QUALIFIERS

There is one urgent topic that we still need to discuss: qualifiers, which are essential in EDIFACT. Firstly, they specify the subject of a particular segment (i.e. what the information contained in it describes); secondly, they determine the format in which particular values are presented; and thirdly, they set out special code lists, without which certain values are meaningless.

For an example of two of these qualifiers, let's look at the DTM segment:

DTM+4:20210230:102'

The first qualifier is the 4 in front of the date. It means that this segment relates to the date (and perhaps also the time stamp) of an order. The 102 at the end states that the value is given in CCYYMMDD format - i.e. four digits for the year (C stands for century) and two digits each for the month and the day.

The NAD segment also contains two examples of code list qualifiers. In field 3039, we enter an ID for the party involved - e.g. the ILN of the company's business partner.

And in the two subsequent fields 1131 and 3055, we can add code list qualifiers:

Source structure: ORDERS_D96A

X.12/EDifact definitions

Message type: EDifact X.12

Element	Description
1131	
1131	Code list qualifier
Value	Definition
107	Excise duty
108	Tariff schedule
109	Customs indicator
110	Customs special codes

Double click applies value as fix value in target field!

Close

Possible codes for field 1131

The screenshot shows a software interface for X.12/EDifact definitions. On the left, a tree view shows the source structure for 'ORDERS_D96A', with 'F3055-4' selected. The main window displays the definition for element 3055, which is 'Code list responsible agency, coded'. Below this, a table lists possible codes for the field:

Value	Definition
1	CCC (Customs Co-operation Council)
10	ODETTE
100	CH, Entreprise des PTT
101	CH, Carbura

At the bottom of the window, there is a note: 'Double click applies value as fix value in target field!' and a 'Close' button.

Possible codes for field 3055

Conversely, field 3055 once again contains a qualifier specifying the subject of the data. We can see this in the first example on page 52:

```
NAD+SU+++Hardware Provider+1 Sample Street+Nowhere+NRW+54321+DE'
NAD+BY+++Lobster: DATA GmbH+Hindenburgstr. 15+Pöcking+BAY+82343+DE'
```

SU stands for the **S**upplier (or to give it its official definition: “Party which provides service(s) and/or manufactures or otherwise has possession of goods, and consigns or makes them available in trade.”)

BY stands for **B**uyer.

OH, AND ONE MORE LITTLE THING: THE CHARACTER SET

EDIFACT files can also appear in various character sets. The character set specifies the characters that can be used within the file, and is included in the UNB segment.

The most important character sets are:

- UNOA: Only capital letters, numbers, spaces and a few special characters such as =.
- UNOB: The same as UNOA, but lower-case letters and a few more special characters are also allowed.
- UNOC: In principle, everything that can appear in the Latin1 character set, aka ISO-8859-1.
- UNOY: The character set that can be displayed with Unicode – so any character, essentially.

In addition, various Latin-X character sets such as Arabic and Hebrew are represented by UNOD through to UNOK.

So far, so clear. But there's a catch:

because a system processing an EDIFACT file won't necessarily know which character set to expect, it needs to read this information from the UNB segment. By the time you are using UNOY and receiving files in UTF16 – i.e. coded in 2 bytes – things get very tricky. For that reason, it has been specified that the UNA segment (if present) and the UNB segment must always be ASCII-coded, even when the rest of the message after the BGM is in Cyrillic or in UTF16. That's how it goes with globalisation: business is booming, but there are plenty of stumbling blocks too.

LEVEL OF FREEDOM

Last of all, we have one more important point to make:

EDIFACT offers an incredible level of freedom, from the qualifiers right up to the question of what information to package in which segment and where to place it in the message structure.

One example of the many available qualifiers is the NAD segment. We can use the qualifier BY for the buyer; however, we could also just as easily use BS (Bill and Ship to), or CN (Consignee), or probably even one of a few other options.

Things don't look much better for the supplier either. And as already noted in the article on in-house EDI systems, we can also position the full address information in a number of different places on the same message. The NAD segment can exist in multiple places within a single message type, most of which represent multiple levels (such as document, line item, detail, etc.), and there are an equal number of opinions out there regarding what addresses can or should be placed where. This often depends on the capabilities of the systems from which the data are derived. If a system only recognises address information at the level of the delivery call-off, then the addresses can only meaningfully be specified at the highest level.

On the other hand, if the system offers the option of adding a separate address to each line item, this option will presumably be taken up and represented through the issue of a DELFOR. As such, it is crucial that both of the parties involved in an EDI process come to an agreement on exactly how the relevant message format should be used. The lowest common denominator is precise information on how the data are prepared or expected, as otherwise it will be impossible to cleanly implement the process. In practice, however, parties are often forced to rely on a small amount of example data and a process of trial-and-error – which may be hugely time-consuming (and frustrating), but generally works out fine. In cases like these, it is important to work with a software package that is easy to operate and offers comprehensive testing facilities in order to support the user in setting up connections.

➔ FIXED RECORD

WHAT IS IT?

Fixed record refers to the structure of various data formats at the lowest level. Whereas CSV separates the individual values from each other using special separator characters, here it is the lengths of the values that are key. And – much like CSV – so is their order. As a result of these two requirements, each value in the record has a precisely fixed start and end position.

A simple example:

```
OH 4711 K0815 20210530
LIN0001S123 0005000009990
LIN0002H456 0003000017950
```



Yes, this may be a very straightforward example, but it covers the basics.

Let's go through each line in detail.

```
OH 4711 K0815 20210530
```

- **OH** stands for order header here. This is the record type identifier that allows the software (or a human) to recognise what kind of record is being transmitted. The identifier is always three characters long, and is completed, in this case, with a space.
- **4711** is the order number, which is eight characters long and completed with spaces.
- **K0815** is the customer number, which has the same length and is completed in the same way.
- **20210530** at the end is the order date, in `yyyymmdd` format (i.e. 8 characters long).
- Behind that comes another space, only you can't see it right now (though you will later).

```
LIN0001S123 0005000009990
```

- **LIN** is the line item identifier. Because it already takes up three characters, there is no need to add a space to it.
- **0001** is the item number. It has a length of four characters and is completed with leading zeroes.
- **S123** is the item number, followed by spaces to take it up to eight characters.
- **0005** is the quantity, which is four characters long, with zeroes added to take it up to the character limit.
- **000009990** represents 9.99, completed with filler zeroes before the value. Instead of using a decimal separator, it is simply specified that the last three digits represent the numbers after the decimal point. Total length 9.

Now, add the lengths of these different fields together. What do you get? 28, both times. To achieve this, a space is added to the end of the header line. If there were more than these two record types, they would all need to be long enough to make room for the longest one. Alternatively, you can just choose a standard computing number such as 64, 128 or 256. This is fairly typical for data formats of this kind, but not really compulsory. In principle, every record type can have its own unique length; however, that length would then need to be precisely defined.

By applying the rule that every record type must be the appropriate length, we can dispense with line separators (which are incompatible with many systems) and put everything together in one long sequence:

```
OH 4711 K0815 20210530 LIN0001S123 0005000009990LIN0002H456 0003000017950
```

Now you can see the space at the end of the order header.

RECORD TYPE IDENTIFIERS

An individual message is typically made up of various parts. These might include a header (as in the example above), and often a trailer record at the end, as well as information on individual shipments, call-offs etc., details on line items, and so on. In many formats, the record type appears right at the beginning. This is extremely practical, as we can then apply just one simple rule:

Look at the first characters of the file to find out what the record type is. With the help of the format definition, we now know how long this record type is and what values appear where. Then, we simply need to look at the following characters to find out what comes next.

RULES IN PRACTICE

All this talk of exact lengths for each record type is very nice in theory, but there are a few variations in practice. If we are taking a purist approach, then every record type really will need to be exactly the right length and have spaces added to fill up any unused characters. In addition, there is no line separation – everything is printed in one long sequence. That’s great for machines to read, but not particularly practical for humans (see above). As such, individual records are often separated from each other by line breaks in practice.

Yet that in turn isn't so great for the computer, since it will read the line breaks as extraneous characters and will need to take them into account when calculating the lengths of the records. In other words, a record type with 128 characters will in fact have 129 or 130 characters (LF or CRLF).

If we separate the record types with line breaks, we can also omit the filler characters at the end while we're at it. As a result, the header record above would suddenly be only 27 characters long, instead of the formally required 28 characters. This isn't exactly what we initially expected from this format, but you will encounter these variations time and time again out in the wild.

There are a few other rules that aren't necessarily compulsory, but are sometimes applied in recognised fixed-record data formats:

- Numerical values are generally brought up to their defined character length using leading zeroes.
- Non-numerical values, by contrast, are completed using spaces added to the right of the actual value.
- Floating-point numbers generally aren't represented using decimal separators; instead, we use a fixed number of implicit decimal places. For example, 3.05 (with leading filler characters) would become 00003050. The last three characters here are implicit decimal places. The number of decimal places in each case is usually specified individually for each value.
- Dates and times can also normally be written without a separator, and the formats `yymmdd` or `yyyymmdd` and `hhmm` or `hhmmss` are widely used instead.
- The record type identifiers generally appear at the beginning of the record – however, there are also formats (typically company-specific ones) that place the identifier somewhere in the middle of the data.
- Of course, if there is only one kind of record then we can dispense with identifiers altogether.

DATA FORMATS BASED ON FIXED RECORD

The two best-known exponents are Fortras and VDA (discussed here in their older formats). Both of these were developed and adopted by consortiums, and are discussed in their own dedicated articles. Then there is also the older IDoc format from SAP, which also forms the subject of its own section. In addition, exports from databases (for example) are also often structured in this way – especially when the data come from an AS/400 (IBM iSeries). Every column in a table then has its own defined length, and often everything is automatically written in one column in a single standard-length string. A few other programs also use files in fixed-record format for imports and exports, as they are very easy to read. You simply move the file pointer along by e.g. 128 characters each time in order to automatically reach the next data set. Within a data set, you can also jump to the positions specified at the beginning in order to find the desired values. Back when most programs still stored their data in custom structures on CDs (or even floppy disks) instead of using generally accessible databases (which weren't so readily available back then), this was the most practical solution for lacking storage space.

→ VDA

The acronym VDA doesn't actually stand for a data format – rather, it is short for the **V**erband der **d**eutschen **A**utomobilindustrie (the German Association of the Automotive Industry). This organisation has issued a set of recommendations to its members on how they should share their data with each other in the context of EDI. These recommendations come with numbers which stand for different message types.

The earlier recommendations still contain custom formats which – in terms of structure – are fixed-record formats. It is these that we will look at in this chapter. Since then, the VDA has been changing tack towards using EDIFACT (one example of this can be found in VDA recommendation 4938), which is gradually giving rise to a special subset (cf. the chapter on EDIFACT).

Message types

Common types or recommendations include 4905 (call-offs), 4906 (invoices), 4913 (delivery notes) and 4920 (forwarding instructions), but there are many more. Why do they all start with 49? Ask us an easier question! You can find a long list of available recommendations in the Wikipedia article on VDA.

As an example, here is the structure of the message type 4905 (call-off):

Structure of the VDA 4905 call-off displayed in Lobster_data

And this is what the format for delivery notes (4913) looks like:

Structure of the VDA 4913 delivery note displayed in Lobster_data

Don't be surprised at the different names for the record types - these can be freely chosen in this software and are something of a matter of taste. One person might be happy with "N[ode] 711", while somebody else might prefer the more detailed "R[ecord]T[ype]_511_Header_Call-Off Data".

Record structure

51101CuNo SuNo 0045600457210209 |END

Everything before “|END” is the file header for a VDA 4905 call-off. The individual values are:

- **511:** The header identifier – always three characters.
- **01:** The version number – always two characters, with a zero at the start as a filler character, if necessary.
- **CuNo:** Customer number – always nine characters, with spaces at the end as filler characters, if necessary.
- **SuNo:** Supplier number – likewise nine characters with spaces as fillers at the end.
- **00456:** The serial number of the preceding order – always five characters with leading zeroes as filler characters.
- **00457:** The serial number for this order – with the same length and filler characters as for the previous order.
- **210209:** The date of the transfer in yymmdd format – in this case, February 9, 2021.

Here, we have displayed the structure of this record in a more tabular format:

#	Name	Data type	Length	P..	Minimum	Maximum	Template	Description
1	D-Satzart_511	Integer	3	1	0	1		Konstant "711"
2	D-Versions_Nummer_511	String	2	4	0	1		Kennzeichnung der Aktualität einer Satzart
3	D-Kunden_Nummer_511	String	9	6	1	1		Ident-Nummer Lieferant> Kunde
4	D-Lieferanten_Nummer_511	String	9	15	1	1		Ident-Nummer Kunde >> Lieferant
5	D-Ueberr_IV_Alt_511	Integer	5	24	1	1		Uebertragungsnummer Alt
6	D-Ueberr_IV_Neu_511	Integer	5	29	1	1		Uebertragungsnummer Neu
7	D-Uebertragungs_Datum_511	Integer	6	34	1	1		Format JMMTTT
8	D-Datum_Multibestellung_511	Integer	6	40	0	1		Format JMMTTT
9	D-Leer_511	String	83	46	0	1		Mit Blank gefüllt

Structure of VDA record type 511 displayed in Lobster_data

Now, add the lengths of these different fields together. What do you get? 128. And every other record type in a VDA 4905 call-off will also be exactly 128 characters long.

In order to achieve this, a certain number of filler spaces are added to the end of each line.

Basic rules of VDA

Every VDA message begins with a header containing the parties involved, the numbers of the preceding and current transfers, and the date of the transfer. In most cases, a few other pieces of information will also be included, but this will vary from type to type.

Equally, every message will end with a trailer stating the number of individual record types within the message. Here is an example, taken from the VDA 4913 delivery note:

```

1m N719 *
○ D-Satzart719 *
○ D-Versionsnummer719 *
○ D-ZaehlerSA711 *
○ D-ZaehlerSA712 *
○ D-ZaehlerSA713 *
○ D-ZaehlerSA714 *
○ D-ZaehlerSA715 *
○ D-ZaehlerSA716 *
○ D-ZaehlerSA718 *
○ D-ZaehlerSA719 *
○ D-ZaehlerSA717 *
○ Leer-12
  
```

The trailer of the VDA 4913 delivery note - displayed in Lobster_data

It almost looks as though record type 717 appears later in the message, since its counter comes after that belonging to the trailer (719).

Incidentally, every record type starts with an identifier (three characters) and a version number (two characters, with a filler zero if required). This version number really does refer to the individual record type. As such, you will sometimes find all kinds of record types from different versions rubbing shoulders within a single message. Numerical values are generally brought up to the necessary length with leading zeroes, while all other values use trailing spaces as filler characters.

FIXED RECORD

→ FORTRAS

Today, Fortras is a messaging standard that is used for data interchange with or between haulage companies. Fortunately, it's relatively straightforward. There are three commonly used message types: consignment information, status reports and unloading reports. These in turn exist in multiple versions, or releases. Releases 2 to 5 (release 1 was skipped from the outset) contain all three message types, and release 6 saw a few minor changes to the consignment information message, while in release 100 all three message types were completely overhauled.

Since one obvious difference between these versions is the length of the individual record types (128 characters in releases 2 to 6, 512 characters in release 100), we can summarise the full diversity of standard Fortras messages in the following short list:

- Releases 2–6 with 128-character records:
 - **BORD128:** Consignment information
 - **STAT128:** Status data
 - **ENTL128:** Unloading report
- Release 100 with 512-character records:
 - **BORD512:** Consignment information
 - **STAT512:** Status data
 - **ENTL512:** Unloading report



Example: BORD512
Structure of a BORD512 displayed in Lobster_data

And on that note, here is an example of just such a file, stripped back to only the record type identifiers and a few key values:

```
@@PHBORD512 1234 56 7 8 SENDER RECIPIEN
A00STDShipment header data
B00001SHPSender information Shipment 1
B10001TELTelephone number
B00001CONRecipient information
B10001TELTelephone number
D00001001Packages, Line items, Weights etc.
F00001001Barcode1
F00001001Barcode2
G00001Shipment totals
H00001Text1
H10001Text2
H10001Text3
B00002SHPSender information Shipment 2
... the second shipment ...
J00Totals for all shipments
Z00Control record
@@PT
```



It may seem a little strange that the record types B00 and B10 appear twice in the source structure above – once with the suffix “-SHP” and once as “Adresse_Rest” (“Address_remainder” in English). The reason for this can be found in the data: Each shipment in the file begins with the sender’s address details (SHP = shipper). However, this can be followed by various other address details, such as those of the recipient (CON = consignee). And each of these addresses comes with additional contact information, such as the telephone number contained in the B10 record. It is this logic – that the SHP address appears at the start of the shipment – that is replicated in the source structure in this special way.

As we have already mentioned, every record type in Fortras release 100 is 512 characters long. That means that in a real-life file, every line will typically come with filler spaces to bring it up to the correct length if the data within the line doesn’t contain enough characters. And as we saw in the general article about fixed record format, this is only done when we need to adhere strictly to the fixed record length. However, if we use line breaks instead, we can happily omit the filler characters at the end.

FIXED RECORD

Two lines merit our closer attention here: the ones beginning with @@.

```
@@PHBORD512 1234 56 7 8 SENDER RECIPIEN
```

All Fortras formats begin with a line like this. @@PH always comes at the beginning, followed by the format code taken from the list above (i.e. anything from BORD128 to ENTL512), which in turn is followed by information about the sender and recipient of the data and a few other small details.

```
@@PT
```

indicates the end of the file.

As with VDA, numerical values take leading zeroes as filler characters, while other values are brought up to the required length using trailing spaces. Fortras also uses implicit decimal places, so that 3.05 (for example) becomes 00003050.

→ XML

WHAT DOES XML STAND FOR?

XML stands for EXtensible Markup Language. It's straightforward, legible to humans, incredibly powerful and – it takes up an unbelievable amount of space. However, thanks to the repetitive nature of many of its textual elements, XML can be compressed very effectively, which goes some way towards making up for that disadvantage. You can find a detailed description on Wikipedia; here, we will focus solely on what is important for EDI/EAI.

A simple example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Order>
  <Header>
    <OrderNo>4711</OrderNo>
    <CustomerNo>K0815</CustomerNo>
    <OrderDate>2021-05-30T00:00:00+2</OrderDate>
  </Header>
  <LineItems>
    <LineItem No="1">
      <ItemNo>S123</ItemNo>
      <Quantity>5</Quantity>
      <PricePerUnit>9.99</PricePerUnit>
    </LineItem>
    <LineItem No="2">
      <ItemNo>H456</ItemNo>
      <Quantity>3</Quantity>
      <PricePerUnit>17.95</PricePerUnit>
    </LineItem>
  </LineItems>
</Order>
```



THE STRUCTURE

First of all, it has to be said that the line breaks and indentations in the example above help make it easier to read. You will often find documents formatted in this way – however, in principle, all this could be written out in one line without making any difference to the meaning. Whitespaces (line breaks, spaces, indentations) outside values have no effect whatsoever. Of course, if line breaks appear within certain values (e.g. item descriptions), they will still be included in the transfer. And now let's take a look at the different elements.

XML HEADER

The XML header comes at the start of the XML file. It specifies the version of the XML format (in practice, this is always 1.0) and the encoding of the file. The standalone="yes" here also indicates that this file does not need to be checked against an external format definition. More on that later. Strictly speaking, this header should always appear – however, there are plenty of examples of XML files that do without it.

In general, statements bracketed inside `<? and ?>` are called ‘processing instructions’. This header takes that format, but it isn’t really a processing instruction, technically speaking. Processing instructions have no direct impact on the data content, so we won’t spend any more time on them here.

TAGS AND ATTRIBUTES

The identifiers bracketed by `<` and `>` are called tags. Every tag is opened and then closed again, unless it has no content inside. We therefore talk of opening and closing tags. The closing tag simply has a slash (`/`) in front of the identifier.

- At the top level, there can only be exactly one tag per file. This is what is known as the root element. An XML file that has more than one set of opening and closing tags at the top level is invalid.
- A tag can simply contain a value, such as here: `<OrderNo>4711</OrderNo>`
- It’s also possible for a tag to enclose other tags, which can be nested as deeply as required. The example above serves as a nice example of this.
- A single tag can also appear multiple times. For example, we could represent the classification of an item in a catalogue structure as follows:

```
<Item no="4711" name="ABC Educational Game">
  <ProductGroup>Educational</ProductGroup>
  <ProductGroup>Games</ProductGroup>
  <ProductGroup>School</ProductGroup>
  <ProductGroup>Children</ProductGroup>
</Item>
```



After all, the item would fit well into all four of these categories. Similarly, the order in our initial example also has multiple line items.

- As well as values and other tags, a tag can contain any number of attributes, as in this example: `<LinItem no="2"> ... </LinItem>`

Unlike in HTML, the value of an attribute is always enclosed in "".

As a rule, attributes are added to the opening tag.

The important point is that an attribute can only appear once per tag.

The following is therefore invalid: `<LinItem no="2" no="3">`

After all, this wouldn't make sense – what number line item would we be talking about?

- Best of all, a tag can contain all these things at once:

```
<ParentTag attr1="blah" attr2="stuff" attr3="gubbins">
  <ChildTag1 another_attr="Maria">
    <ChildChildTag_a>So long</ChildChildTag_a>
    <ChildChildTag_b>Farewell</ChildChildTag_b>
    <ChildChildTag_c>Auf Wiedersehen, goodbye</ChildChildTag_c>
    This is the value of child tag 1.
  </ChildTag1>
  <ChildTag2>Adieu, adieu, to you and you and you</ChildTag2>
  This is the value of the parent tag.
</ParentTag>
```



- If a tag contains neither values nor child tags, it can also be written out in abbreviated form by simply merging the opening and closing tags into one.

```
<Tag_Without_Contents attr1="Attributes are allowed" />
```

Can you see the slash (/) at the end? That tells you that this is the end of the tag, and that there won't be another closing tag.

- Comments are opened with `<!--` and closed again with `-->` at the end. They can span multiple lines, but they have to be placed within individual tags and can never be nested.

And with that, we have covered everything we need to know about the structure of XML. So far, so good. But there's a little bit more to it than that.

SPECIAL CHARACTERS AND NOTATIONS

What happens if a `<` or a `>` appears in a value? That would knock an XML parser off course. It would see a `<` and think it was the start of a new tag, when in truth it was merely a value stating “a < b”. To avoid confusing the parser, we mask these characters using things called entities. This would replace `<` with `<`. Every entity begins with an ampersand (`&`) and ends with a semi-colon (`;`). Between those comes the code for the character in question. The “lt” in `<` stands for ‘lower than’. As you might expect, we then code for `>` with `>`; (“gt” standing for ‘greater than’). And because the ampersand (`&`) also has a special meaning in XML, it too needs to be coded as `&`. You’ve guessed it – “amp” stands for ‘ampersand’. The semi-colon (`;`) doesn’t need to be coded, however. Because it is only meaningful when it is preceded by an `&` along with a valid code, a standalone semi-colon won’t confuse the parser. However, one thing that causes major problems is when quotation marks appear in attribute values. These need to be coded too as `"`; (“quot” for ‘quotation mark’). Last but not least: single quotation marks (aka apostrophes) need to be masked as `'`. Incidentally, you can also write out the numerical values of the ASCII or Unicode codes for the masked characters between the `&` and the `;` – however, these aren’t as easy to remember as `lt`, `gt`, `amp`, `quot` or `apos`. Those who have already created HTML pages in text editor will immediately think of the many other entities they have used there. For example, when writing in German, you need `Ä` for capital Ä, or `ß` for the letter ß. These combinations date back to a time when the Internet was almost exclusively written in ASCII. Back then, everything that didn’t appear in ASCII code needed to be specially coded. Nowadays, we state the encoding at the top of HTML pages (if we are working cleanly). Modern browsers are also all compatible with Latin1 and Unicode, which means that there is no longer any need to code umlauts and other special characters.

As far as notation goes, the first point to note is that XML is **case sensitive**, which means that we need to pay attention to our use of capital letters. In other words, **The** is not the same as **the**, which is different from **THE**. This is a critical point for you to bear in mind. After all, some systems encourage us to develop sloppy habits on this front. There are also a few additional rules for identifiers, by which we mean the names of tags and attributes. Spaces are taboo, in any case. They are used solely to separate tag names and attributes within a tag. Entities are also forbidden. And special characters such as umlauts will

interfere with some parsers too. In brief, only the following characters are permitted: Upper- or lower-case letters (a-z|A-Z), numbers, underscores (_), dashes (-) and full stops. Colons have a special meaning that we will discuss below. In addition, no identifiers may begin with “xml” (irrespective of whether it is written in upper case, lower case or a mix of both). This is reserved for special purposes.

RULES IN PRACTICE

The following discussion relates exclusively to data XML structures. These are a lot simpler than document structures. For example, documents from OpenOffice, NeoOffice, KOffice and so on are also saved in an XML format. After all, a file in OpenDocument format is nothing more than a ZIP archive containing a handful of XML files (alongside any embedded pictures and so on). One of those files – generally the biggest one by far – will contain the text from OpenOffice Writer (or whichever program is being used). Very special rules apply here – e.g. the order of all the elements within the file will be hugely important. By contrast, if you take another look at the examples on pages 75-77 – either the order or the item and its product groups – you will quickly see that the order in these cases makes absolutely no difference. The line items contain the attribute “no”, while the product groups for the item can also be rearranged into any order. OK, the “so long, farewell, auf Wiedersehen, goodbye” example admittedly wouldn’t be as funny if we changed the order. At any rate, the following rules apply to XML files that transport solely non-document data (although some of them are equally applicable to documents too):

- If no values or sub-tags are specified, the short form is always equivalent to the longer notation comprising opening and closing tags:
`<Tag attr="abc"></Tag>` is equivalent to `<Tag attr="abc" />`
- A completely empty tag (i.e. one containing no attribute values) can even be omitted altogether: `<Empty />` can simply be deleted.
- If an attribute has no value, it can also be omitted:
`<Tag empty="" >value</Tag>` is equivalent to `<Tag>value</Tag>`
- The order of the tags and attributes is irrelevant. All organisational criteria are represented through the data themselves, e.g. through the line item numbers in the initial example.

```
<ParentTag attr1="blah" attr2="stuff" attr3="gubbins">
  <ChildTag1>Text1</ChildTag1>
  <ChildTag2 attr1="a" attr2="b" attr3="c">Text2</ChildTag2>
  <ChildTag3>Text3</ChildTag3>
</ParentTag>
```

is equivalent to:

```
<ParentTag attr1="blah" attr3="gubbins" attr2="stuff">
  <ChildTag2 attr3="c" attr2="b" attr1="a">Text2</ChildTag2>
  <ChildTag3>Text3</ChildTag3>
  <ChildTag1>Text1</ChildTag1>
</ParentTag>
```

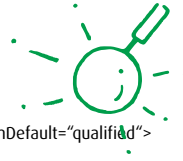


Unfortunately, at this point we should note that many XML parsers (especially custom-made ones) will see things differently. In those cases, empty tags will be required after all, and they will have to be provided in a very particular form (either `<long></long>` or `<short />`). In some data formats, the order of the tags can also be meaningful, instead of relying purely on attributes, as in the example above with the line item numbers.

SCHEMAS

A further advantage of XML structures is that you can precisely define their structure in a globally applicable form. Whereas formats such as EDIFACT or VDA come with descriptions that are more-or-less comprehensible to humans, as well as a set of general rules, XML structures can be represented in an entirely machine-readable way. This used to be done with the help of a DTD (Document Type Definition), but today, the standard is the XSD (the XML Schema Definition). There are other approaches, but XSD is currently the most widely used. These schemas are themselves written in XML – a major advantage over the tricky notation used in the old DTDs. Which also makes them comparatively more powerful.

A schema for our opening example might appear as follows:



```

<?xml version="1.0" encoding="UTF-8"?>
<s:schema xmlns:s="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <s:element name="Header" minOccurs="1" maxOccurs="1">
    <s:complexType>
      <s:sequence>
        <s:element name="OrderNo" type="s:string" minOccurs="1" maxOccurs="1"/>
        <s:element name="CustomerNo" type="s:string" minOccurs="1" maxOccurs="1"/>
        <s:element name="OrderDate" type="s:time" minOccurs="1" maxOccurs="1"/>
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="LineItem" minOccurs="1" maxOccurs="unbounded">
    <s:complexType>
      <s:sequence>
        <s:element name="ItemNo" type="s:string" minOccurs="1" maxOccurs="1"/>
        <s:element name="Quantity" type="s:integer" minOccurs="1" maxOccurs="1"/>
        <s:element name="PricePerUnit" type="s:float" minOccurs="1" maxOccurs="1"/>
        <s:element name="Name" type="s:string" minOccurs="0" maxOccurs="1"/>
      </s:sequence>
      <s:attribute name="no" type="s:integer"/>
    </s:complexType>
  </s:element>
  <s:element name="LineItems" minOccurs="1" maxOccurs="1">
    <s:complexType>
      <s:sequence>
        <s:element ref="LineItem" minOccurs="1" maxOccurs="unbounded"/>
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="Order" minOccurs="1" maxOccurs="1">
    <s:complexType>
      <s:sequence>
        <s:element ref="Header" minOccurs="1" maxOccurs="1"/>
        <s:element ref="LineItems" minOccurs="1" maxOccurs="1"/>
      </s:sequence>
    </s:complexType>
  </s:element>
</s:schema>

```

This is all fairly straightforward. If you compare the schema with the sample data, you should be able to quickly figure out the main principles. The purpose of these schemas is firstly to specify the maximum possible structure of an XML file. ‘Maximum’ in the sense that the “name” element, for example, appears in the XSD file, but not in our example XML file. In other words, the schema sometimes describes more than what is present in the actual data. Secondly, the schema serves to check the validity of existing XML data. This involves taking an order file and comparing it to the schema to see whether any elements or attributes appear that are forbidden by the schema; whether anything mandatory is missing (`minOccurs="1"`)? Or does anything appear more often than it should? A schema can also specify the number of values that an attribute or tag is allowed to have. You can also state the type of certain values or define custom types that are derived from other types (like in programming languages), as well as a host of other things.

We don’t have room for a comprehensive discussion of XSD here, so we’ll just add one more thing:

Complex XML structures are best defined using multiple schema files. This involves one central XSD that integrates other schemas by means of `include` or `import` instructions. That allows us to define certain basic types for things such as pricing information, telephone numbers or web addresses in one file; specify everything relating to customer information in another file; and then define all item-related information in a third file; before using all of this in a central document that describes the structure of an order or a catalogue.

IT WAS ALMOST TOO EASY: NAMESPACES

If only we could leave it there, XML would be wonderfully straightforward. And so readable too! Unfortunately, there is another element that makes everything more complicated: namespaces. As we have already mentioned, complex structures are sometimes described over multiple XSDs. And this is where namespaces often come into play. We could compare these to the packages used in Java. A class called `Price` might appear in a package called `en.example.basicTypes`, while another completely separate and independent class called `Price` appears in the package `en.example.itemInfo`. Similarly, in XML, a price element can be defined as a basic type in the following way:

```
<s:simpleType name="Price">
  <s:restriction base="s:float">
    <s:minInclusive value="0" />
  </s:restriction>
</s:simpleType>
```



This specifies that a price is written a decimal value that can be zero, but can never be negative.

However, it is possible that the price information for a particular item might simultaneously also include the purchase price, the sale price and the VAT rate:

```
<s:complexType name="Price">
  <s:sequence>
    <s:element name="PurchPrice" type="Price" />
    <!-- minOccurs and maxOccurs are not specified, so a 1 is assumed for both -->
    <s:element name="SalePrice" type="Price" />
    <s:element name="VAT" type="s:float" />
  </s:sequence>
</s:complexType>
```



The complex type is called "Price", and two of its values are of a type also called "Price" – but this is the simple type. Now if this doesn't lead to some degree of confusion between types ...

And this is where namespaces come into play. You can see what these look like in the schema itself. First of all, a namespace needs to be declared:

```
<s:schema xmlns:s="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
```


Here, we have declared the namespace "s". This is the exact URL used in schemas. In normal XML files, any URL could appear here in principle, since it doesn't actually need to link to anywhere – unless you actually want to test the structure. But we'll come back to that at the end. In any case, every namespace gets its own URL.

We use the declared namespace by placing it as a prefix in front of the tag or attribute names (or attribute values, at least in schemas):

```
<s:element name="ItemNo" type="s:string" minOccurs="1" maxOccurs="1"/>
```

Both the “element” tag and the “string” type are contained in the namespace “s”, which simply contains all the element and attribute types defined by the World Wide Web Consortium (W3C for short).

If we apply this to our price example, we should place the basic definition of price (i.e. that it cannot be negative) in the “basic” namespace (for example), while the pricing information for an item including purchase price, sale price and VAT will appear in the “item” namespace. Once we have given the simple price its namespace, we can then define the item price, followed by a mini-item:



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!-- Schema for item price -->
<!-- Schema for basic price -->
<!-- Import basic schema -->
<!-- Note: URL must match schema tag -->
...
<complexType name="Price">
  <sequence>
    <element name="PurchPrice" type="basic:Price" />
    <element name="SalePrice" type="basic:Price" />
    <element name="VAT" type="s:float" />
  </sequence>
</complexType>

<element name="Item">
  <element name="Name" type="s:string" />
  <element name="PriceInfo" type="Price" />
<!-- Here we are dealing with the (complex) price in the same schema/namespace as for the item, which is why there is no prefix -->
```

```
<!-- Here, we begin by specifying which schema and namespace the types and elements defined below should appear in. -->
```

```
<!-- We have also already indicated that we will use elements from the "basic" namespace, and stated where their definitions come from. -->
```

```
<!-- And now we import the basic.xsd schema. -->
```

```
<import namespace="http://www.lobster.de/madeup/basic.xsd"
  schemaLocation="basic.xsd />
```

```
<!-- Please note: The URL must exactly match the one provided in the schema tag above it. We have also assumed that the file basic.xsd is located next to the item.xsd file. -->
```

```
...
```

```
<complexType name="Price">
```

```
  <sequence>
```

```
    <element name="PurchPrice" type="basic:Price" />
```

```
    <element name="SalePrice" type="basic:Price" />
```

```
    <element name="VAT" type="s:float" />
```

```
  </sequence>
```

```
</complexType>
```

```
<element name="Item">
```

```
  <element name="Name" type="s:string" />
```

```
  <element name="PriceInfo" type="Price" />
```

```
<!-- Here we are dealing with the (complex) price in the same schema/namespace as for the item, which is why there is no prefix -->
```

```

<s:element name="OfferPrice" type="basic:Price" />
<!-- This is the other simple price, so it needs a prefix -->
</s:element>
...
</s:schema>

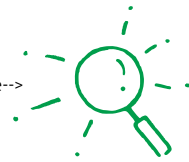
```

The item itself is part of a catalogue that is assigned the namespace "cat" (let's leave the schema completely to one side here). In the XML file, which uses both schemas, the end result looks something like this:

```

<cat:Catalogue xmlns:cat="http://www.lobster.de/madeup/catalogue.xsd">
<!-- Here, the namespace "cat" is declared in the first element that uses that namespace-->
...
<item:Item xmlns:item="http://www.lobster.de/madeup/item.xsd">
<!-- And now we also declare the "item" namespace -->
<item:Name>USB Stick 32G</item:Name>
<item:Price>
<item:PurchPrice>5.50</item:PurchPrice>
<!-- Here, the price from "basic:" is still just the number, which is why we don't see the namespace anymore -->
<item:SalePrice>17.99</item:SalePrice>
<item:VAT>19.0</item:VAT>
</item:Price>
<item:OfferPrice>13.49</item:OfferPrice>
</item:Item>
...
</cat:Catalogue>

```



If we now want to validate this XML file – i.e. check that its structure is clean – we will need to include the schema files with the specified URLs. If we decide to skip the validation, then the URL could be anything we like, provided it looks like a URL.

But let's leave things here. There are endless examples and detailed explanations of namespaces to be found online. All you need to know for now is that namespaces exist, and what the prefixes stand for.

 **OPENTRANS**

Let's start by quoting a paragraph from the **openTRANS homepage**:

"The openTRANS initiative brings leading German and international companies together under the leadership of the Fraunhofer IAO with the goal of standardising business documents (orders, delivery notes, invoices etc.) in order to build a foundation for electronic system-to-system communication. An expert working group has been formed to define these business documents using XML as a basis and to develop integration solutions for buyers, suppliers and marketplace operators."

In other words, we can define openTRANS as a messaging standard that takes XML as its underlying data format. The choice of XML is a sensible one, as this means any software that understands XML will also be able to read or generate openTRANS documents. Incidentally, openTRANS and BMEcat are closely linked, with openTRANS schemas even using types and elements derived from the BMEcat world.

So what kinds of documents are there? As of September 2021, openTRANS is in version 2.1, which contains:

- RFQ (Request For Quotation)
- QUOTATION
- ORDER
- ORDERCHANGE
- ORDERRESPONSE
- DISPATCHNOTIFICATION
- RECEIPTACKNOWLEDGEMENT
- INVOICE
- INVOICELIST
- REMITTANCEADVICE

This list has been taken from the openTRANS homepage.



Can you use openTRANS, and are you allowed to?

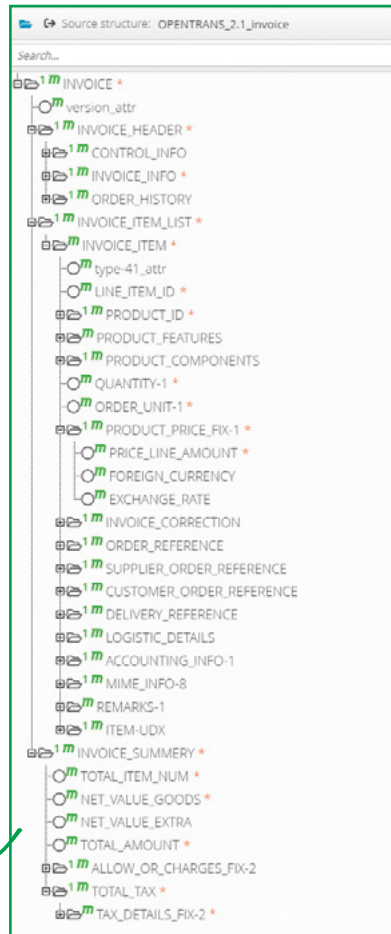
The answer to the second question is yes. It's even free of charge. Of course, we aren't the final authority on this, so you can find more detailed terms and conditions in the FAQs on the openTRANS homepage.

As for the "can you" part: As we have already said, the choice of XML as the underlying data format for this messaging standard means that any EDI/EAI software that can handle XML will also be able to work with openTRANS documents. Once you have registered on the homepage (free of charge), you can download format descriptions in either human-legible (PDF) format or as machine-readable XML schemas. Beyond that, however, you will need to do a bit of extra work, depending on your software.

Example:

As a final example, here is the rough structure of the INVOICE in the openTRANS standard version 2.1:

As you can see, this structure has a lot of elements. That represents an awful lot of work. So it's handy that you don't have to do everything yourself!



 **BMECAT**

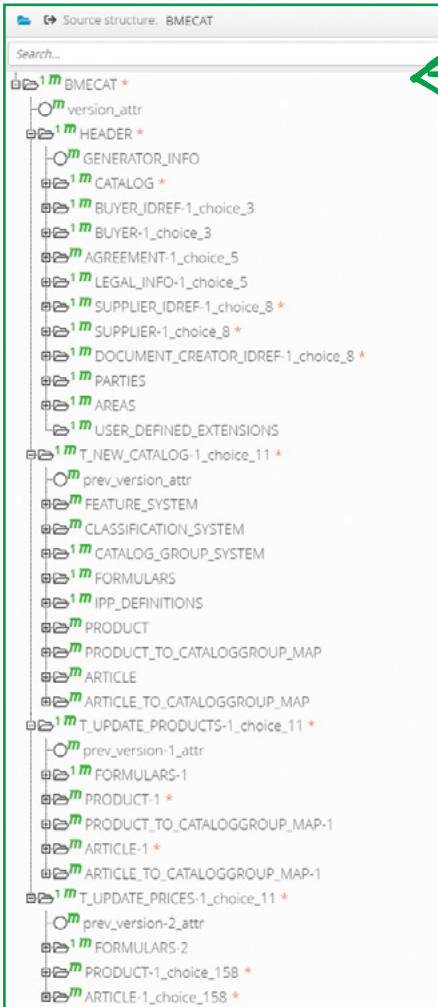
To start with, let's take a look at the **BMEcat project homepage**:

"The German Association for Supply Chain Management, Procurement and Logistics (BME), based in Frankfurt, has launched an initiative to develop an electronic data interchange standard for product catalogues that has been actively supported by major companies. [...] BMEcat is currently open to public review in the version '2005 final draft'. During this phase, the new version will be tested in practice and any errors will be uncovered. BMEcat 2005 grants new sectors and product groups access to the electronic interchange of product information. [...] BMEcat establishes a basis for simple transfers of catalogue data from a wide range of different formats, and in particular, it creates the right conditions to drive the online exchange of goods between companies in Germany. The XML-based standard BMEcat has been successfully deployed in many different projects."

The 2005 version is still the latest one, and is also used as part of the openTRANS standard. That means that elements from BMEcat 2005 can also be found in openTRANS structures. Even their schemas are linked. The BMEcat messaging standard is based on XML.

What can BMEcat do?

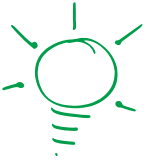
Let's take a quick look at the broad outline of the BMEcat structure:



Structure of BMEcat 2005 -
displayed in Lobster_data

The key points relating to this structure are:

- The structure is fairly large. It comprises over 2500 elements in total (which are mostly numbered in sequence in the structure).
- There are three types of BMEcat files, which you can see under the Choice-228 node. A choice (the term comes from the XML schema) offers multiple, alternative elements, only one of which may be used. In this case, only one of the following types of content can be transferred:
 - An entire (new) catalogue (T_NEW_CATALOG)
 - A product update (T_UPDATE_PRODUCTS)
 - A price update (T_UPDATE_PRICES)
- In other words, each individual file only uses part of the overall structure. Even an entire new catalogue will consist of only around 1500 elements in total. And only a small number of those will actually appear in the file. However, certain elements will appear very frequently.
- An entire catalogue consists chiefly of three main parts:
 - The catalogue groups
 - The products
 - The classification (which products are assigned to which catalogue groups)
- Updates contain only the most essential information. In principle, a product update should cover all the data relating to the items, including their classification into various catalogue groups, but it should not include any changes to existing groups. And price updates are limited to a small amount of basic product information, as well as their pricing data. If any changes are made to the catalogue groups, an entirely new catalogue will be required.
- Catalogue groups can be assigned to just one superordinate group in a structure. Products, by contrast, can belong to any number of different catalogue groups.
- Certain general information is contained in the header, which is separate from the three types outlined above. This can include the catalogue being referred to, the language, the parties involved.
- If one of the elements available in the standard proves inadequate, it is possible to specify certain user-defined extensions. This allows for extensions to be made to the format according to your needs. However, it requires the agreement of all the parties involved!

**Who uses it, and what does it cost?**

In fact, it's used by everyone who wants to exchange catalogue data – especially the manufacturing and retail sectors. According to the BME's own marketing, the format is now the "de facto standard for the interchange of electronic product catalogues" and is used primarily in the German-speaking world.

There is no charge to use the format. Once you have registered on the homepage (also free of charge), you can gain access to the download zone, where this will be explained to you. You will also have access to format descriptions for the various versions in a range of different file formats, including PDF and XSD.

→ CSV

WHAT DOES IT MEAN?

CSV is short for **Comma Separated Values** – though the comma itself isn't mandatory. In principle, any other character can also be used as a separator – of course the sensible thing would be to use characters that appear only rarely within the values themselves. A popular choice alongside the comma is the semi-colon (;), but out in the wild you might also encounter the vertical bar (|), the at-sign (@), the tab, and other outlandish ideas, including unprintable characters such as the ASCII 7 bell character.

A simple example:

```
OH;4711;K0815;30.05.2021
LIN;1;S123;5;9.99
LIN;2;H456;3;17.95
```



Here, the semi-colon has been used as a separator. As you can see, CSV is an incredibly economical format, as each value is separated by just one character. At the end of each record there is typically a line break, which also takes up only one character, or at worst two characters. Fixed record can't compete with that – let alone XML.

Let's go through each line in detail.

```
OH;4711;K0815;30.05.2021
```

- **OH** stands for order header here. This is the record type identifier, which lets a piece of software (or a human) recognise what kind of record is coming next.
- **4711** is the order number.
- **K0815** is the customer number.
- **30.05.2021** is the order date, in standard German date format (dd.mm.yyyy).

```
LIN;1;S123;5;9.99
```

- **LIN** is the line item identifier.
- **1** is the line item number.
- **S123** is the item number.
- **5** is the quantity.
- **9.99** is the price per unit.

In CSV, unlike in fixed record, it isn't advisable to omit line breaks. This is because a very practical rule is applied here: if there are no more values in a record, you can also omit the separators. Let's assume that in a particular line item, the price per unit might be followed by the name and description of the item:

```
LIN;1;S123;5;9.99;USB Stick;Memory stick with USB 3.0 and 4 GB capacity
```

if we wanted to leave these two textual values out, we wouldn't have to write:

```
LIN;1;S123;5;9.99;;
```

instead, we could simply leave out the last two semi-colons altogether. If this were a fixed record file, we would be able to skip the line break – but then we would also have to add extra spaces as filler characters. Not exactly a fair trade.

RECORD TYPE IDENTIFIERS

An individual message is typically made up of various parts. These might include a header (as in the example above), and often a trailer record at the end, as well as information on individual shipments, call-offs etc., details on line items, and so on. Typically, the record type will appear at the very beginning. This is extremely practical, as we can then apply just one simple rule:

Look at the first value in each line to find out what record type is coming next. With the help of the format definition, we now know what values will appear in which order. We simply chop each line up by its separators, and there we have it. A very easily digestible format.

RULES IN PRACTICE

As mentioned at the beginning, it's important to choose a separator character that only rarely appears in the values. 'Only rarely' isn't the same as 'never', however. So what do we do when the separator does turn up in a value?

Example:

```
LIN;1;S123;5;9.99;USB Stick;"Memory stick; USB 3.0; 4 GB of capacity"
```

Here, the semi-colon also appears twice in the item description. But you can already see how we resolve this: we put quotation marks around the value. The use of the " character for this is itself very widespread, but not necessarily mandatory.

The most commonly used alternative is the apostrophe ('). Within a given file, the quotation character is just as fixed as the separator.

That brings us straight on to the next problem: What if the item description itself contains quotation marks (or another quotation character)? To get round this, the character is typically escaped, or invalidated. This is done using the backslash:

"This text contains \"quotation marks\" partway through."

A \" is thus for an invalidated ", which is interpreted merely as the character itself, and not as a quotation character.

And to add to the fun: If the backslash (\) itself appears in the text, it escapes itself by adding an additional backslash: \\

One more thing: In order to make this compact but highly unstructured format easier for humans to read, a sort of header is often provided on the first line, containing only the names of the values that will also appear in the subsequent lines. This is particularly common in data exports with just one record type that do not have a record type identifier at the beginning. As an example:

```
Surname,FirstName,Street,HouseNo,Postcode, City
Meier,Hans,Main Street,22,12345,Downtown
Müller,Emil,Side Road,17,98765,Suburbia
```



In these cases, if we only want to process the actual data then we simply ignore the first line.

VARIATIONS

When humans process data, they seldom do so directly in CSV format.

Instead, they typically use Excel or an open-source equivalent (such as OO Calc). In Excel, you can export your tables at any time as CSV files, and if you double click on a .csv file in Windows then Excel will usually be the program assigned to open it. Excel does one thing differently when exporting to CSV: Instead of invalidating quotation marks with an escape character such as a \, it simply doubles them:

"This text contains ""quotation marks"" partway through."

This is completely allowed!

You should expect to one day receive files in this format too. But your converter software should be able to cope.

As for the other variations:

CSV is used particularly frequently for compact exports or even imports into databases (for example). Because these export/import routines are generally bashed out very quickly, you will encounter a wide range of non-standard solutions. One developer might think that her values will never contain anything that needs to be wrapped in “”, and as such, her routines will strictly refuse to read any files with “”. Others will be overly cautious, and will wrap even purely numerical values in “”. You should therefore make sure that the software you are using to process CSV files is flexible not only in terms of the choice of special characters (separators, quotation marks and escape characters), but also when it comes to quoting rules.

USE

Unlike fixed record, there are no major, pre-defined messaging standards that are based on CSV. As mentioned in the previous section, CSV is used especially often for quick and straightforward data imports and exports. This is because it is wonderfully compact, has the lowest overhead of any formats (although EDIFACT and X12 can compete here too, they are much more complex in terms of logic), and is incredibly simple.

 **IDOC****GENERAL INFORMATION**

IDoc is the format in which SAP systems read and export their data. You can find a rough outline of how it works in the **chapter on SAP** under “Data sources and data sinks”.

Fundamentally, IDoc is a fixed record structure. Or XML too, depending on the circumstances. However, because the format itself is confusing enough, let’s take a moment to explain that last sentence. SAP has long provided its ALE interface for exchanging entire documents. For more information on this, please see the chapter mentioned above. IDocs in fixed record format are exchanged via ALE. ALE communication was originally intended for interchange between two SAP systems. However, for some time now, SAP has also offered communication with the outside world (i.e. with non-SAP software) via the module PI (formerly XI), which speaks XML. Generally speaking, the XML structures are straightforward conversions of the old fixed record structures, which are simply clad in XML tags. This means that the content, aside from a few header fields that are omitted in XML, along with a few other minor details, remains the same, regardless of whether the IDoc is in fixed record format or XML.

STRUCTURE

The structure of IDocs is – how can we put this? – extremely complex. As a result, we don’t want to go into too much detail here. Instead, let’s simply take a look at the structure for the ORDERS05 format – i.e. an order. You can find it displayed on the following page.

The left-hand screenshot shows the general structure of the IDoc. The right-hand screenshot shows the **E2EDP01006** node on its own, opened out. This node alone is more complex than everything else around it. The P stands for 'Position' (German for 'line item'), and each of its sub-nodes contains data for a particular line item.



We are talking about nodes here, but of course, this is only how they are represented in this particular specialist software. IDoc simply contains segments – i.e. with a record type identifier at the front followed by lots of data.

We will spare you a detailed account of this, however. Instead, we just want to make a few points:

- In principle, every IDoc will begin with a record of type EDI_DC40. The addition of EDI in this identifier tells you that it is intended for electronic data interchange. The structure of this EDI_DC40 is always the same, regardless of what kind of IDoc you are looking at (though things can change between the different versions). It contains basic information such as:
 - The type and version of the IDoc (e.g. ORDERS05).
 - The client of the SAP system to which it belongs.
 - The document number.
 - The message type (more on this later).
 - Information about the sender and the recipient.
 - The date/time stamp.
 - etc.
- An IDoc type (such as ORDERS) can contain multiple message types. For example, an IDoc of the type ORDERS might contain not just an order, but also the order confirmation. This information would then appear in the message type field.
- Each segment type is represented in SAP by a structure data type that defines the number, sizes, types and names of the fields in order. These structure types all have names beginning with E1, such as E1EDP01. In the record type identifier of a fixed record IDoc, the name of the record type begins with E2, giving E2EDP01 (Don't ask why!). The record type identifiers explain why the nodes in the structures above are named the way they are (with E2 at the beginning). In XML IDocs, however, the tag names once again begin with E1 because they represent a type name.
- Both IDoc types (such as ORDERS) and individual record types (e.g. EDP01) have their own versions. ORDERS05 is version 5 of the ORDERS IDoc, while EDP01007 is version 7 of the EDP01 record type.

- The following rules apply to fixed record IDocs: Each segment begins with half a dozen fields containing information such as the client and the document number. Every segment of an IDoc has to have the same values here. Parent-child relationships are also established between segments. However, the hierarchical structure of the XML IDoc means that it doesn't require these fields.
- In XML IDocs, the name of the root element is identical to the message type; after that comes one or more IDoc elements, which are in turn followed by the actual IDoc beginning with the EDI_DC40 control record.
- SAP systems are often highly customised to the needs of individual operators. When that is the case, the document types will also deviate from the standards offered by an unmodified SAP system. These are known as CIM types. In other words, IDocs of the same type and the same version can vary from SAP installation to SAP installation.

Overall, the IDoc format is more or less a table dump presented in fixed record format. Because the IDoc was originally only intended for direct communication between two SAP systems, and not for sending data in this format across the world (or to other software packages), this is perfectly fair. However, this realisation doesn't help us much in our day-to-day work.

Within the standard, the XML files created by an SAP XI or PI are structured in a very similar way to the fixed record IDoc (apart from the XML packaging). Record types and fields are virtually identical, aside from the header fields named above. Of course, the output format can be tweaked, in deviation from the standard IDoc. Fortunately, every SAP system is capable of outputting the format description for its own IDocs. For fixed record IDocs, these are called IDoc parser files, while XML IDocs are assigned an XML schema. You can obtain both of these through transaction WE60.

THE MOST IMPORTANT THINGS FOR YOU

If you own an SAP system yourself, you have partners who have announced their plans to send you IDocs or your partners expect to receive them from you, you should make sure to comprehensively integrate this document type, or enable SAP communication as a whole. Ideally, your software should be able to talk directly to SAP and independently request the structures that are applicable to this particular installation (i.e. the CIM types). As we have already mentioned, SAP also offers the option to export descriptions of its structures. Your software should be able to directly import those descriptions (IDoc parser files or XML IDoc schemas), so your partners can send you the right structures to match your IDocs from the word go.

→ ANSI ASC X12

WHAT IS IT?

ANSI ASC X12 stands for American National Standards Institute Accredited Standards Committee X12. This messaging standard defines over 300 message types for all kinds of different sectors, including retail, transportation and insurance. There are also many parallels with EDIFACT – largely due to the fact that X12 is a predecessor to EDIFACT. It is still widely used, particularly in the USA, but also in Australia (among other territories). The X12 committee itself runs X12 as a national standard and refers to international standards on a page belonging to the UN/CEFACT, which itself references UN/EDIFACT in turn. All of which might make you think you don't need to pay any attention to X12, since EDIFACT is recognised as an international standard. Unfortunately, it's not quite as simple as that. Anybody who deals with business partners outside Europe will generally also come into contact with X12 sooner or later. For that reason, we have provided a little more detailed information below:

Example:

Here, you can see a small X12 file containing an order (in which the first line crosses a line break):

```

ISA * * * * *ZZ*SENDER *ZZ*RECEIVER *041201*1200*U*
00305*000000101*1*P^~
GS*PO*SENDER*RECEIVER*041201*1200*101*X*003050
ST*850*000000101
BEG*22*NE*101**041201*123456
FOB*DF*ZZ*JMJ
DTM*037*041205
DTM*038*041215
DTM*002*041218
TD1*CNT90*1
TD5****JJ*X
TD3*40
N1*OB**92*7759
N3*111 Buyer St
N4*Conyers*GA*30094*US
N1*SE*Foo Bar Sellers
N4****US
REF*DP*101
PO1*100*1*EA***ZZ*BL47*HD*100
PID*F****Widget
    
```



ANSI ASC X12

```
P04**1*EA
N1*CT**38*CN
N4****CN
CTT*1*100
SE*22*000000101
GE*1*101
IEA*1*000000101
```

If you haven't read the article on EDIFACT yet, you should go back and do so now, as the logic behind the order of the segments is very similar. By way of clarification, the screenshot to the right shows part of the structure of message type 850 (purchase order) displayed in Lobster_data:

```
├─ m LOOP_ID_-_SPI
│   └─ 1 m SPI *
│       └─ m REF-4
│           └─ m DTM-3
│               └─ m MTX-2
│                   └─ m LOOP_ID_-_N1-1
│                       └─ 1 m N1-1 *
│                           └─ m N2-1
│                               └─ m N3-1
│                                   └─ 1 m N4-1
│                                       └─ m REF-5
│                                           └─ 1 m G61
│                                               └─ m MTX-3
│                                                   └─ m LOOP_ID_-_CB1
│                                                       └─ 1 m CB1 *
│                                                           └─ m REF-6
│                                                               └─ m DTM-4
│                                                                   └─ 1 m LDT-1
│                                                                       └─ m MTX-4
```

```
Source structure: 850_004060
Search...
├─ 1 m ISA *
├─ 1 m GS *
├─ 1 m ST *
├─ 1 m BEG *
├─ 1 m CUR
├─ m REF *
├─ m PER
├─ m TAX *
├─ m FOB *
├─ m CTP *
├─ m PAM
├─ m CSH
├─ m TC2 *
├─ m LOOP-ID-SAC
├─ m ITD *
├─ m DIS
├─ 1 m INC
├─ m DTM
├─ m LIN
├─ m SI *
├─ m PID
├─ m MEA
├─ m PWK
├─ m PKG
├─ m TD1
├─ m TD5 *
├─ m TD3
├─ m TD4
├─ m MAN
├─ m PCT *
├─ m CTB
├─ m TXI *
├─ m LOOP-ID-LDT
├─ m LOOP-ID-AMT
├─ m LOOP-ID-N9
├─ m LOOP-ID-N1
├─ m LOOP-ID-LM
```

The SPI loop in X12 message 850

EDIFACT calls segment groups (SG1 to SGx), while X12 refers to them as loops. The logic, however, remains the same. Individual segments can be repeated. Whenever a group is repeated, the first segment in the group (or loop) always indicates that it is starting again from the beginning. And here too, once we have passed a particular point in the structure, we can't go back again.

THE DETAILS

Differences:

X12 differs a little from EDIFACT when we examine it in detail:

- First of all, there is no equivalent to the UNA segment. Instead, the separators used are identified by virtue of their position in the ISA segment:
 - The character directly following ISA is the field separator (in the example, the character used is *).
 - Position 105 is where we find the separator for values within components (aka composites – in this case the separator character is ^).
 - The character after that is the segment separator. In our case, the segment separator is a line break.
 - The decimal separator is not specified, and there is no escape character either. The entire ISA segment is structured like a fixed record.
- One thing that has recently been introduced to EDIFACT but is not used there, is the repetition character. This appears in position 83. If there is an "U", it stands for 'unused'. If this character is used, a field or an entire composite can be repeated multiple times within a segment, with the individual repetitions separated by this character. This needs to be specified within the message description, however – something that doesn't happen all that often.
- There are no subsets in X12. The X12 committee takes a parsimonious approach here, and charges a fee for the format descriptions of each new version.
- The message types are identified by their numbers (850 in the example). The more meaningful "PO" (for purchase order) in the GS segment is not directly equivalent to this, since it refers to a functional group – and therefore might refer to multiple message types that belong together thematically. For example, the "SO" functional group contains ten different message types (version 006040).

- Segment identifiers do not need to be three characters long, as in EDIFACT. Two can sometimes be enough as well.

Commonalities:

- As we have just seen, X12 features segments, standard fields and composites (also known as components), while loops are equivalent to segment groups.
- X12 also has a wide range of control segments that introduce files or bracket messages:
 - ISA functions broadly as a combination of UNA and UNB, and includes information such as the parties involved, the date/time, and the most important special characters. There is no required character set, since this is set to ASCII in X12.
 - GS and GE work similarly to UNG and UNE, and bracket groups of thematically associated messages (functional groups). In the example above, the “PO” in the GS segment stands for purchase order.
 - ST and SE bracket messages in the same way as UNH and UNT do in EDIFACT. In the ST above, the code 850 stands for the message type, which is a purchase order.
 - At the end of the file comes the final IEA segment, which works in the same way as the UNZ in EDIFACT. Groups are mandatory in X12, so this segment contains the number of groups in the file (GS/GE).
 - The ISA/IEA, GE/GE and ST/SE pairs also contain identification numbers to ensure that they are associated with each other. SE also contains a counter for all the segments in the message, including the SE segment itself. As such, GE also contains the total number of messages (aka transaction sets) within its group.

Of course, X12 has evolved over time. The versions are named differently, however. For example, version number 006040 stands for:

version 6, release 4, sub-release 0.

Incidentally, the end of the GS segment (and also the ISA segment) in our example message indicates that the message is written in X12 version 003050.

→ ENG DAT

WHAT IS ENG DAT?

Strictly speaking, ENG DAT doesn't really belong on our list. It's neither a data format, nor a messaging standard in the narrow sense of the term. We really ought to create a new category for it, but that would be overdoing things slightly. So here we are.

ENG DAT stands for **Engineering Data Message**, and is a defined workflow for exchanging technical documents, primarily in the automotive industry. In addition to technical documents, it can also be used to exchange contact information, which is done by means of ENGPART messages that are transferred within an ENG DAT message.



The latest specification of ENG DAT is V3.1, while the current version of ENGPART is V4.1. You can obtain more detailed information about ENG DAT from the VDA (the German Association of the Automotive Industry) and from the **Strategic Automotive product data Standards Industry Group**, or SASIG for short. The OFTP protocol has emerged as the standard means of data transfer.

The automotive sector in particular (i.e. the branch of industry involved in cars, accessories, spare parts, transportation and so on) exchanges enormous quantities of CAD files. CAD stands for **Computer-Aided Design**, which refers to digital techniques used to design products, components, and so on. These CAD files contain technical information, but they have no room for organisational information, such as points of contact, relevant departments, and so on. Data of this kind – which are also known as metadata – therefore need to be transmitted alongside the pure CAD files, and in the process, we need to make sure that the right CAD files are kept together with the right extra data. These metadata form part of the ENG DAT message alongside the application data, and might contain information about the sender and the recipient, or about the purpose of the data.

General contact information – the partner master data – is exchanged by means of ENGPART messages. In addition, businesses in the automotive sector need to be able to request certain documents from their partners (suppliers/customers), or to confirm receipt of such documents. These defined processes are known as workflows.

There is no official requirement that OFTP should be used for transferring data; however, the OFTP protocol – like ENG DAT itself – originates from the automotive industry, where it

has developed into a standard transfer path. Until relatively recently, OFTP was still used over ISDN lines, but companies are now beginning to switch to the TCP/IP- and TLS-based OFTP2. You can find more information about this in the section on the OFTP transfer protocol.

THE TECHNICAL SIDE

As we've already discussed transfers via OFTP, we won't go into more detail on that. But how can we indicate which files form part of a particular transfer (for example)? After all, we could easily send multiple ENGDAT messages in a single connection. This problem is solved using file templates:

During transfer, each file is assigned a logical file name. And – aside from a sequential counter – this is generally identical for all the files within a single message. Here's what that looks like in practice for ENGDAT versions 2 and 3:

- **ENGDAT V2:** "ENG" or "EN2" + time stamp in "yyMMddHHmmss" format + route (aka address code, five characters long) + number of technical documents (three characters) + counter for the current file (three characters). For example, the first of four files sent at 12.30 p.m. on 23/02/2021 would be:

```
ENG210223123000ROUTE004002
```

That comes to 26 characters in total.

- **ENGDAT V3:** "ENG" or "EN3" + time stamp in "yDDdHHmmss" format + route (aka address code, five characters long) + number of technical documents (four characters) + counter for the current file. The same thing in version 3 would be:

```
ENG1054123000ROUTE00040002
```

Once again, it comes to 26 characters. The date format here is particularly odd. Only the last digit of the year (the 1 from 2021) is used, while the date is counted as the nth day in the year, from 001 to 365 (or 366). But that's just the way it's been defined.

In each of these examples, we are sending 4 associated files whose logical file names differ only in terms of the sequential counter that appears at the end of the file name. Because the total number of files is included in the name, we can easily spot if a file is missing and request it from our partner.

Next, let's talk about the data formats. The CAD data themselves (the application data) are held in one of the CAD formats used by the various different programs. Of course, the parties involved will need to agree on a single format.

But what do metadata and ENGPART messages look like? Previously, EDIFACT was used for these data, but in the current versions (ENGDAT V3 and ENGPART V4) XML comes into play. We don't need to go into any more detail here – you can look into it if you actually intend to use ENGDAT. That said, these low-level things ought to be handled by the software you are using, so you shouldn't have to engage with them directly yourself.

And now let's take a quick look at the defined message types. Generally speaking, a V3 ENGDAT message can come in three different conformance classes (CCs):

- **CC1:** Request for documents (this is further subdivided into 'a' and 'b' in its structure).
- **CC2:** Transfer of documents with no additional information about the files included.
- **CC3:** Transfer of documents with additional information about the files included.
- **CC4:** Confirmation of an incoming message (this is further subdivided into 'a' and 'b' in its structure).
- **CC5:** This doesn't exist as a message; we only refer to conformance class 5 when the software we are using supports CC1 to CC4.

One other thing: Strictly speaking, in EDI, the data should be exchanged fully automatically between the IT systems of the partners involved. In other words, there should be no manual intervention. However, this theory falls by the wayside in ENGDAT, since transfers need to be made interactively by an employee whose job is to input or define the relevant metadata to go with the application data, and to maintain the partner master data. The data interchange itself takes place largely automatically, but a little manual work is a necessary part of the process. So if you already work with ENGDAT or are likely to use it in future, then you should make sure that the EDI software you choose offers you a reasonable level of ENGDAT integration.

MISCELLANEOUS

WHAT OTHER DATA FORMATS ARE THERE?

We've already described the most important standards over the previous pages, but there is still one more that we need to mention here:

BWA

The BWA format comes from the German book industry. It's a messaging standard comprising a total of four different messages:

- Order
- Delivery note
- Acknowledgment
- Title master data

The underlying format is a somewhat adventurous mixture of CSV and fixed record. Fortunately, this format is now considered outdated. For some time now, the German Publishers and Booksellers Association (Börsenverein des deutschen Buchhandels) – or more specifically, its “Working Group on Processes, Streamlining and Organisation” – has recommended using the EDIFACT subset EANCOM instead, since it simply offers more options.

Nonetheless, here is an order record in BWA format to serve as an illustration of its structure:

Not exactly clear, is it? But as we said, the BWA standard is now considered outdated, and EANCOM is recommended instead.

BWA order displayed in Lobster_data



So far so good on the messaging standards front. Now we will turn our attention to the – let’s say – more idiosyncratic formats.

MS EXCEL

Let’s get one thing clear from the start: Excel is **not a data interchange format** within the meaning of EDI! Regrettably, that doesn’t stop some people from using it as one. :(Excel generally means manually completed forms, which regularly exhibit small or even large discrepancies and are likely unsuitable for fully automated processing.

Some particularly imaginative peers will even send out Excel files containing internal links to other Excel files. These will inevitably be stored on C:\somewhere\nowhere, and won’t be sent out along with the file linking to them.

Our top tip is to avoid any data interchange in Excel format! Excel can export files to CSV and re-import them too, which is better than trying to communicate directly via Excel. After all, we are talking about automated processes here, and not about an employee looking at an Excel file on a screen and manually copying its contents into their system. If this is unavoidable: fine. You’ll just have to bite the bullet. Modern converters can read or output Excel files too. But you (and your partner) will have to live with the fact that you won’t be able to work with a standard of seamless, fully automatic processing that is (strictly speaking) required for EDI, and which is perfectly possible if you use standard formats.

PDF

The same goes for PDF as for Excel: It’s designed for people to look at on a screen, or to print out. It isn’t designed to be used in EDI processes, and it is **not** suitable for them either. Only in rare cases where PDF files have been produced fully automatically and with a reliably uniform structure is it possible to process them automatically to a limited degree. If this requirement isn’t met, all you can do is apply an OCR-like pre-processing to convert what the human user sees into a somewhat machine-readable form.

PDF is a fantastic invention and is also often used (in conjunction with electronic signatures) to archive invoices and similar documents in a way that protects them from being manipulated. But it’s completely unsuitable for EDI.



COMMUNICATION PATHS, DATA SOURCES & DATA SINKS.

Introduction	112
Public paths	115
Email	115
FTP	117
HTTP	119
AS2	125
SFTP/SCP	131
X.400	132
OFTP	135
WebDAV	138
Internal paths/sources/sinks	140
File system	140
SAP ALE and RFC	144
Databases	146
Message services	150
AS/400 DataQueue	153
Miscellaneous	156

COMMUNICATION PATHS, DATA SOURCES & DATA SINKS

→ INTRODUCTION

BASIC INFORMATION ABOUT COMMUNICATION PATHS, DATA SOURCES & DATA SINKS

Beyond the data formats used in EDI (and EAI), we are also interested in how data come into a system, and how they leave it again afterwards. In other words, how do your partners' and customers' data enter your EDI system, how does that system communicate with your internal systems, and how do your partners get their data back?

It is difficult to make any clear distinction here – whether between the communication itself and data sources or sinks, or between public and internal communication. After all, emails, FTP transfers and HTTP requests could just as easily be used on your own intranet in the same way as they are sent throughout the world over the web. Likewise, it is perfectly possible for databases or ERP systems to be connected over long distances (e.g. via a VPN).

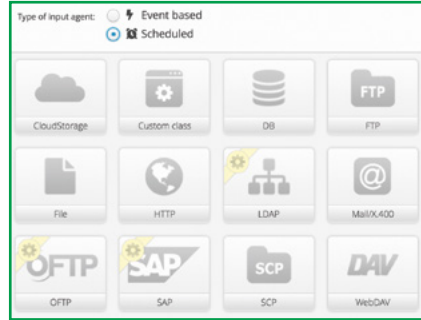
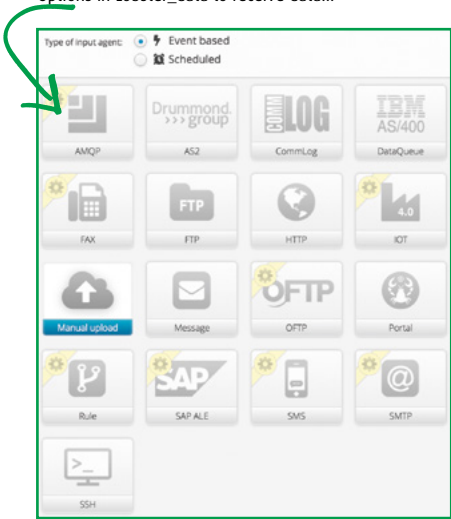
Every data source and sink naturally comes with one or more modes of communication. For example, a database might be accessed via JDBC or ODBC; warehouse management systems can import and export text files or we can also directly access the tables in the underlying database; and systems like SAP often have their own specialised access methods, in this case referred to as ALE and RFC.

Nonetheless, over the following pages we shall draw a distinction between public and internal. However, you should keep in mind that the boundary between these two concepts is fluid in much the same way as the boundary between EDI and EAI tends to be fluid in practice, since many systems and communication paths need to be accessed for both.

Example showing the options within a specific system

On the following page, you can see the options offered by the EDI software `Lobster_data` for receiving data or obtaining them for processing purposes:

Options in Lobster_data to receive data...



or to actively obtain data on a scheduled basis.

As you can see, the FTP protocol appears twice, both in “Event-based” and as an option that can be selected under “Scheduled”. The same goes for a few other modes of communication. This is because Lobster_data comes with an integral, dedicated, fully functional FTP server. Therefore, a data transfer can be initiated in one of two different ways.

Under option one, your partner can decide that they want to send a file to you. Provided you have set up a suitable user account for them, they open an FTP connection to your EDI server and send you the data. That means your system simply waits for an external party to send it a data transfer to process.


INTRODUCTION

Alternatively, you can agree with your partner that they should make their data available on their own FTP server, where you can retrieve the files yourself. Of course, this process should be fully automated, so you will need to configure a scheduler to manage the retrieval of the files.

If you have noticed that there isn't actually a download command in OFTP equivalent to the "get" (or RETR) used in the standard FTP protocol, read the chapter on OFTP to discover how to get around this issue and create a similar query.

Once the EDI system has completed its work (whether this is format conversion, enrichment, reconciliation, etc.), it needs to send the result of its activity off again:

AMQP	HTTP(S)	SAP
AS2	IoT	SAP ALE
AS/400 DataQueue	LDAP	SCP
Cloud Storage	Mail/X.400	SMS
DB	Man. Upload	SMTTP
File	Message	SSH
Own Class	OFTP	WebDAV
Fax	Portal	
FTP	Rule	



Lobster_data can send the data it creates back out into the world (or your company intranet) using any of these methods.

As you can see: when inputting and outputting data, you will encounter a wide assortment of methods that we have divided up below into either public paths or internal paths. This shouldn't really make any difference to an EDI system, since its sole purpose is to connect the internal systems to the outside world. However, if you can also connect internal systems on both sides then you will have already met one of the most important requirements of an EAI software. Once again, we should note here that any strict divide between EDI and EAI is a purely theoretical construct. In practice it is largely unfeasible to handle them both separately using two independent systems.

→ PUBLIC PATHS

GENERAL INFORMATION

EDI software, in particular, is designed to exchange business data with partners around the world. Once upon a time, VANs – Value Added Networks – ensured that data arrived at their destination reliably and without being accessed by any unauthorised parties. Nowadays, however, we have the open Internet, along with a whole host of communications protocols to protect your data from nosy competitors.

We simply choose the appropriate protocol to suit our requirements. Sometimes speed of response will be important, while at other times we will be more interested in secure transfers or straightforward processing. Over the following pages, we will introduce you to the most common protocols, offer a quick overview of how they work, and weigh up their respective pros and cons. Which one you decide to use will then be up to you! Provided your software can handle your chosen protocol, of course.

→ EMAIL

Everybody is familiar with email, so we don't need to explain what it is here. But we should still take a look at how it works.

Sending emails

When you click "Send" in Thunderbird, Pegasus or Outlook, the program establishes a connection with a mail server that you (or your IT department) will have already specified in your software. The protocol used for this is called SMTP – Simple Mail Transfer Protocol. You can find a detailed description of it on Wikipedia. This protocol is designed to send emails from you to your server, and from there on to many other servers across the world – until they finally end up on the recipient's own mail server. This is where they normally remain until the recipient establishes contact with their mail server and empties their electronic mailbox. That brings us on to the next point:

Receiving emails

The POP protocol – Post Office Protocol – has been around for a long time now. It's currently in its third version, so you will also hear the name POP3 quite a lot. It is comparatively spartan – you can use it to call up a list of emails currently waiting from the server, to download those emails, and to delete them from the server.

Beyond that, it doesn't offer a whole lot of functionality – but for most users this is perfectly adequate. Once you've downloaded your emails to your local drive, you can sort them into different folders (often automatically using filters in your email software) and manage your own data independently.

However, anyone who needs more than that can use IMAP – the Internet Message Access Protocol. On the one hand, this method could theoretically result in emails being left on the server forevermore; yet on the other, they come neatly pre-sorted into folders on the server (similar to a directory structure in your file system), and you can even share your emails with other users – e.g. an entire project team. Although IMAP is not much younger than POP3, there are still many mail servers where it hasn't been implemented yet.

You can find detailed information about the special features of both these protocols on Wikipedia, along with their pros and cons in everyday use. However, we are not primarily interested in emails as they are used in their traditional sense – i.e. for communication between two or more people. We are far more interested in how useful email is for EDI.

Email and EDI

In principle, it's possible to exchange any data via email – including anything that might be sent across the world under the auspices of EDI. However, this would not be our first choice of communication method. The main reasons for this are:

- Some mail servers apply size restrictions, so large attachments could simply go missing.
- Nowadays, emails are typically no longer acknowledged with a confirmation of receipt, meaning that it is impossible to know whether your data has arrived or not.
- There is no guarantee that an email will arrive at its destination quickly – or at all.
- Emails are completely unsecured against access by third parties, meaning that they can be read or manipulated. Of course, you can use software like PGP to at least encrypt and/or sign the content, but more secure paths are available.
- You are at the mercy of your email software – for instance, there are mail servers out there that will take it upon themselves to modify the encoding of texts in attachments.
- Virus scanners and other security measures can delete or block individual attachments, or even entire emails.

The shortcomings of the normal email led to the development of AS1 (Applicability Statement 1), a standard for securely transferring business data via email. However, as this never really caught on among users, we can safely bypass it.

Nowadays, business data are still sent by email – even in EDI, though we should really call this ‘half-EDI’, since in most cases there will be a human at one end of the transfer. For example, invoices might be automatically issued in PDF format and sent via email; or employees working for a small customer may still enter orders manually into Excel files that are subsequently emailed to the supplier, where they are then input into the automatic processing system. In true EDI, where data travel from one system to another without manual intervention, emails are now superfluous. Yet an EDI software should also be capable of sending and receiving data via email as a precautionary measure for processes where humans are still involved. And of course, the software should also ideally support all of the standard protocols (SMTP, POP3, IMAP).

→ FTP

FTP – or File Transfer Protocol – is designed for file transfers, as its name suggests. Most people will have already worked with FTP at some point – e.g. in FileZilla or your Internet browser. You will also probably be aware that you can use FTP from a console (e.g. Unix Shell, or DOS Box in Windows). You can find out more about the operating principles of FTP in Wikipedia.

FTP for EDI

Essentially, we can only do one thing with FTP: transfer files. Users can log into an FTP server and either use “get FileName” to download a file (or alternatively “mget FileTemplate” to download multiple files), or they can use “put FileName” to upload a file (or multiple files with “mput FileTemplate”). All well and good. Once that’s done, we end up with a file stored somewhere – either on a local drive or on the server. So what’s next? Well, the EDI software needs to be able to access the file and process it. In the simplest scenario, the workflow might look as follows:

- A system job (a Unix Cron Job or its Windows counterpart) regularly retrieves files from a particular location via FTP, or alternatively a partner sends them at regular intervals to your FTP server. In either case, the data ends up stored in local directories.

PUBLIC PATHS

- Meanwhile, the EDI software regularly checks the relevant directory for new files.
- If it finds any, it reads and processes them.

And things might work similarly in the opposite direction, too. The processing result by the EDI system might be stored somewhere in a directory ready for another system job to send the data elsewhere via FTP, or for your partner to collect them from you.

This would be a solution for a very simple EDI system that didn't have any FTP capabilities itself. Of course, it would be better if everything happened in one place - i.e. if the EDI software could fetch or receive files via FTP independently. In this scenario, there are two options for inputting data via FTP:

- 1 The EDI software establishes its own FTP connection with a remote server, retrieves the data and processes them.
- 2 A remote client establishes a direct connection with the EDI system (which acts as a FTP server) and hands it the files.

Either way, we skip the intermediate step of storing files locally and then re-inputting them using a scheduled task. Option two even allows us to process the data the moment they are received from the partner. This is a huge advantage when it comes to time-critical information.

Likewise, the return path is also much quicker and simpler if the EDI software being used provides its own FTP client and can output your data without any delay.

Pros:

- FTP transfers data directly to the target system. That means you know if your data have arrived safely, and at what time. Plus, it's almost impossible for a file to get lost en route (as can sometimes be the case with email).
- FTP clients exist on virtually every operating system, so all of your partners will be able to communicate with you.
- If you opt for a binary transfer, you can be sure that your data will arrive untampered with.

Cons:

- You can find out whether your data transfer was successful, but you have no way of knowing whether the EDI system is actually processing the data.
- FTP is not necessarily secure against eavesdropping or manipulation, although FTPS (i.e. the use of SSL during FTP transfers) provides some help with this.
- Firewalls and NATting often interfere with data port negotiation by preventing one system from instructing another to open a particular port, or by preventing the transfer from being routed through the port.
- Decent firewalls will usually identify which port is being negotiated and allow it to open; however, if SSL encryption is being used, the negotiation will go unheard. This can make it much more difficult to use FTP(S), and require a whole host of additional configurations (e.g. defining port ranges, and so on).

This is why AS3, an FTP-based standard to ensure secure data transfers, was designed. It includes features like confirmations of receipt (aka MDNs), but much like AS1 (its email-based counterpart), it never really caught on and it plays an insignificant role in practice.

In short, FTP is much better suited to EDI scenarios than email, and it is used very frequently for these purposes; however, there are more optimal communication methods out there, such as AS2.

→ HTTP

HTTP – Hypertext Transfer Protocol – is familiar to everybody. Without it, there would be no Internet as we know it today. It was originally used to transport HTML pages and images, but has since been greatly expanded and offers a range of possibilities for transferring data of any kind. HTML forms – including with data upload functionality – have long been standard on the web, and files of almost every type are available for download. You can find out more about the history of HTTP and how it works on Wikipedia.

Additional protocols have also been created based on the HTTP protocol that are customised for various aspects of data transfer. Particularly noteworthy examples include WebDAV and AS2, which both have their own dedicated chapters. Here, however, we will look at the use of the original HTTP protocol for EDI purposes.

Browser-based or automatic

The HTML forms we mentioned above, along with download links on websites, will be the most familiar means of transferring data via HTTP for most users. You simply enter some information, perhaps add a file (or even multiple files) for upload, and then click a button to send everything to the web server. After evaluating the information, the server might then respond not only with a website, but with an entire data file that you can save to your hard drive. We probably won't need to elaborate on clicking download links here.

As an example, you might sign in with a certain international postal service (the one with the brown trucks) in order to access specially created webpages where you can download invoices addressed to your company (for instance). Strictly speaking, this doesn't count as EDI, since only one side of the transfer is performed by a system that independently receives data, processes them and makes them available. At the other end, there is a human sitting in front of a browser. And yet...that doesn't have to be the case. Anything that a human can input into a form (especially something that appears in a simple link) can also be entered automatically by a piece of software.

Simple GET requests

If you are capable of clicking a link in order to download or upload a file (something that your browser handles for you anyway), then an EDI system will also be able to obtain data by accessing the same URL: <http://www.example.com/orderresponses/12345.pdf>

Admittedly, this is a poor example of EDI, since PDF documents are unsuitable for EDI processes.

Indeed, the URL given above only contains the address of a document. If you want to send data, they can easily be inserted into the URL too:

<http://www.example.com/formular.cgi?parameter1=value1¶meter2=value2>

This method of sending data to the server is called GET. But it comes with a few disadvantages. For one, it is only possible to insert a limited number of relatively small values into a URL. Some very old web servers only tolerate URLs with a maximum length of 255 characters, and even if that limit doesn't apply, things quickly get confusing with this method.

That makes it effectively impossible to transfer files. And it's also possible for anybody to see the files, since the entire URL travels openly across the web and can even be stored on various other computers for caching purposes. However, when used in conjunction with authentication mechanisms such as HTTP Basic Authentication, it can still be a perfectly adequate way to retrieve a particular customer number along with a certain pre-defined item of data – e.g. all new orders.

A more complicated option: POST

Strictly speaking, we need to distinguish between three varieties of POST:

- **Simple POST:** What would appear in the URL when using GET is now sent to the server as the body of the request via a data stream. As an example: `parameter1=value1¶meter2= value2&File=Long%20text`.
- **Raw POST:** a great option for sending a single file directly to the server. The body – i.e. the data stream – that goes to the server after the headers contains the file content in its original form. There is no need to code any additional special characters here, as in the example of the simple POST where the space character was replaced by `%20` (something that would be especially necessary in a GET, incidentally).
- **Multipart/form-data:** This version could be viewed as a combination of the previous two methods. On the one hand, it is very useful for transferring large files (generally encoded in Base64), just like a raw POST; however, it can also be used to combine multiple parameters, and even multiple files. Much like in an email containing multiple attachments, the body is subdivided into individual parts, each of which contains a parameter. It makes no difference whether those parameters contain just one short text or an entire binary file. The use of MIME types and suitable content encodings allows each part to be transferred in a way that is appropriate to its contents.

In short, each of these alternatives has its pros and cons, and you should be able to decide which one you plan to use on a case-by-case basis. HTML forms generally use either a simple POST (when there is only a small amount of information to transfer) or multipart/form-data when files need to be transferred too. That's why we call it multipart/form-data.

```
<form enctype="multipart/form-data" method="post" ...>
```

Identification

HTTP was originally designed to make information available to the entire world. However, content soon began to appear that was only intended to be accessible to a restricted circle of users. This is why methods were developed to identify the user or system (the 'client') that is accessing a URL. Likewise, sometimes the client will want to make sure that they are accessing the right server rather than one belonging to some phisher or spoofer.

- One easy way of identifying the client to the server is via user names and passwords. In principle, this information can be transferred in one of two different ways, which we will describe in more detail below.
- Conversely, the server can only identify itself to the client by providing a trusted certificate. That happens at the point when HTTPS comes into play – by which we mean the use of SSL encryption in HTTP transfers. No doubt you will have already encountered various dialogue boxes online informing you that the certificate for a given server is untrustworthy, for one reason or another. On a lot of websites, we simply ignore the warning – though hopefully nobody does so when it comes from their bank.
- We can also use the same mechanism to designate the client as an authorised user. To do that, we simply reverse the certificate process, in that the client delivers a certificate to the server for it to review. In scenarios where only a precisely defined group of clients are allowed to gain access, the valid certificate can simply be stored on the server.

The walls have ears

HTTP does not have any built-in encryption. Even HTTP Basic Authentication – the simplest way of logging in to a web server – will encode the user name and password in Base64 during transfer, but it won't encrypt them. As such, if you want to be reasonably sure that no third parties can see your data, you should use an HTTPS connection, which encrypts everything that goes through it in SSL.

Web services

Web services and HTTP don't necessarily have to go together, but in practice, the vast majority of web services are accessed with HTTP, since it works particularly well as a request-response protocol. You send your request to the server and instantly receive the result of the query as a response. Web services could also, in theory, operate over FTP or even email in cases where response times aren't an important consideration.

The most commonly used type of web service is the SOAP request, although some people prefer to use other options such as REST or RPC. Once again, you can find a detailed account of all these web services on Wikipedia. We have opted to briefly touch on SOAP below, which works as follows:

The client (e.g. an EDI software package) sends an XML document containing a request (HTTP POST) to the server. This request could be an inventory request, a room reservation, or anything else that the server is ready to accept. The XML document consists of a SOAP envelope with some header data, and a SOAP body containing the actual application data for the request.

The server (which can itself be an EDI system) receives the XML, evaluates it, puts the result into another SOAP body, wraps it with a SOAP envelope, and sends the whole thing back to the client as a response to the request. You can find a few brief examples of SOAP-XML requests of this kind in the article on EAI and EDI.



Pros:

- The great thing about HTTP is that it's a request-response protocol. Unlike email, where you simply fire off your message and cross your fingers that it will arrive, or FTP, which is only able to confirm that your data have been successfully transferred, HTTP is capable of immediately telling you whether or not your data could be processed. And yes, it can even send back the result of that processing – just like a web service. On top of that, it can do so synchronously – i.e. as a direct response to the incoming request.
- If there is anything that can get through the majority of the world's firewalls, then it's HTTP.

PUBLIC PATHS

- There are countless HTTP servers out there, with the best known being Apache and MS IIS.
- If correctly designed, an HTTP-based interface for downloading and uploading files can be accessed equally by a human working through a browser and by an automated piece of software.

Cons:

- HTTP wasn't originally designed for exchanging important business data. There are mechanisms to for the server to authenticate the client and vice versa, and data can also be encrypted. However, although both of these mechanisms have been individually standardised, their combined use is not. In principle, anyone can pick and choose the techniques that they find personally useful.
- It isn't possible to sign data in HTTP or its spin-off versions. As such, there is no way to prove that the data you receive are genuine, or that they have come from the purported sender.

These disadvantages are addressed by the AS2 standard. AS2 uses HTTP as a transfer protocol, but specifies a range of mechanisms that dictate how authentication between the partners involved in the communication should take place, how data should be signed and encrypted, and so on. AS2 can only be used between two partners who have agreed in advance to do so, and who have already exchanged identifiers and certificates with each other. What's more, AS2 software comes with a quality seal of sorts that guarantees that the software complies with the defined standard. If you want to offer users the option of transferring data to you and receiving it from you either manually or fully automatically, then HTTP is very useful. It would also make sense for an EDI software to provide HTTP interfaces, or to allow for connections with one of the commonly used web servers. If you have high security requirements, however, then it would be better to opt for an EDI system with AS2 integration.



 **AS2**

AS2 stands for **Applicability Statement 2**. Fantastic name, right? Tells you...well, nothing, really. But we can look up what it means.

In RFC 2026 – “The Internet Standards Process” – we learn that:

“An Applicability Statement specifies how, and under what circumstances, one or more TSs may be applied to support a particular Internet capability. An AS may specify uses for TSs that are not Internet Standards, as discussed in Section 7.

An AS identifies the relevant TSs and the specific way in which they are to be combined, and may also specify particular values or ranges of TS parameters or sub-functions of a TS protocol that must be implemented. An AS also specifies the circumstances in which the use of a particular TS is required, recommended, or elective (see section 3.3).”

And TS in turn stands for Technical Specification. I’m sure you’ll agree that that clears everything up. And to think people complain about legal language being incomprehensible! In summary, an applicability statement describes how we should use and combine multiple technical standards in order to achieve a particular functionality online.

Let’s take a practical approach to the topic and ask the basic question:

What is AS2?

AS2 is a transfer method developed specially for the interchange of business data. As such, it places special emphasis on things like security and traceability. Security is achieved through encryption and signatures, while traceability is provided by confirmations of receipt (known as MDNs).

The whole thing is based on HTTP, or HTTPS. This offers the advantage that the response to a transfer can be sent immediately (synchronously) within the same connection. In addition, the HTTP protocol can be applied almost anywhere, and is only blocked by firewalls in rare cases. HTTPS (i.e. SSL or TLS) also offers basic encryption of the entire communication. Unlike basic HTTP, however, AS2 offers only the send direction for data (equivalent to a POST or PUT in HTTP), with no file downloads similar to an HTTP GET.

At most, you receive a brief confirmation of receipt in response.

There's more to it than that, however. As with SSL, additional certificates also come into play. Data receive additional levels of encryption and are signed using the S/MIME standard. On top of that, there is a confirmation of receipt. But let's not get ahead of ourselves.

Interlude: certificates

First, let's take a closer look at these certificates:

In very simple terms, certificates (and we are particularly talking about public key certificates here) work by means of an ingenious mathematical trick known as asynchronous encryption. This process uses two keys in the form of large numbers. Either of these keys can be used to encrypt the data, but the other key is always required in order to decrypt them. In other words, the code used to encrypt the message cannot subsequently be used to decrypt it.

This is quite practical as it means one of the codes can be published (this is known as the public key) so that anyone wanting to send data can use it to encrypt them first. Once encrypted, no one – not even the sender – can decrypt the data. Only you can do so, using the other code that you have kept to yourself (as you might expect, this is then called the private key). This ensures that the message can only be read by the intended recipient.

This process also works in reverse. If you encrypt something with your private key, it can only be made legible again using your public key. In this way, you can guarantee that the data come from you and only you – provided that your private key hasn't fallen into anybody else's hands. This reversal is exploited for the purposes of signing the data – only this doesn't involve encrypting the entire message. Instead, a hash value (a kind of code – the simplest example would be a checksum) is created for the data, and only

this value is encrypted using the private key. The recipient then creates the same hash value for the data (i.e. they run the same algorithm), decrypts the sender's value, and compares the two figures. If the two values don't match, that means either the data have been manipulated, or they haven't come from the purported sender. Either way, the recipient will be suspicious of the message and can choose to reject it.

Certificates contain the public key from a given pair of keys, along with some metadata – whom it belongs to, which server it is valid for etc. The certificate can also let you know whether the message has been certified and by whom, so that you can trust it. All this is essential information in order to understand the principle behind AS2.

How AS2 works

Let's enlist the help of Alice and Bob. Alice wants to send data to Bob via AS2.

During the transfer, the following steps take place:

- Alice creates a hash value for the data and signs it using her private key.
- Alice then encrypts the data (along with the signed hash value) using Bob's public key.
- Alice establishes an HTTP or HTTPS connection with Bob.
- If she uses HTTPS, the entire transfer will have an extra layer of encryption – but we won't go into that here.
- The identifiers (serial numbers) of the certificates used are also sent with the message in various special HTTP headers, along with the AS2 identifiers of the sender and recipient.
- A simple HTTP confirmation of receipt can be sent at this point if the official confirmation will only be sent later.
- Bob decrypts the transfer using his private key. The HTTP header specifies which key he needs to use.
- Bob creates a hash value for the encrypted data, decrypts Alice's hash value using her public key, and compares the two.
- Bob decides whether to send Alice a positive or negative confirmation. He might choose to send a negative one if he suspects that the message has been manipulated, or if the decryption process has gone wrong.
- The confirmation of receipt (Message Disposition Notification or MDN) can be signed using Bob's private key.
- Bob sends the MDN to Alice either synchronously (as a response to Alice's HTTP request) or asynchronously (in a new HTTP connection between them, this time established by Bob).

This process is very neatly represented in a diagram in the Wikipedia article on AS2. Incidentally, if you're wondering what we need the HTTPS encryption for, given that AS2 takes care of all that anyway: AS2 encrypts the data, but not the HTTP header. That means anyone can find out who is sending data to whom and with what certificates (as well as various other bits and pieces of information). HTTPS provides additional encryption for these metadata. Aside from that, the use of encryption and signatures is not strictly mandatory in AS2. You can choose to skip all that, if you want. However, if you don't at least use HTTPS then you might as well publish your data in the newspaper...

The MDN

The MDN is an important component of AS2, and therefore merits a more detailed description. It contains the message ID for the data transfer in need of confirmation. This ID is provided by the sender (in our case Alice) in an HTTP header. So in simple terms, the MDN is a way of saying, "Thank you – I have received your message (no. XYZ), managed to decrypt it, and evaluated it (most likely) as authentic." Or alternatively, the MDN might say that the data were corrupt, an unknown certificate was used, or similar. A negative MDN is known as an NDN.

This information is important for the sender. Comparatively to sending a confirmation of receipt by registered letter, at this point, responsibility for the data is handed over to the recipient. The message confirms that the data have been received cleanly. If something goes wrong after that, the sender is off the hook. In this regard, it makes no difference whether the MDN is sent immediately as a response to the HTTP request that the data came in with, or a few minutes or even hours later by way of an independent transfer. In some cases, the HTTP transfer itself is carried out by one system, while the decryption and signature check are completed later by a separate system. That's why you have the option to use either synchronous or asynchronous MDNs. A synchronous MDN goes out as a direct HTTP response to the HTTP request that the data arrived with. By contrast, an asynchronous MDN is sent back to the sender of the data in a separate HTTP request. However, the recipient can't simply decide that they want to wait a while before sending their MDN – rather, the sender is the one who specifies to the recipient how they should send the MDN. And if the sender requires an asynchronous MDN, they will also simultaneously provide the recipient with the URL to which that MDN should be sent. Any AS2 software that meets the standard will be able to handle both of these alternatives.

What you need for AS2

If you want to exchange data with your partners via AS2, you (and your partner) will need at least the following:

- One AS2 identifier per participant. You can either set these up yourself, or each participant can specify their own.
- One certificate per participant – although different certificates can be used for signatures and for encryption. In this case two certificates per participant would be needed. You can also get by without any certificates, but that wouldn't be in the spirit of AS2.
- The public keys to all the certificates used by your partners.
- And – naturally – a software package capable of handling AS2.

Certificates are self-generated. There are a wide range of software solutions on the market, and a good AS2 software package will be capable of creating certificates. However, your partners may choose to use software that insists on signed certificates (i.e. ones that are marked as trustworthy by Verisign or similar bodies). In that case, the signatures will cost you money. The certificates are then exchanged in advance. You can opt to simply send them via email, since you are only swapping the certificates with the public key.

You should also make sure that your software comes with Drummond certification.

The Drummond Group defines test cases that all AS2 systems taking part in a given round of testing need to be able to handle. Only software that can carry out all the required tasks error-free with every other participant will receive the Drummond Group seal of quality. This is a highly intensive (and expensive) process, and in order to pass, the software actually needs to meet more test criteria than are set out in RFC 4130 (the specification for AS2). Just ask the software manufacturer if they can provide you with a Drummond certificate.



Pros:

- HTTP gives you instant confirmation that the transfer itself was successful (at least through the HTTP response code).
- On top of that, the MDN confirms that the data were received cleanly, and generally also confirms that there were no problems with the decryption and signature check. This can happen synchronously or asynchronously, depending on your requirements.
- The data are encrypted, which ensures that only the intended recipient can read them.
- The signature then guarantees that neither the content nor the sender can be manipulated.
- HTTPS also adds that little bit of extra security on top.
- All the security mechanisms listed above mean that the transfer can take place cheaply over the open Internet, with no need for either an ISDN connection or a VAN.

Cons:

- Good AS2 software (ideally with Drummond certification) is expensive.
- If one of the partners in the communication insists on signed certificates, that will also cost you money.

In terms of expense, you can quickly earn back the investment in your software and in any certificates once you start carrying out a significant number of transfers per month. With ISDN-based processes (such as OFTP) or with VANs (such as X.400), you pay extra for each additional transfer, whereas AS2 communication runs over your standard Internet flat rate. It's up to you to figure out the exact number of transfers that you consider to be 'significant'. However, a lot of businesses are currently migrating from X.400 to AS2 precisely because they can save on continuous transfer costs over the course of many years.



→ SFTP/SCP

SFTP (SSH/Secure File Transfer Protocol) and SCP (Secure Copy Protocol) are two transfer protocols that are both based on SSH and take advantage of the authentication and encryption possibilities offered by SSH. They typically communicate with an SSH server on the server side and converse directly over port 22. Although it may have a similar name, SFTP is only superficially similar to the widely used FTP. While both are used for the purpose of transferring files, on a technical level they have little in common. However, SFTP still offers more possibilities than the more widespread SCP. The latter is only able to copy files (and recursive directory trees), whereas SFTP can also issue commands to rename and delete files or set up directories (in a similar way to standard FTP).

Once again, you can find the key facts about both protocols on Wikipedia. But we are interested in the following:

How they can help us with EDI?

Compared to standard FTP, SFTP/SCP offers two advantages:

- The entire communication is encrypted (though this can also be achieved with FTPS – i.e. SSL-encrypted FTP).
- Only one port needs to be opened in the firewall (usually port 22).



Here's how the ports work:

With FTP, a new port is negotiated for each individual data transfer (whether a file or just a directory listing). When unencrypted FTP is used, firewalls can listen in on the negotiation and open the correct port, but this isn't possible with FTPS. For that reason, we need to go to the trouble of restricting the data ports to a specific port range and opening all of them in the firewall. And when all your networking is outsourced to a service provider, that can be a frustrating process.

By contrast, SFTP simply uses port 22 (or an alternative port if the SSH server is configured differently) for both commands and data transfers. This port is simply left open and you're done.

SCP enjoys the same advantage, except that there are few things it can't do that SFTP can. However, if you don't need those features (or you deliberately want to limit other parties to just copying), then SCP works perfectly well. On the other hand, once the other party has retrieved the data intended for them, they won't be able to delete those data without the commands from Secure Shell itself – and permitting an external party to execute SSH commands on your own hard drive is not always a good idea.

That brings us on to the disadvantage – at least for one half of the communication. Because an SSH server is used to set up the connection and for the encryption, there is a risk – depending on the software used – that a user who should only have access to SCP or SFTP might also log into a Secure Shell and interfere with things on the server. He might have his own account and only restricted permissions, but even so, users still manage to slip through the net from time to time. That can be a real headache for many administrators.

Aside from that, SFTP and SCP naturally come with the same pros and cons as standard FTP (except for the ones that we have already discussed as points of difference). Some relief can be provided by a system with its own protocol integrations. This kind of software can (for example) make an SFTP and SCP server available that doesn't offer any of the possibilities of a secure shell. It can also immediately admit the transferred data for processing, without any of the interim steps set out in the chapter on FTP.

→ X.400

X.400 is a kind of email, albeit a very special one. For the protocol has very little in common with the email format used over the Internet, and there aren't many possible providers either, as is the case with standard email. For example, the only X.400 provider in Germany is Deutsche Telekom, which took over from the federal postal service. That's why X.400 is also known as Telebox400 and BusinessMail X.400, as these are the names under which it used to be offered by the postal service and is now offered by Deutsche Telekom.

As ever, there is an informative Wikipedia article on X.400. Here, we will focus on its practical use in EDI.

The basics

As we have said, X.400 has very little in common with standard email over the Internet. However, it is still an email protocol. That means users send messages with senders, recipients, texts, and sometimes attachments too. These messages are transferred to a mail server, which (sometimes) sends them on to other mail servers, until they eventually land in the recipient's mail box where they are collected by the recipient. That's where the similarities end, however.

As for the differences:

- Not just anybody can run their own X.400 server online. This offers a degree of security, since the messages can only travel via known (and hopefully trustworthy) locations.
- Once a message is delivered into your (or somebody else's) X.400 server, its arrival is confirmed by means of a report.
- In turn, the recipient can confirm to the sender that the message has been received (as was once the case with regular email until spammers ruined everything).
- This means we can class the network of X.400 servers and clients as a VAN, since as well as enabling transfers, it provides a guarantee that not everybody can listen in on the communication.
- As is typical for VANs, this additional service comes at a price. A fee is payable for the mail box, and you pay for every connection and message on top. Even the confirmation of receipt by the recipient comes at a cost, which is why they are only rarely sent.
- X.400 servers used to be accessed via ISDN, but for some time now it has also been possible to contact them over the Internet, including with SSL encryption. The ISDN version costs more money due to the connection fees.
- You can use the FileWork software distributed by Deutsche Telekom or another alternative as your mail client. The protocol should be integrated into any EDI system, since X.400 is very commonly used in EDI.
- There is also a gateway between X.400 messages and online emails, which converts the messages in both directions so that users of standard email can communicate with X.400 customers – however, this also costs money.



X.400 or AS2?

You will probably have noticed how many times the words “that comes at a price” appear in the description above. This is one of the biggest disadvantages of X.400. Its advantage over other transfer methods is its relative security (ISDN or SSL encryption, only trusted servers and participants, etc.), as well as the option of receiving receipt confirmations. However, most other protocols also offer security, and the cost of receipt confirmations means that many users don’t bother to send them.

If you’ve read the article on AS2, then you will no doubt be thinking of MDNs – the receipt confirmations used in AS2. They form part of the standard, and are indispensable. On top of that, the actual data transfer in AS2 is free of charge (aside from your Internet flat rate). That’s why so many organisations are currently moving away from X.400 and towards AS2. The reasons for this shift vary from company to company. Cost saving for transmissions is definitely an important factor, given the many firms that are connected to the Internet via a dedicated line or have installations in computer centres and existing infrastructure is also used in this case. Being in direct contact with communication partners also eliminates the X.400 provider, which also avoids a potential point of failure. Under certain circumstances this can lead to greater reliability.

And as for security: if you rely on sending your data unencrypted over the X.400 network, you may be placing more trust in the server operators than is really wise. By contrast, a fully encrypted and signed AS2 message sent via HTTPS will take quite a bit of effort to hack. On the other hand, as we have already seen, some ‘services’ are fairly deeply rooted in the servers of various different IT companies.

Whether you decide to use X.400 or AS2 (or something else altogether) will naturally depend on your data volumes, as well as what your communication partners are able to support. If X.400 meets your needs well, you may hesitate to switch to AS2 simply as a means of reducing costs, and vice versa. You might even need both of them. As such, before you buy your EDI software, you should make sure it can handle both options and as many other protocols as possible.

 **OFTP**

The ODETTE File Transfer Protocol (OFTP) is a data transfer protocol developed in the mid-1980s, designed to cater to the requirements of the automotive sector:

- Identification of both parties via passwords
- Ability to restart a previously interrupted transfer
- 'Data transparency', i.e. ability to transmit data of any type
- Data compression
- Gateway operation, i.e. transfers can be made from several points right up to the final recipient
- End-to-end acknowledgements when files reach their final destination

From an international perspective, this way of working was initially intended as an interim solution before the rollout of the FTAM standard, derived from the OSI application layer model. Accordingly, it was designed to work over an X.25 network, as it was thought to be the best option for worldwide data exchange at that time. And yet, although FTAM was indeed introduced not long after, the standard never really caught on. This meant that OFTP became the protocol of choice instead.

To deliver the above benefits and the ability to unambiguously identify a file, a concept known as a Virtual File Name was developed, comprising the following elements:

- the Odette or station ID of the sender
- the Odette or station ID of the recipient
- the Virtual File Dataset Name (which is similar to the file name of a file)
- Virtual File Date and Time (date the file was created or first transferred)
- As of OFTP 1.4: Virtual File Counter

When used together, these values create a Virtual File Name that is unique to the file. However it meant that the same dataset name (e.g. 'DLVNOTE4913') and date was used to distinguish between files in many data exchange scenarios. The issue here is as follows:

it was only possible to generate and uniquely name one file per second, which often caused problems as volumes increased. This led to the introduction of the four-digit file counter, which allows for 10,000 such files to be created – per second. In conventional EDI scenarios, this approach to naming files is usually sufficient to avoid overlaps.

Four OFTP file formats have been developed to ensure data transparency:

- **Unstructured (U):** no structure is defined. Instead, the file contains a stream of data. Both parties will have to agree on their meaning and any necessary transformations in advance.
- **Fixed (F):** each record in the file has the same length. 128 bytes are most common when transferring 'old-style' VDA messages.
- **Text (T):** similar to Unstructured, but instead it contains text.
- **Variable (V):** the records in the file can have different lengths.

In practice, Fixed and Unstructured formats are most commonly used. However, as older VDA formats are gradually replaced with new EDIFACT-based counterparts, Fixed is also becoming less and less important.

Over time, the ODETTE project, which is responsible for the protocol, adapted it to current requirements and published these changes in the form of RFCs as freely available standards.

- OFTP 1.2 (VDA 4914/2, 1986) – initial version based on X.25
- OFTP 1.3 (RFC 2204, 1997) – supports TCP/IP for transfers
- OFTP 1.4 (not published as an RFC) – introduced a negative end-to-end acknowledgement and a counter as another addition to the virtual file name
- OFTP 2 (RFC 5024, 2007) – introduced message encryption and signatures as well as signed receipts; enabled the use of TLS for transmissions and defined a public-key authentication in addition or as an alternative to password verification

Over and above the basic protocol itself, a mechanism was developed when defining OFTP 2 that allows communication partners to be sent new certificates if previous certificates have expired or been compromised. Likewise, a catalogue for interoperability tests was also created to serve as a basis when certifying OFTP2 connectors. Today, the use of an Odette-certified connector is often requisite when exchanging data with larger partners via OFTP 2.

Although OFTP 2 is backward compatible with OFTP 1, it took businesses until the mid-2010s to be open to the possibility of making the change. This meant that the comparatively cost-intensive and, when used via ISDN, very slow transmission method prevailed. It was only with the ISDN switch-off at national level (starting in Sweden, driven by the influential firm Volvo), that migration to OFTP 2 picked up speed. OFTP 1 now only plays a marginal role in very specialist applications.

Like other communication protocols, OFTP is based on a challenge-response sequence. In other words: one partner sends instructions to the other and has them confirm in a response whether the transmission was successful or failed. What sets OFTP apart, is that it allows the initiator (partner A) and responder (partner B) to exchange roles during a session. This flexibility is necessary because files or end-to-end confirmations can only be sent by the initiating side – there is no way to download files directly. To reproduce this sequence, partner A has to contact partner B, carry out the registration and then immediately reverse roles. Partner B, who then becomes the initiator, can thus begin to transmit the data queued for transmission. When ISDN was still used, whoever initiated the OFTP connection to transmit data – whether partner A or partner B – also had to consider the costs. This issue is largely redundant today thanks to dedicated lines and any problem that does come up is more likely to be related to the chosen network configuration (firewall etc.) and not the price tag.

→ WEBDAV

WebDAV (**Web**-based **D**istributed **A**uthoring and **V**ersioning) is an extension of HTTP – that in a certain sense – takes it back to its roots. Tim Berners-Lee’s original idea wasn’t for the Internet to be a medium for selling shoes, ringtones and other trinkets or for watching films; rather, he intended it for serious work. Websites shouldn’t be made available to the world to be consumed passively. Rather, they should be actively edited on a joint and cooperative basis. We still find some of that spirit in the form of wikis.

When multiple people jointly edit a file online, they still need the basic options of downloading the file from the server and then re-uploading it in order to save it. However, they also need mechanisms to block other users from accessing the file during editing (to avoid different users overwriting each other’s changes). And ideally version control too, in order to restore previous versions. All this is offered by WebDAV, along with a few other operations for moving, copying and deleting files and working with directories.

WebDAV and EDI

All these nice editing and versioning options are of little relevance to EDI. We are only interested in how our EDI system receives data and sends them back out again. In other words, it’s all about transporting data. With this in mind, when we look at the methods offered by WebDAV (on top of HTTP’s methods), we soon establish that the ones that are relevant to EDI are actually already available in the FTP protocol. What’s more, FTP even offers a few more options. Indeed, even HTTP itself provides us with everything we need in the form of GET and POST.

So what is the benefit of WebDAV? Well, its advantage over FTP is that HTTP can get through almost any firewall. It’s also allows us to avoid the typical problems of FTPS (see the chapter on FTP) – even when using SSL encryption. And compared to simple HTTP, it has the advantage that we can work with and modify a directory structure in a similar way to FTP. Otherwise, it should make little difference whether you dock a WebDAV server to your HTTP server (there are plenty of options here) or a simple CGI script that receives data via POST (downloads via GET should work regardless). Oh yeah, and user rights are another bonus – but FTP has those too.

PUBLIC PATHS

When all is said and done, it only makes sense to use your WebDAV for EDI if one of your partners uses it to provide you with data. For that reason, your EDI software should be able to handle the WebDAV protocol too, just in case.

 **INTERNAL PATHS/SOURCES/SINKS****GENERAL INFORMATION**

EDI systems are used to enable internal and external systems to communicate with one another, while EAI systems focus on internal systems, and ETL/ELT almost exclusively takes place between databases. All three concepts need options for providing your systems – such as databases and ERP systems – with data, or receiving data from them.

Without staking any claim to completeness, we will now present to you at least the most important of these system types and communication forms. There are most definitely other options out there, but this list should cover well over 90 percent of what is available.

And let's not forget that we're also able to communicate with our own computers as we do with others out there on the web. Therefore, we can add most of the communication methods under public paths to our list as well – despite the fact that nobody would actually use OFTP to connect two computers on their own network.

 **FILE SYSTEM**

The file system is of course the most easily accessible option, especially the local one.

We can make data available for processing here and store the results afterwards – provided, of course, that the different systems communicating with each other all have access to the same file system.

However, not all relevant programs will run on the same computer. Neither for EDI or EAI purposes. If they did, they would affect each other's performance – not to mention the risk that all business-critical processes could be knocked out in one fell swoop with the loss of a single computer.

One solution for this is a SAN, or Storage Area Network, where all your important processes can save their data. This generally also offers features like high availability, etc. However, there is also a smaller version: Network Attached Storage, or NAS for short. This is none other than the good old file server, accessed by means of suitable protocols such as NFS or SMB – only the file server itself and attempts to access it are scattered all over the company network.

INTERNAL PATHS/SOURCES/SINKS

The local file system scarcely merits any closer attention, and building a SAN would be too big a topic for us to address here. What we are primarily interested in are the requirements that an EDI/EAI software package needs to meet in order to access drives on remote computers.

NFS

NFS is the **N**etwork **F**ile **S**ystem from the world of Unix. It's fairly antiquated these days, and for historical reasons it doesn't offer especially high security standards. However, the current version 4 represents a significant improvement on the widely used version 3, and has gained a lot of ground on this front.

If you are operating in a Unix/Linux environment then NFS can make your life very easy. Simply by mounting the relevant network drives (ideally automatically while booting), you can access them through your software just as easily as a local partition or directory. In fact, Windows servers can also handle NFS – only this option is rarely used. If the remote computer runs on Windows, you can instead use Samba to mount its shares. This differs little from the equivalent process on Unix/Linux (aside from a few minor operational differences – but that takes us on to SMB).

In other words, if your EDI/EAI system runs on Unix/Linux, that's all it has to be concerned with; strictly speaking, it won't even be able to tell that one directory is local while another is stored on a server in the basement. From the point of the view of the application, access is identical whether the file name is `/user/local/data/File.txt` or `/mount/fileserver/commdata/File.txt`.

SMB

SMB is what Windows mainly uses to create its network file system. Here, things are a little more complicated than with NFS mounts. You are probably already familiar with the famous connected network drive. It is typically represented using letters from the end of the alphabet (in particular the letter T), with the T: drive pointing towards a shared directory on a remote computer somewhere. This is a neat solution for an office computer where the user logs in, the network drives are (usually automatically) connected, and everything is then available and ready for them to work with.

INTERNAL PATHS/SOURCES/SINKS

For server processes, however, which generally run as a Windows service with no user login, this is unworkable. Drive letters like T: always relate to the logged in user, and do not exist for processes that launch automatically when you start up your computer. This calls for a different technique- and you will already be familiar with it from your file manager (aka Windows Explorer). You specify the server and the share name of the directory you are looking for directly, a little like this:

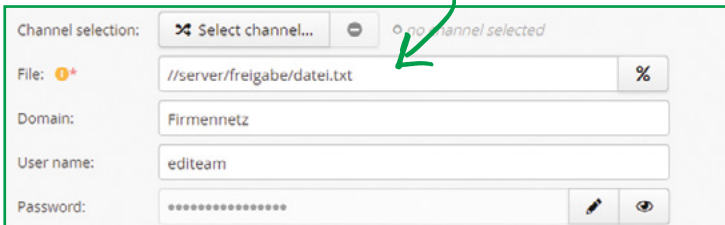
```
\\servername\sharename\path\File.txt
```

This UNC notation allows you to reach every known computer on the network and access all their shared directories. (Incidentally, this also works for Unix, although standard slashes / are used instead.)

However, we still need a user authentication for SMB. Whereas NFS (or Samba) mounts are completed during boot, and identified accordingly, a Windows computer will expect users to authenticate themselves whenever they want to access the shared drive.

One way of solving this problem is to run the software in question on the user account of a domain administrator. That way, it would provide your user's identity whenever it accessed the UNC path. If that identity is known on the remote computer too, it will grant the software access. However, not all system and network administrators will consent to allowing some random program to run in the context of such a powerful user.

The alternative is for the software to log in as a specified user during each access attempt. Here's how that might look in practice:



Details of an SMB authentication in Lobster_data

The minor difference

In principle, programs should be able to access shared directories on remote computers as easily as local drives. Yet there are still a few limitations. The simplest of these is that the server on which the required directory is saved might not be running at the moment, or might be inaccessible due to a network fault; however, this can just as easily happen during access via FTP, for example.

Another limitation is that you shouldn't fall for the illusion that you are interacting with a local directory. Not only is the access time slower, since everything is running over the network, but you also can't rely on the same mechanisms that make your life easier when working with local drives.

Let's take file events as a simple example. You may already be familiar with these. Imagine you've opened Windows Explorer; some process saves a new file or deletes one, and you see the change immediately without having to refresh. The event (file added/deleted) is sent into the system as a message, and the Explorer reacts to it instantly and updates the display. However, this process is far less reliable over the network - perhaps because we don't want to overload the network with these kinds of minor events. The downside being that your software is suddenly no longer able to rely on this mechanism whenever it interacts with a network resource.

Utility for EDI/EAI

Local or network file systems are a simple solution for exchanging data from EDI/EAI systems with other processes; however, we shouldn't forget that although we can provide the relevant system with a file to process, the system still needs to check for itself whether there is any new work (since file events are unreliable, as we have seen). That means the various recipients need to regularly search their directories for new files, similarly to FTP data interchange - though at least on the in-house network we don't need to worry about intercepted or manipulated transfers.

Some simpler software packages offer only one such data interface - or at least only allow you to program one. In that case, you have no other choice. However, if you have access to other ways of ensuring that delivered data is processed immediately, you should take advantage of them.

INTERNAL PATHS/SOURCES/SINKS

→ SAP ALE AND RFC

Before we start, let's clarify one thing: the following article is about the big SAP systems formerly known as SAP R/3 or referred to as SAP ERP Central Component (ECC) or rather SAP S/4HANA in its most recent version. The smaller solution SAP Business One is a separate topic altogether, and one that we won't be discussing here.

You might have already read the section on IDocs above, where we talked about two basic formats: fixed record IDocs, and XML IDocs. The XML variant relates to SAP XI/PI – a kind of interface to the outside world that comes with SAP. Things are fairly straightforward here, with XML as a basic format and e.g. HTTP(S) as a standard communication protocol – although alternative protocols are also possible, of course. Once again, this interface isn't of particular interest to us here, since we have already covered both XML and HTTP in previous chapters.

By contrast, IDocs based on fixed record format are transported via a unique interface provided by SAP called ALE.

ALE stands for Application Link Enabling. It was originally used to connect multiple logically associated SAP installations with each other. Or to quote SAP:

“The purpose of Application Link Enabling is to guarantee a distributed, but integrated, SAP installation. This involves business-controlled message exchange using consistent data across loosely linked SAP applications.”

The interface can also be used for communicating with third-party software, however. The only requirement is that the software must be capable of cleanly handling ALE. If we go down another layer, we find RFC, which stands for Remote Function Call. ALE is ultimately based on RFC, so e.g. the most RFC used to connect SAP and third-party systems is IDOC_INBOUND_ASYNCHRONOUS

There are many more examples beyond these two, however. The bottom line is that third-party software needs to be able to handle not only ALE, but also RFC.

RFC

There are plenty of functions (or function modules, in SAP terminology) that can be called up externally, and if there's something you need that doesn't come with SAP, you can program it yourself with the help of SAP Consultants (and there are plenty of those too). Unless you already have an in-house specialist, of course.

We have already mentioned two of these functions above:

- IDOC_INBOUND_ASYNCHRONOUS can be used to deliver an external IDoc into SAP (such as a new order).
- IDOC_OUTBOUND_ASYNCHRONOUS is what SAP uses to issue an IDoc that will then be converted and transmitted by your EDI software (for example).

There are other important RFCs beyond these, such as RFC_READ_TABLE, which you can use to quickly retrieve a few pieces of data from an SAP system. This is capable of a wide range of things – e.g. you can use it to find out what IDoc types are available in your SAP system, or what extensions to these are known. You can even use it to query their descriptions.

Yet RFCs don't exist solely within the SAP system itself. Connected third-party software (such as an EDI or EAI system) can also provide RFCs, which in turn access the SAP system, allowing this type of communication to also run in both directions.

ALE

ALE is a kind of intermediate layer between RFCs and the outside world. Any third-party software or other SAP installations that exchange IDocs with an SAP system will rely on ALE. ALE masks some of the intricacies of the actual RFC calls that it carries out. And in many cases, the third-party software will be an EDI or an EAI system.

SAP itself refers to “translators” that talk to the SAP system via ALE on one side, and communicate with another piece of software on the other. These translators need to be able to:

INTERNAL PATHS/SOURCES/SINKS

- automatically integrate structure descriptions from IDocs into their own structure definitions,
- accept IDocs from the SAP system and interpret the information according to their own structure definitions,
- provide adequately powerful mapping functionality (i.e. they need to be able to handle the conversion between different data formats),
- hand over the generated IDocs to the SAP system.

In other words, they need to be able to communicate reasonably well with the SAP system and also handle the basic functionality of an EDI/EAI software package. If you are looking for a software package of this kind and you run an SAP system (or you are toying with the idea), you should make sure that the software is capable of handling SAP ALE and RFC and that it meets the above criteria.

SAP already offers whole libraries full of routines that programmers can use for any software that needs to utilise both interfaces. These SAP Connectors are available for a number of different programming languages, including Java and .NET, which are used in turn by the EDI/EAI software manufacturer.

You can obtain an extremely detailed (and fairly technical) account of ALE from SAP, including instructions for programming on the SAP side.

→ DATABASES

Generally speaking, databases don't just stand around doing their own thing; rather, they form the foundation for all the different systems that run within a business. From bulky ERP systems to small payroll programs, there is scarcely any program nowadays that still stores its data in a file system. Instead, most software makes use of the capabilities of a database system. Some programs place very particular requirements on the database, or only work in conjunction with a certain product, while others are more flexible.

The state of the art today is still the RDBMS, or **Relational Database Management System**. These are databases that are interrogated using **SQL (Structured Query Language)**. Other options are also available, such as object-oriented databases, but these lead something of a niche existence.

INTERNAL PATHS/SOURCES/SINKS

Relational databases can be used for just about anything, and they are unbeatable when it comes to storing and managing enormous quantities of data.

Access

In order for a program to manage its data in a database system, there needs to be an interface between the two. On Windows systems, this is usually ODBC (Open Database Connectivity), which is also available on Unix/Linux. A platform-independent alternative is JDBC (Java Database Connectivity), although any software using this needs to be written in Java. Nowadays, any serious database will offer both ODBC and JDBC drivers. Programs can also often access databases via APIs (Application Programming Interfaces), although this requires very close integration between the software and the database, of a kind that really ought to be avoided these days. In theory, if you use the standard ODBC and JDBC interfaces, it should be easy to swap out the database system without disturbing the software connected to it. In theory.

In practice, however, databases don't only support standard SQL; rather, they always come with specific features of their own, ranging from minor differences in the notation of the SQL statements right up to the call syntax used in stored procedures (these are small programs written directly within the database system in a language specific to the database, and which are designed to handle useful tasks). If your software uses any of the specific features of a particular database system then you can forget about any straightforward swaps. However, none of this has much to do with EDI or EAI (or even ETL).

Databases and EDI/EAI/ETL

EDI/EAI/ETL software is designed to link together many different systems, so it shouldn't come with any special requirements in terms of the databases it accesses; rather, it should be capable of accessing as many different database systems as possible. Of course, these EDI/EAI/ETL programs can also store certain things in their own database tables – but because their day-to-day work requires them to be highly flexible, it would be strange if they placed any special requirements on their own data.

INTERNAL PATHS/SOURCES/SINKS

In other words, the systems we are talking about here need to be able to handle ODBC and/or JDBC. If need be, we can also stick on some additional functionality to deal with particularly idiosyncratic databases that might only be accessible via API, but in any case the standard options need to be covered. The tool has to adapt to suit the circumstances within your business, and not the other way around.

However, that doesn't mean you can cheerfully go about connecting your newly purchased EDI or EAI software to databases belonging to all kinds of different programs, and writing whatever data you like into them. Nothing can go badly wrong as long as you restrict your software to merely reading data. More complex programs, however, (let alone something as big as an SAP ERP system) will not take kindly to any external software messing around with their tables. There are simply far too many dependencies to take into account, and you can very quickly end up ruining something. You should use alternative interfaces for this instead, if they are available. You should also make sure you know exactly what you are doing when you set up your ETL processes.

All that aside, however, direct access to databases is actually still very useful for something as simple as lookups (such as obtaining an address to go with a customer number), for example. And because you may well have more than one database system in your company, you shouldn't make it too difficult to access those systems. If you need taking half a day to gain access to an additional database, or if you end up having to reconfigure every last detail whenever you define a new process, then you know something is wrong. It should also be possible to access several different databases with a single process if the data you need is stored across multiple databases.

Once you've accessed the database system, you then need to deal with the tables. That can be straightforward or complicated, depending on your software. The more support your software offers with this, the more you can concentrate on actually defining your EDI or EAI process.

INTERNAL PATHS/SOURCES/SINKS

One last thing: when we talk about databases, we are often talking about mass data. This involves moving around entire lists of product master data, and hopefully large quantities of customer master data too; it involves orders, invoices, delivery notes and much more besides whizzing in and out every second, and being either written directly to the various databases or read from them, or at least requiring huge numbers of lookups to be carried out. And all this needs to run at high performance. Databases are usually streamlined to guarantee performance; however, the program accessing them needs to be able to cope with these torrents of data too. Millions of records need to be read, processed and saved effortlessly.

Conclusion:

Let's quickly summarise what EAI, EDI or ETL software (or software providing all three) needs to offer when it comes to databases.

- Access to (almost) any database via ODBC and/or JDBC
- Quick and easy connections to new database systems
- Straightforward access not only to database systems, but also to the table structures inside them
- Simple methods for reading and saving significant quantities of data, as well as for rapid lookups and smaller insert/update statements – not to mention stored procedure calls
- High-performance processing of enormous volumes of data
- The ability to collect data from different databases and process them together

Even if you don't anticipate any initial need for direct access to databases, you should still make sure that any product you choose meets the above requirements. If you don't, and later find that you need to carry out one of the tasks set out above, then you might come to regret your decision. What's more, databases can sometimes prove extremely useful in EDI and EAI processes as a place to temporarily store processed data and then retrieve them after running additional sorting (e.g. using an "order by" clause).



INTERNAL PATHS/SOURCES/SINKS

→ MESSAGE SERVICES

Message service is a very general term. A message service's sole purpose is to facilitate the interchange of messages between multiple systems which - in principle - don't need to be aware of the other's existence. Each connected system simply sends this service messages to be received by other systems. And any system that wants to receive these messages can sign up to particular channels or topics. The interesting thing here is that the sender doesn't need to know exactly who received the message, and often doesn't even find out whether it was received at all. Nor does it particularly matter how the message is ultimately processed. This constellation uncouples the various systems from each other both logically and temporally. The message is handed over to the message service, and that's it.

Basic message distribution variants

'Channels or topics' is a fairly vague description. In more accurate terms, message services offer a variety of ways to distribute messages. We can broadly distinguish between the following approaches:

- **Queues:** Messages are placed in queues and distributed to interested recipients in order (according to the FIFO principle). There can be multiple possible recipients for each message, but only one of them will actually receive a given message.
- **Publish/subscribe:** Messages are fed into a given pool, and all recipients who are interested in that pool will receive the associated messages. This is sometimes also known as broadcasting.
- **Routing:** Similar to the publish/subscribe process, except that the messages are sent with an additional feature called a routing key. The possible recipients are only interested in messages with particular keys, and in this way, they can filter out the messages that are relevant to them. These keys are entirely freely selectable, and a recipient can also express an interest in more than one key.
- **Topics:** This is broadly similar to routing, except that topics contain more than one value; rather, they consist of multiple parts. For example: "Earth.Europe.Germany" (with a full stop as a separator), or "Politics.Economics.Germany". Recipients can register for the exact topic, or for parts of it - e.g. "Earth.Europe.*" (everything that comes under Earth.

Europe, with the * as a placeholder for a word), "Politics.#" (everything political, with the # as a placeholder for any number of words), or even "#.Germany" (regardless of whether this relates to the geographical or political classification).

- **RPC: Remote Procedure Call.** Each message requires a certain procedure to be carried out on another server, and simultaneously indicates whom the result should be sent to afterwards. Logically enough, recipients register for those messages for which they also offer the relevant processes.

A full explanation of how it works would go beyond the scope of this article. So, if you're looking for a more detailed take, please check out the tutorial for RabbitMQ (<https://www.rabbitmq.com/getstarted.html>), which describes the individual aspects of a message service in different programming languages. We will limit ourselves to what message services are available. And a more interesting question is: what kinds of message services are out there? The short answer is: many. There's the Java Message Service JMS, for example. This is an API – i.e. a programming interface – that provides all the key functions you need to build a message service. One widespread implementation of this is Apache ActiveMQ, but there are several dozen more besides.

But Java isn't the only way to provide and implement message services; you can do so with many other programming languages too, which means that message services are a dime a dozen. And that brings us to a problem: strictly speaking, message services should be as flexible as possible in order to be able to connect any kind of system as a sender and/or recipient of messages. That's why the aforementioned RabbitMQ interfaces are available in over half a dozen languages, for example. But integrating many different programming languages is not the only way to make message services flexible: we can also give them an interface that is accessible not via programming, but through the network. One example of this kind of protocol is STOMP (Simple Text Oriented Message Protocol), while another would be AMQP, the Advanced Message Queuing Protocol. The latter is not a text protocol, but a binary protocol, which means it can do more things than STOMP. Let's take a closer look at it:

AMQP

The real, server-side implementation of a message service is also known as a broker, and the technology behind the broker should be hidden from the outside world. Instead, it should offer an interface via which messages can be delivered to and from clients. AMQP is an interface of this kind. It was developed by a fairly large consortium of companies, and is now also an OASIS standard. One of the advantages of AMQP is that it doesn't cost anything. Anyone can implement and use the protocol in their broker or client.

Another benefit is that increasing numbers of brokers (such as RabbitMQ or Apache ActiveMQ) use this interface. If you already run an in-house message service then there is a very good chance that there will be AMQP available for it. With AMQP, clients and servers (aka brokers) communicate via TCP. It makes absolutely no difference who the party at the other end is, or how (e.g. in what language) they are implemented. In principle it's the same as for web servers and browsers, making it the ideal basis for linking different systems together in a network. And this brings us to the real key point:

Message services and EDI/EAI

You may be wondering what the topic of message services is doing in this guide. Well, in principle, a message service offers an ideal way for systems within a company network to communicate with each other. OK, so SAP might also offer ALE as a direct mode of communication – but that is highly specific to SAP. Other products rely on HTTP, or even on simple data system interfaces, but these forms of communication are only stop-gap solutions, strictly speaking. Whereas HTTP or even FTP are suitable methods for communication between different companies over the Internet (and protocols like AS2 are designed specifically for this), the file system is primarily intended for data storage and not for communications. When it comes to data interchange over a single local network, however, none of these methods serve as suitable solutions – whereas message services are designed for that very purpose. Whether your system is an EAI or EDI one (or can handle both), it always needs to communicate with other in-house products (e.g. over LAN). Things like ESB can help with this – but as we have already said, this is a rather vague concept.

INTERNAL PATHS/SOURCES/SINKS

By contrast, the actual implementation of this communication can very well take the form of a message service – provided, of course, that the software already available in-house is capable of working with message services. In this case, AMQP would once again be a good option.

If you are currently considering purchasing a piece of EDI or EAI software, you should check whether your existing (or soon-to-be-purchased) systems are capable of handling message services. And if you have any doubts, you should take this into account when choosing your EDI/EAI solution.

→ AS/400 DATA QUEUE

Data queues are a concept from the IBM AS/400 (now known as System i, iSeries or i5) designed for rapid communication of processes both on the iSeries and outside it. Although the name queue suggests FIFO processing, in principle there are three possible working methods:

- **FIFO:** First In, First Out – the typical queue concept in which the entries are processed in order of arrival.
- **LIFO:** Last In, First Out; also known as a stack. Here, the newest entry is always read first.
- **Keyed:** Here, the order of the entries doesn't count; instead, they are read via a key that is assigned to each entry in the queue.

Whichever mode is used, there will always be one or more processes that add entries to the data queue, and one or more other processes that regularly query the data queue and retrieve the available entries.

How does that help us?

So what can we do with a data queue in EDI/EAI? Well, the AS/400 has a rock solid reputation and is virtually unbeatable, which is why it's still very popular when it comes to business-critical processes. There are even major companies who wouldn't consider using anything other than an AS/400 for critical tasks.

INTERNAL PATHS/SOURCES/SINKS

On the other hand, AS/400 (and in particular its unique OS/400 or i5/OS operating systems) is in a world of its own. It has its own complete file system (though it is also capable of emulating other systems within the definition applied in Unix), and it can't be accessed externally using any old method. Data are often transferred via FTP – or alternatively you can access AS/400 files directly with SQL, since they take the form of database tables.

However, as we've already pointed out in the chapter on FTP: this protocol actually only transports files. Some degree of scheduling is generally needed if you want to get hold of these files, or process the files you receive. Likewise, you can only query database tables actively, and this typically also requires scheduling of some kind. So if you want to respond to new data (e.g. a new order) as quickly as possible, you will need to search for new files or fire off select statements at extremely frequent intervals. This is a rather dissatisfactory solution.

By contrast, data queues are specifically designed to be checked frequently for new entries. Instead of entering an entire new order that needs to be sent out, we can simply add the order number, which can then be used to read the full order using SQL. This vastly improves performance and also allows us to very frequently check for new orders (since checking for an order number is not very costly) and therefore react very quickly.

As we have already said, these data queues are also accessible to processes that don't run on the iSeries. This access takes place with the help of IBM drivers, such as the IBM Toolbox for Java. Here's what the configuration for this kind of access might look like:



INTERNAL PATHS/SOURCES/SINKS

Host:* 1,2,3,4 %

User:* user %

Password:* ***

DataQueue:* /QSYS.LIB/EDI.LIB./WISSEN.DTAQ

Key:

Check DataQueue (in ms): 1000 Check connection

Use DataQueue value(s) for SQL

Empty SQL result set is treated as error

Custom class:* DefaultAS400DataHandler: Replace ID '&1' with DataQueue va ?

DB alias:* AS400

SQL query:* select dies,das,jenes,from dort where schlussel=&1

Test SQL... Create structure...

Settings for an AS400 data queue in Lobster_data

In this example, the value read from the queue is simultaneously used as a parameter in an SQL statement (placeholder “&1”) The result of this statement can then be subjected to further processing. If you operate an AS/400, this is one way in which you can integrate it into your EAI/EDI concept.

 **MISCELLANEOUS**

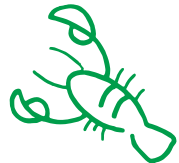
In this guide, we have already described the typical methods that different software systems use to communicate with each other. Of course, there are other options beyond these:

For example, you might encounter a situation where you need to print data out as well as processing and storing them electronically. Classic examples of this include delivery notes, or labels with barcodes. This too forms part of EDI/EAI, although not in the narrowest sense of the term. The software you use should offer the option of contacting the available printers directly (and making the data available for printing in PDF format), or it should offer interfaces that you can use to add your own code.

Interfaces of this kind are very useful in general, as you will often encounter non-standard methods for acquiring or transmitting data. For example, your company might have a custom-developed software that doesn't support any standard interfaces. An interface would allow you to program your own solution. Professional EDI/EAI software should offer you the option of docking your own code in order to quickly establish a connection.



Make sure you obtain proper advice when choosing your EDI/EAI solution!
The experts at **Lobster DATA** will be happy to help (info@lobster-uk.com).



SOFTWARE SOLUTIONS.

Our recommendation!

COMMUNICATION PATHS, DATA SOURCES & DATA SINKS



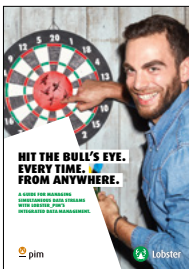
Lobster_data

...lets you transfer business data between different systems and platforms securely and transparently. It handles in-house communications and also effortlessly connects you with external partner companies, such as service providers, suppliers and customers. You can also use Lobster_data to convert business information into structured, editable data ready for manipulation and enrichment via a graphical interface. It's quick, secure and flexible, with no programming required.



Lobster_pro

... is putting an end to dealing with data and processes as separates. Because if streamlined processes are what you're looking for, then there's no getting around a coherent and consistent database. After all - it's what creates transparency and lets you actively manage your process chain. This is why Lobster_pro lets you connect data islands that have grown over time. So say 'goodbye' to the endless searches for the relevant information and 'hello' to displaying and orchestrating business processes consistently without media breaks. This is business process automation at its best. This is digitalisation at its best.



Lobster_pim

...lets you make all the product information in your company available via one central integration platform. Sales, marketing, support, web-shops and external partner companies all have access to the same up-to-date product data, regardless of their location - all in the interests of seamless product communication.

THE COMPACT GUIDE TO EDI

**EVERYTHING YOU ALWAYS
WANTED TO KNOW ABOUT
ELECTRONIC DATA INTERCHANGE ...**

Pöcking, January 2022

All rights to this publication are owned by

Lobster DATA GmbH
Hindenburgstraße 15
D-82343 Pöcking

A catalogue record for this publication is available from the German National Library.
IDN: 1070093181, <https://d-nb.info/1070093181>

Third revised edition

Text: Sascha Raubal
Design: Lobster DATA GmbH
Printing & binding: Stulz Druck Medien, Munich

All rights reserved.

The reproduction of this work in print or electronic format – including extracts – is prohibited.

Electronic data interchange is an essential part of the modern business world. It involves integrating various applications and systems and facilitating communication between them, both within individual companies and between different businesses. To help you make sense of acronyms like EDI, EAI, ESB, ETL/ELT, SOAP and SaaS, we created **“THE COMPACT GUIDE TO EDI – EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT ELECTRONIC DATA INTERCHANGE”**.

This guide presents the standard concepts of electronic data integration, as well as various ways in which it can be implemented. It provides you with a decision-making tool to help you and your company identify where you're at and what you really need. You will also learn about different data formats and messaging standards and discover all kinds of useful things about communication paths and application types.

Use it as a reference guide! It's designed to teach administrators and IT teams the basic facts about EDI, and to provide as straightforward an introduction as possible.



Lobster

Lobster DATA GmbH

Bräuhäusstraße 1

82327 Tutzing

T: +49 8158 4529 300

F: +49 8158 4529 301

information@lobster.de

lobster-world.com