# It's Useful After All - VIN Numbers, DITA, and iXML

Cars today are identified using so-called Vehicle Identifier Numbers (VINs for short). These are usually alphanumeric strings between 11 and 17 characters and are capable of identifying the vehicle's configuration as sold - model year, engine, body - but also the vehicle as an individual. The VIN has a variety of uses, from identifying compatible accessories to vehicle ranges affected by a technical bulletin, and so on. For repair shops, then, the use cases are obvious - it becomes easy to plan a service, from understanding what needs to be serviced and when to acquiring spare parts and consumables in time for the service occasion.

But the VIN can also help locate the service documentation applicable to the vehicle if the documentation is marked up accordingly.

Most car manufacturers today have been around for a long time, which means an ever-growing variety of available models, variants, and configurations, much of it changing from one year to the next and thus requiring a growing set of spares, tools and consumables. Similarly, the required service documentation grows in size as the applcable service procedures and parts lists change or are replaced for each passing model year.

Much of today's service documentation is authored in XML format, allowing for the reuse of shared procedures while also identifying model- and variant-specific tasks. This is done by "profiling" the XML - using markup to state what a specific element applies to. For example, a car model may come in petrol and diesel engine variants, meaning that the service information specific to the engines need to be marked up accordingly but much of the other documentation is shared between the variants.

The DITA specification uses a topic-based approach where each topic is meant to cover a single set of information such as a description of a component, a disassembly of that component, or a parts list applicable to that component. The idea then is that topics are authored for all components that are required to fully build a product. The topics are assembled to full "books" using DITA maps, essentially a method to assemble the topics into the required chapters and sections to provide, say, a complete illustrated parts catalogue.

But importantly, DITA topics - and their individual elements - can be profiled using profiling attributes available for every DITA structure. The specification includes properties such as "product", "platform", and "audience" but can also be extended to, say "model year" or "engine type". We can identify specific topics as applying only to a whitespace-separated list of engine variants, model years, or markets. Or something else, including a combination of profiles. And of course, if a topic is not profiled, it applies to everything.

Technical documentation that is profiled using model and variant information also identifiable using a VIN can then be made to match a VIN that expresses a specific configuration. For example, a disassembly task might have this root element:

```
<task modelYear="2022" engineType="petrol-EU6" drive="LHD" platform="commercial" product="st
    ...
</task>
```

This matches a commercial vehicle, a MY 2022, left-hand drive station wagon with a petrol engine for EU. The matching VIN, however, is a 17-character string that looks something like this: ‘SC6GN1BA?NF123456’.

Here’s where it gets interesting. The VIN string is position-driven, with the various positions encoding vehicle information:

- World Manufacturing Identification
- Vehicle Type
- Body Type
- Engine Type
- Drive
- Check Digit
- Year
- Plant
- Sequential Number

The allowed values are specified for each group and can be expressed as a grammar. EBNF notation, for example, has been used by manufacturers in the past, even though most of them default to an Excel spreadsheet or an ERP system.

For pointy-bracket professionals, however, there is Invisible XML, iXML for short:

“Invisible XML is a language for describing the implicit structure of data, and a set of technologies for making that structure explicit as XML markup.”

iXML allows us to write *grammars* for those various implicit structures and then use an iXML implementation to serialise an instance of that grammar in XML format. Thus, the VIN string as defined for a specific manufacturer can be expressed as an iXML grammar, meaning that a string such as ‘SC6GN1BA?NF123456’ can be output in XML:

```
<vin>
   <manufacturer-id>SC6</manufacturer-id>
   <vehicleType>
      <commercial-vehicle>GN</commercial-vehicle>
   </vehicleType>
   <platform>
      <sv>1</sv>
   </platform>
   <engineType>
```

```
        <petrol>B</petrol>
    </engineType>
    <drive>
        <lhd>A</lhd>
    </drive>
    <check-digit>?</check-digit>
    <modelYear>
        <y22>N</y22>
    </modelYear>
    <plant>F</plant>
    <sequence-number>123456</sequence-number>
</vin>
```

The XML is still little more than an XML version of the VIN, but already much more usable.

But let's briefly go back to that technical documentation. The days of publishing the documentation on paper are long gone. Instead, the documentation is commonly published electronically in a portal application and made accessible to potentially any repair shop, as long as they have an interbet connection and a browser.

The author has designed and helped implement one such portal application for an automotive manufacturer. The vehicle service documentation is authored in DITA and exported as-is to an XML database, eXist-db. A portal application built on top of eXist-db lists the available documentation, provides search and filtering, and allows for on-the-fly publishing of the DITA content. Importantly, the topics are listed and filtered as XML, unchanged, providing direct access to the DITA profiling information.

The user interface for listing and filtering the DITA is done in XForms, with the raw file listings in XML format like so:

```
<collection
    name="/db/test/content"
    created="2023-01-03T10:19:57.112+01:00"
    owner="admin"
    group="dba"
    permissions="rwxr-xr-x"
    uri="/db/test/content">

    ...

    <file
        selected=""
        type="topic"
        uri="/db/test/content/dita-examples/01/second_portal_topic.dita"
        name="second_portal_topic.dita"
        created="2023-01-03T10:20:13.072+01:00"
        last-modified="2023-01-03T10:20:13.072+01:00"
        id="my_second_portal_topic"
        outputclass=""
        dita-content-type="content"
        product="A B"
        audience="D E"
        root-profiles="product(A B) audience(D E)"
        include="true"
        include-profiles="product(B)">
        <title>Topic 2</title>
    </file>

    <file
        selected=""
        type="topic"
        uri="/db/test/content/dita-examples/02/topic_3.dita"
        name="topic_3.dita"
        created="2023-01-03T10:20:12.595+01:00"
        last-modified="2023-01-03T10:20:12.595+01:00"
        id="topic_3"
        outputclass=""
        dita-content-type="content"
        product="A"
        audience="novice"
        root-profiles="product(A) audience(novice)"
        include="false"
        exclude-profiles="product"
        include-profiles="">
        <title>Topic 3</title>
    </file>

    ...

    <profiles>
        <product>
            <value>B</value>
        </product>
        <platform/>
        <audience/>
    </profiles>
</collection>
```

This lists two topics (the `file` element is exactly one listed topic). If you look carefully, you'll note that the first one includes the attribute `product="A B"`. This says that the topic represented by the `file` element and referenced in `uri="/db/test/content/dita-examples/01/second_portal_topic.dita"` applies to products A and B.

The second `file` identifies a topic (`uri="/db/test/content/dita-examples/02/topic_3.dita"`) that applies to product A only, as stated in `product="A"`.

Finally, the above listing concludes with a `profiles` structure:

```
<profiles>
    <product>
        <value>B</value>
    </product>
    <platform/>
    <audience/>
</profiles>
```

This is an XML fragment that is the result of a filter selected elsewhere in the XForm and then applied to the file listing. It essentially says "show only the topics applicable to product B" and if you now check the first of the two `file` elements, you'll spot the attribute `include="true"`. The second `file` has `include="false"`. These are both inserted as a result of applying the `profiles` filter.

Now, that filter - the `profiles` was created using an XForm dropdown list, but can easily be created by other means. For example, given a VIN string, we can serialise that string as XML using an iXML grammar and an iXML engine, and then convert the XML to the `profiles` format above and apply that to the file listing.

This filters the file listing, meaning that we limit the list of DITA topics (and maps; they are not shown above but part of the XML format, too) to only those topics that match the filter. DITA, however, can be profiled on any element, allowing us to mark up fragments of the topic. For example, we might identify a step as only applicable to a specific engine type. If the topic was then published using that profile, the step would be included. If a conflicting profile was used, the step would be excluded.

DITA defines an XML-based filtering format known as "DITAVAL", used by including the filter alongside the topic when publishing. When publishing a listed topic, the portal application converts the `profiles` XML to the DITAVAL format and provides the DITAVAL filter to the DITA publishing process alongside the topic.

Similarly, we can convert the XML-serialised VIN iXML to DITAVAL format and bypass the file listing and filtering functionality.

And finally, we can go in the "other direction" - we can use portal functionality to create a DITA filter, a set of profiles, and convert that set to a series of matching VIN strings. This is useful when wanting to knoww what VIN ranges match vehicles with specific variants, model years, and so on.