Improvement of the meta-model of S1000D

CPF-2015-014FR

Table of contents

References

Table 1 References

CPF-2015-NNNCC_Improvement of the meta-model of S1000D.docx	Improvement of the meta-model of S1000D

Jean-Jacques Thomasson

1 Introduction

1.1 Description

This CPF is issued after presentations of the subject to Steering Committee 22 (2014-02-21 Chepstow) and EPWG 77 (2014-04-08 Paris).

It is the result of the work of a working group, created in 2011, whose objective is to address the following business needs:

- The extension of S1000D to industry specific but standard XML vocabularies (tags sets)
- The connection and reuse of existing data from ie engineering not already using the S1000D structures. IE Documentation provided by small and medium size business enterprises or services companies, documentation of embedded software and firmware.

1.2 Classification

This CPF is to (select all that apply)

Areas	YES/NO
Introduce new capability	YES
Modify existing capability	YES
Editorial	YES
Bug fix	NO

2 Business case

2.1 Business requirement

This CPF is a correct answer to several matters: possibility to simplify S1000D for small and medium size suppliers (that we could call lean or modularized S1000D), possibility for third parties to deliver simple tagged files rather and preferably to PDF or Word files, documentation of firmware using particular vocabularies.

It is to be noticed that this CPF does not modify the current tags set of S10000D. It is only modifying its meta-model. The use of another XML vocabulary spec is still not necessary/mandatory after this CPF is implemented: the S1000D vocabularies covered by its namespace can stay alone independently from any other vocabulary.

This CPF introduces NO change of the current content models of the S1000D elements.

When a new vocabulary is added to S1000D some components names may nevertheless conflict when they are from the same namespace, in which case this will need to be resolved by adding a dedicated namespace name to one of the conflicting vocabularies.

It must also be noted that the proposed meta-model does not open the door to allow for the uncontrolled modification of S1000D vocabularies. The official S1000D vocabulary remains fully under control of the S1000D Authorities.

2.2 Obligatory Questions

2.2.1 Why does the current specification not meet this requirement?

The current specification does not meet this CPF because the meta-model of the current schemas must be modified in order to make them compatible with the new proposed features: extensibility of the vocabulary by use of the mechanism named “domaining” and possibility to add new modules types by use of a mechanism named “specialization”.

Those two mechanisms are detailed further in this CPF.

2.2.2 What benefit does this requirement add?

The main benefit that this requirement adds is:

1. Openness of the model (i.e. easy integration of diverse xml structures within the S1000D structures, for example the common and standard XML tags sets for geographical localizations of objects)
2. Hybrid solution supporting both S1000D and i.e. DITA at a reduced cost.
3. Fall down of the cost of production of the technical documentation.

2.2.3 What are the use cases for this new requirement?

The typical use cases are Flight Manual and documentation relating to software or screens or documentation provided by small suppliers.

For Flight Manual, some constructor would like to have the choice between the ATA2300 model and the crew module type.

Others would need to have an additional optional vocabulary for linking external data.

Another usage of this new meta-model could be a better management of old or new vocabulary. One new vocabulary could be first an optional domain before it becomes part of the core set of elements. The reverse is also true: one set of elements can become an optional domain before it is completely removed from the standard.

3 Proposed conceptual solution

3.1 Solution 1 (preferred)

3.1.1 Concept

In order to explain the concept (design) of the solution we start here from the organization of the schemas of the S1000D as it is today.

The source schemas are currently logically organized around the concept of “libraries of reusable objects” as a key of the overall design. Then the official master schemas are:

- attributeGroups.xsd: This schema contains all the definitions of groups of attributes. Then it is the library of reusable groups of attributes. This schema is, of course, including the schema attribute.xsd which is the library of reusable attributes.

- attribute.xsd: This schema is the library of reusable attributes.
- complexElements.xsd: This schema contains all the definitions of complex elements.
- complexTypes.xsd: This schema contains all the definitions of complex types.
- simpleElements.xsd: This schema contains all the definitions of simple elements.
- simpleTypes.xsd: This schema contains all the definitions of simple types.
- **appliccrossreftable**Schema.xsd, **brex**Schema.xsd, **crew**Schema.xsd, **descript**Schema.xsd…: those schemas contain the specific business logic of each type of module they represent. Here the business logic is the applicability, the business rules (brex), the types crew and descript.

And there are two particular schemas which are used for the inclusion of two standards vocabularies: xlink and RDF.

- xlink.xsd: This schema contains the particular vocabulary of xlink.
- rdf.xsd: This schema contains the particular vocabulary of RDF.

Each of the above schemas is one file.

The new proposition is a reorganization of those schemas with the following key principals: The vocabulary is reorganized in three classes: one named “core”, one named “modules” and one named “domains”.

- “core” contains the core “objects” (element names, attributes, groups…) composing S1000D. This class is very similar to the actual S1000D.
- “modules” contains the definitions relatives specifically to the types of modules. For instance, the typical restriction of the <content> element to contain only <decription>, or <brex>…
- “domains” contains common vocabularies that can be easily activated or removed from the global model. For instance, the tags for the management of extended applicability are typically a domain.

Each class is organized as S1000D master schemas are today: libraries of reusable objects. One library for the attribute, one for the attribute groups, one for the simple elements, one for the complex elements and so forth.

At the end, no definitions of any element are modified by this CPF which is limited to a reorganization of the schemas and their content.

A graphical design has been done in order to illustrate the links that must exist within classes and between classes.

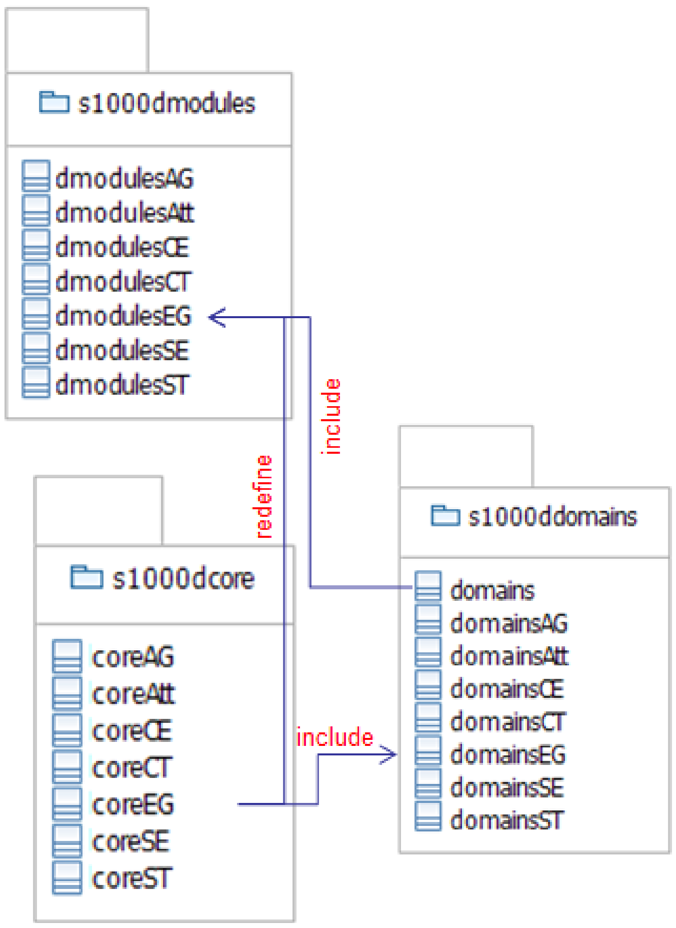


Figure 1: Links between the three classes

Speaking at the class level, the figure 1 shows that:

- Each library of groups of elements (`dmodulesEG`) within the class "module" (`s1000dmodule`) includes the schema "`domains`" of the class "domain" (`s1000ddomains`). It also redefines the library of groups of elements (`coreEG`) of the class "core" (`s1000dcore`).
- Each library of groups of elements of a domain (`domainsEG` in class "`s1000ddomains`") includes the library of groups of elements of the class "core" (`coreEG` in "`s1000dcore`" in the figure)

Inside the classes, there is also a design of the links:

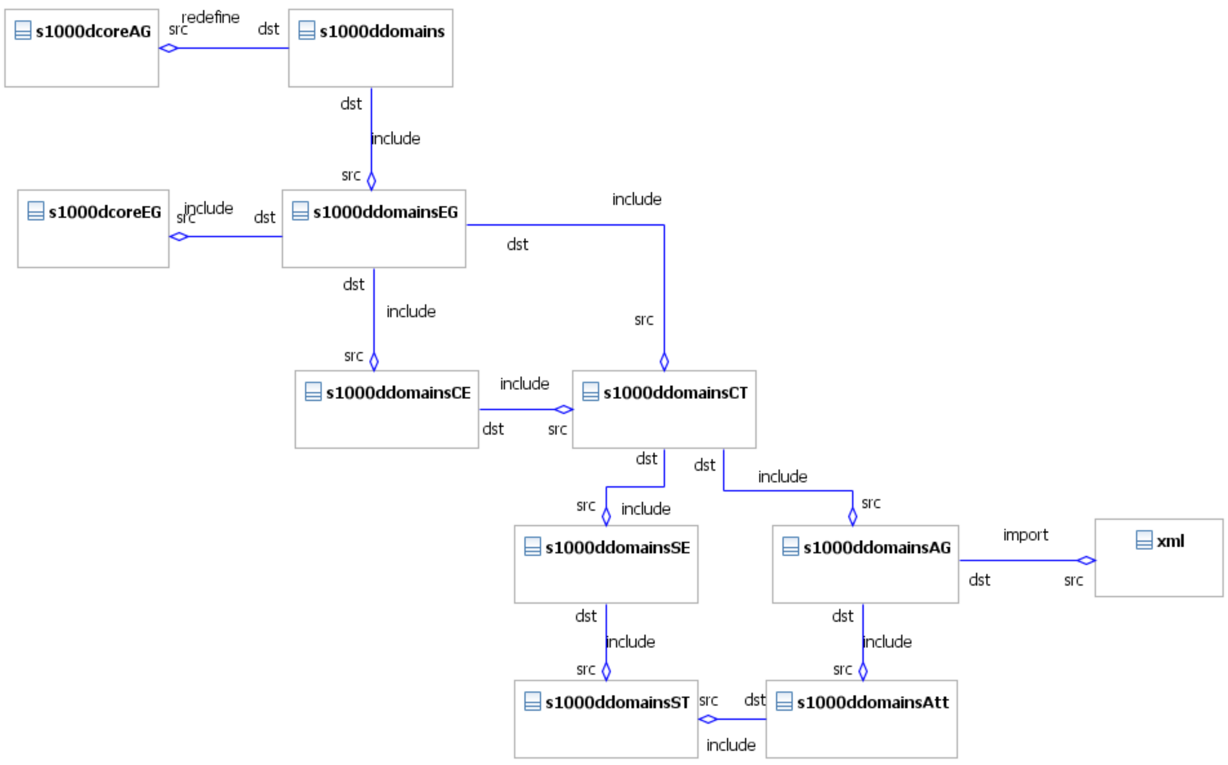


Figure 2: Links between libraries of XML objects of each domain of the class s1000ddomains

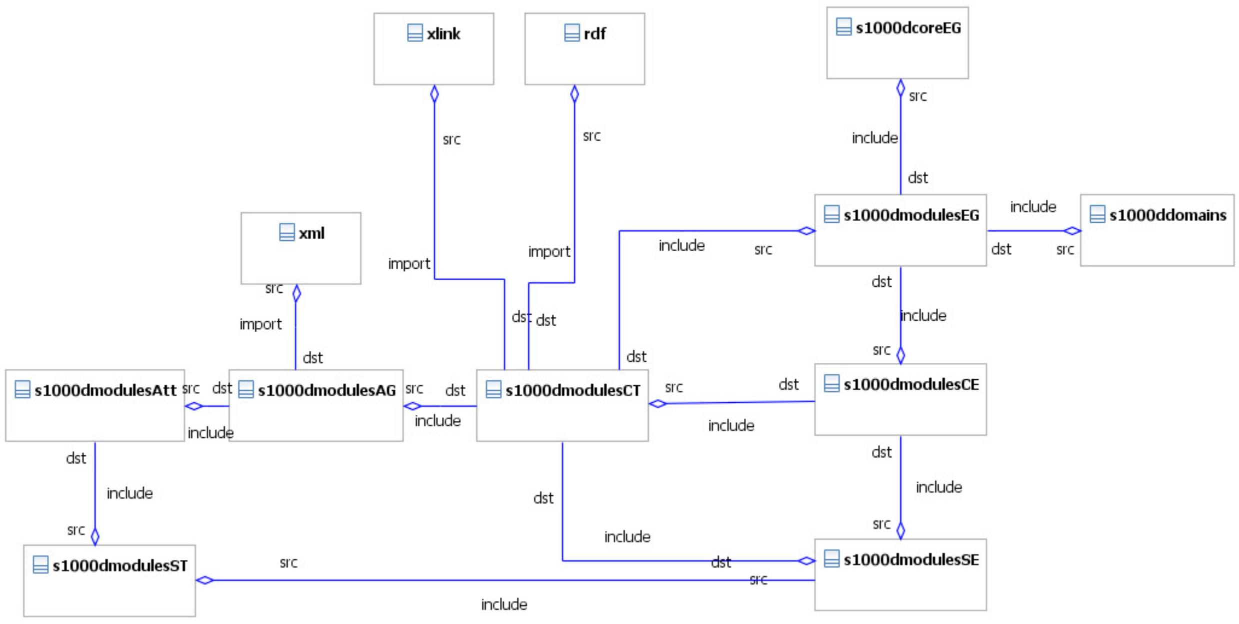


Figure 3: Links between libraries of XML objects of each type of modules in class s1000dmodules

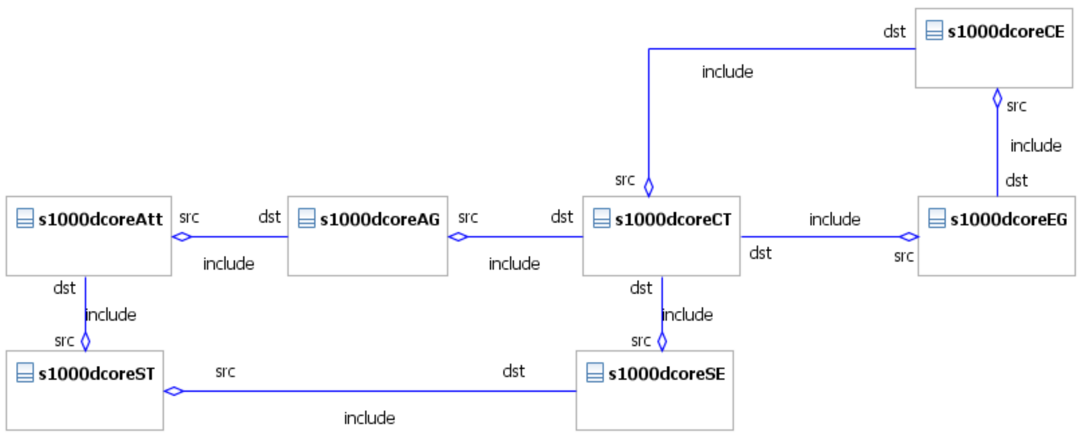


Figure 4: Links between libraries of XML objects in class s1000dcore

This new design has been evaluated, tested and is available today for further implementation.

It is to be noticed that when s1000D is coupled to DITA, some elements' names are conflicting (most of them referring to table construct or ids) and must either be changed or renamed or prefixed by the external namespace. It concerns:

- 27 attributes: id, year, type, class, pgwide, frame, version, valuetype, cols, release, colnum, colname, namest, charoff, char, colwidth, colsep, rowsep, nameend, spanname, morerows, valign, scope, name, width, height and status
- 24 elements: text, task, section, copyright, screen, pre, table, tgroup, colspec, spanspec, thead, tfoot, tbody, row, entry, p, xref, title, step, data, choice, boolean, note and term
- 2 groups: text and status

3.1.2 Traceability to Business Case

By this rewriting of the schemas, S1000D becomes more flexible and extensible.

More flexible:

- Adding or removing a "domain" becomes not only a simple extension/restriction mechanism of class "domains" but also a perfectly controlled procedure.
- Adding a new type of module, also called "specialization", becomes also not only simple but also a perfectly controlled procedure.
- Management of each new release (modification) of the schemas is facilitated.

The technical XML Schema rules that are used (inclusion, import and redefinition) must conform to the designed that is defined by the graphics here after.

And last but not least, this new design opens the path to a connection to DITA[1], as designed in the following graphic:

[1] On his side, DITA must also adopt a new design before this connection can become possible.

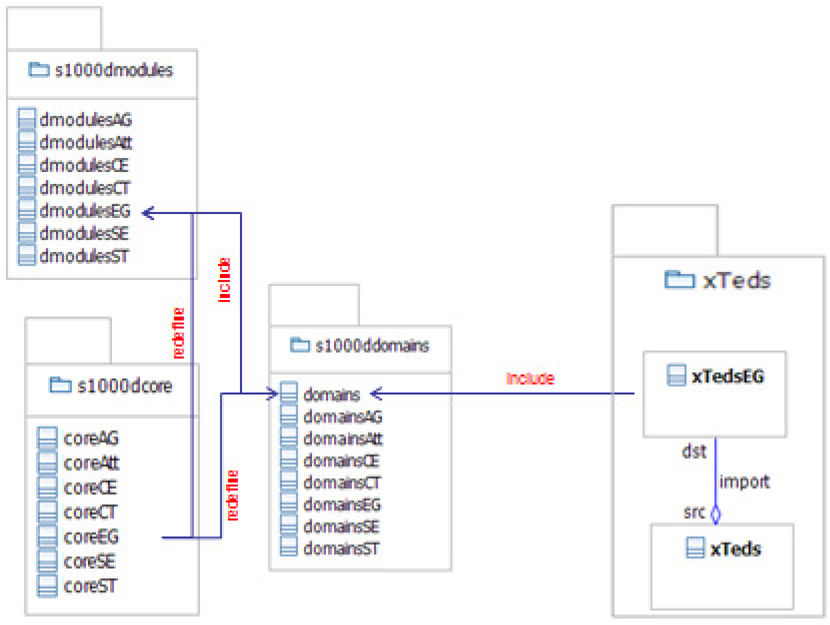


Figure 5: Inclusion of a new vocabulary by use of the "domaining" mechanism: example with xTeds (technical description of transducers)

A similar inclusion can be done for additional vocabularies: sensorML, KML (used for geographical positioning systems) or even DITA.

3.1.3 Effect on the specification

This proposed solution affects

Areas	YES/NO
Chapter text	YES
XML Schemas	YES
BRDP	YES
Default BREX file	YES
Bicycle sample files	YES

Update the List of Chapters in the CPF Viewer

- The chapter list will be identified later, impact on text itself should be reduced.

3.2 Proposed solution 2 (alternate)

None

3.2.1 Concept

None

3.2.2 Traceability to Business Case

None

3.2.3 Effect on specification

None

4 Detailed solution

4.1 Solution design

Screens captures are used as basic illustration of the implementation as pictures are worth more than words to explain these mechanisms.

The top level of the physical implementation of the new model is a directory containing a folder named S1000D. It is clearly physically different from “others” models that it can embed or connect.

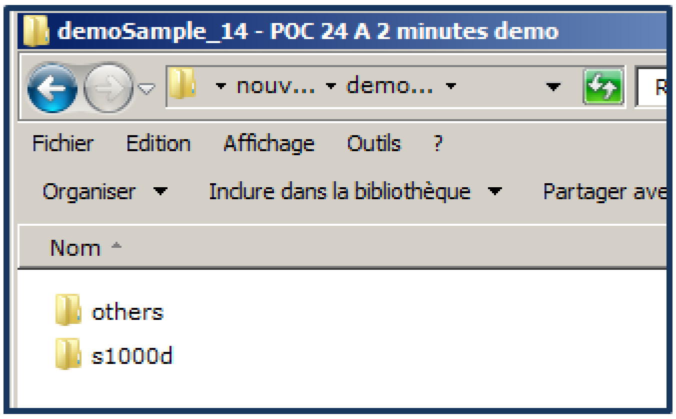
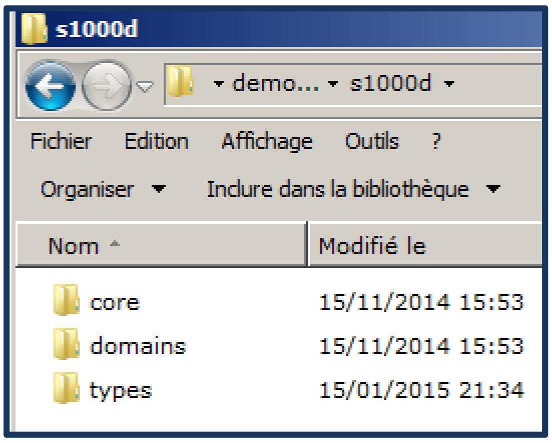


Figure 6: The root of the model, here the folder s1000d, is in any directory

The s1000d directory can have any name, for example “s1000d v5.0”

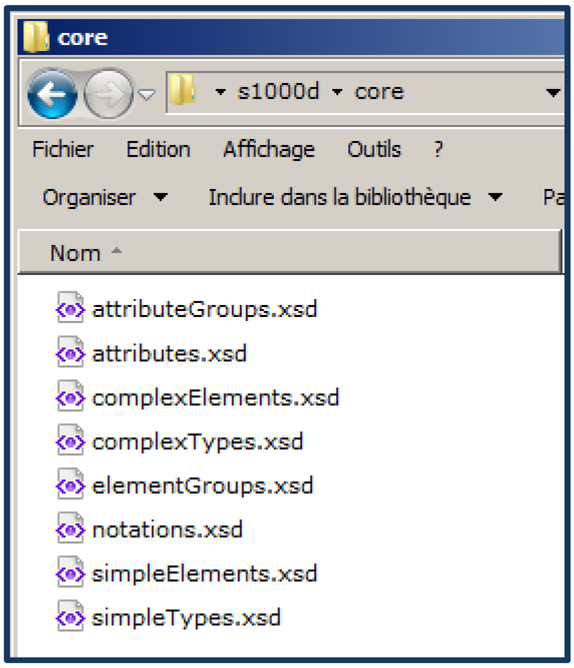
It contains 3 subfolders: core, domains and types.


s1000d
demo... ▸ s1000d
Fichier Edition Affichage Outils ?
Organiser ▾ Inclure dans la bibliothèque ▾

Nom	Modifié le
core	15/11/2014 15:53
domains	15/11/2014 15:53
types	15/01/2015 21:34

Figure 7: The content of the root folder s1000d

The folder `core` contains:

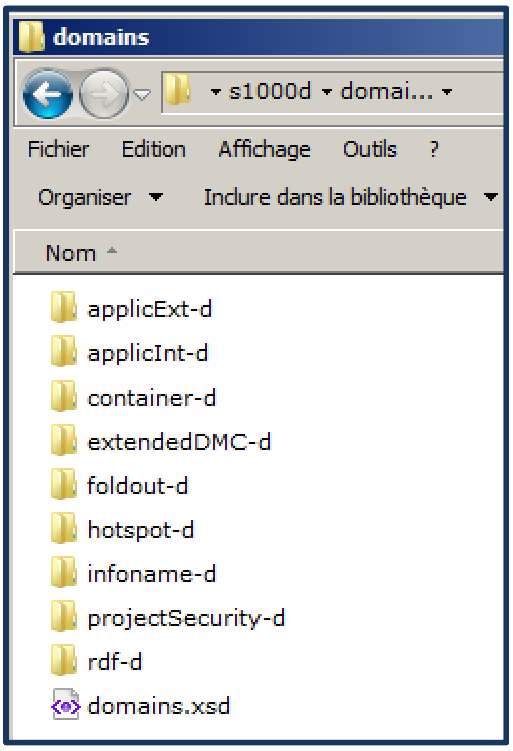

core
s1000d ▸ core
Fichier Edition Affichage Outils ?
Organiser ▾ Inclure dans la bibliothèque ▾ Pa
Nom

attributeGroups.xsd
attributes.xsd
complexElements.xsd
complexTypes.xsd
elementGroups.xsd
notations.xsd
simpleElements.xsd
simpleTypes.xsd

Figure 8: Content of the `core` folder

This is the standard organization known so far and used by the masters' schemas of v4.0, 4.1…

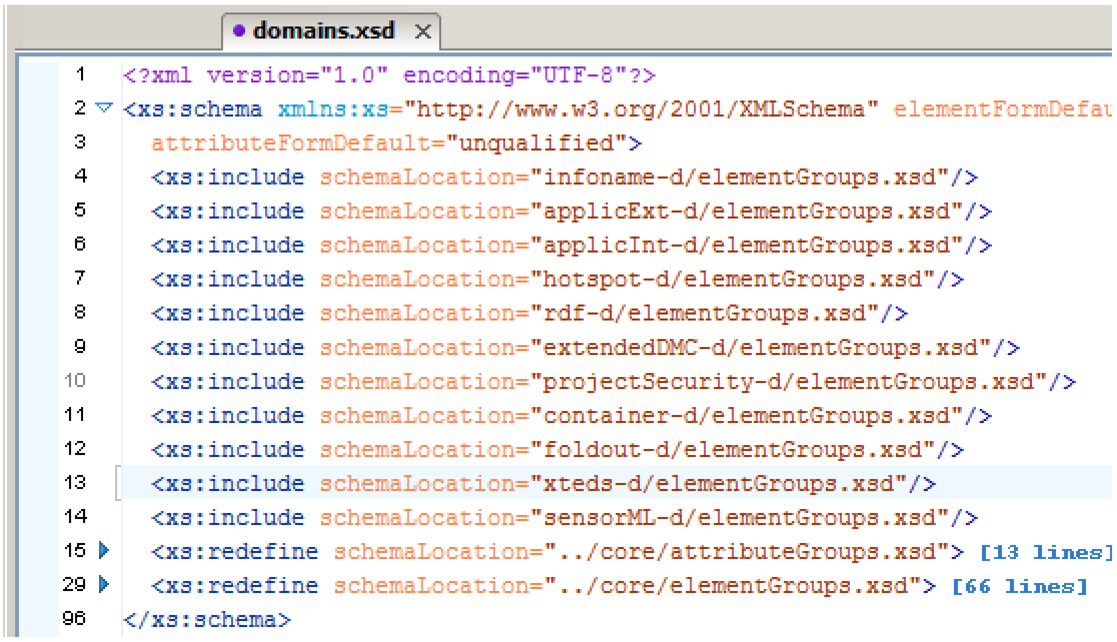
The domains folder contains one subfolder per domain and one schemas named "`domains.xsd`":


domains
s1000d ▸ domai...
Fichier Edition Affichage Outils ?
Organiser Inclure dans la bibliothèque
Nom

applicExt-d
applicInt-d
container-d
extendedDMC-d
foldout-d
hotspot-d
infoname-d
projectSecurity-d
rdf-d
domains.xsd

Figure 9: Content of the folder `domains`

We have created domains for elements that can be subject to modularization: applicability, container, extended DMC, foldout, hotspot, infoname, project security and also rdf. By doing this, those sets of elements are clearly identified and can be very easily included or removed by commenting the corresponding inclusion in the `domains.xsd` schema, as shown here after:

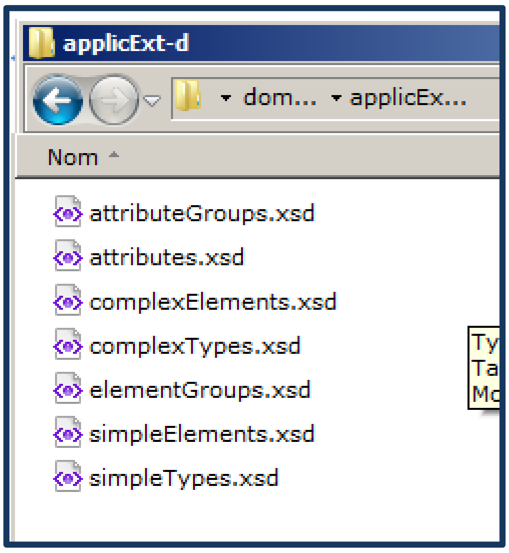


```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefau
  attributeFormDefault="unqualified">
  <xs:include schemaLocation="infoname-d/elementGroups.xsd"/>
  <xs:include schemaLocation="applicExt-d/elementGroups.xsd"/>
  <xs:include schemaLocation="applicInt-d/elementGroups.xsd"/>
  <xs:include schemaLocation="hotspot-d/elementGroups.xsd"/>
  <xs:include schemaLocation="rdf-d/elementGroups.xsd"/>
  <xs:include schemaLocation="extendedDMC-d/elementGroups.xsd"/>
  <xs:include schemaLocation="projectSecurity-d/elementGroups.xsd"/>
  <xs:include schemaLocation="container-d/elementGroups.xsd"/>
  <xs:include schemaLocation="foldout-d/elementGroups.xsd"/>
  <xs:include schemaLocation="xteds-d/elementGroups.xsd"/>
  <xs:include schemaLocation="sensorML-d/elementGroups.xsd"/>
  <xs:redefine schemaLocation="../core/attributeGroups.xsd"> [13 lines]
  <xs:redefine schemaLocation="../core/elementGroups.xsd"> [66 lines]
</xs:schema>
```

Figure 10: Content of the schema `domains.xsd` (here including S1000D and non S1000D domains sucha as xteds and sesorML)

`Domains.xsd` contains one include per domain and redefinitions of core elements when modularization has impact on attributes of the core set of elements.

Each domain contains the classical structure based on libraries of reusable objects:

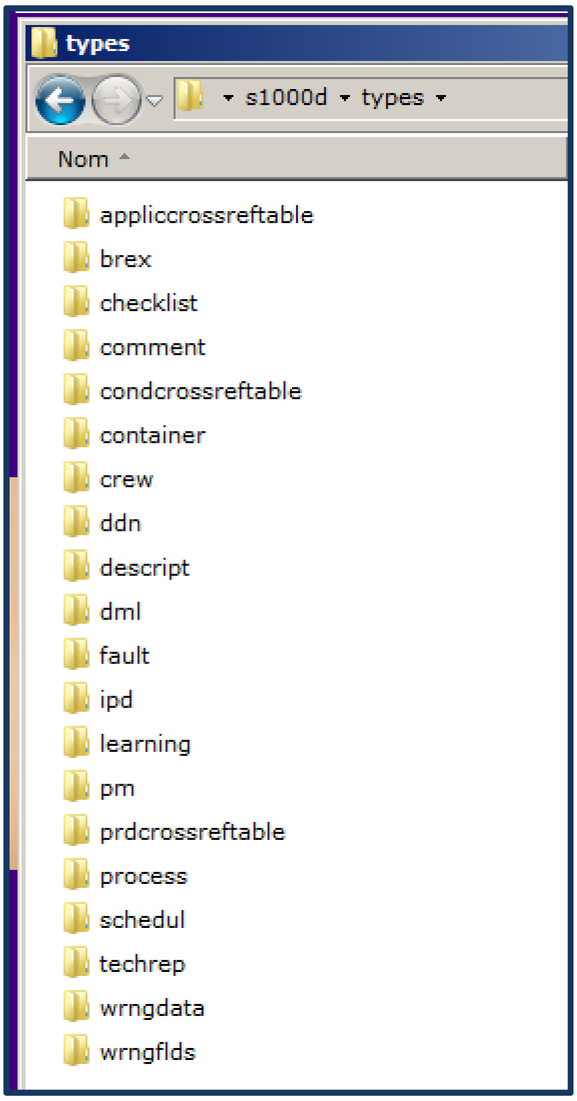


attributeGroups.xsd
attributes.xsd
complexElements.xsd
complexTypes.xsd
elementGroups.xsd
simpleElements.xsd
simpleTypes.xsd

Figure 11: Content of a domain folder (here `applicExt-d`)

A domain folder contains the definitions of objects (attributes, elements, groups, types…) specific to this domain.

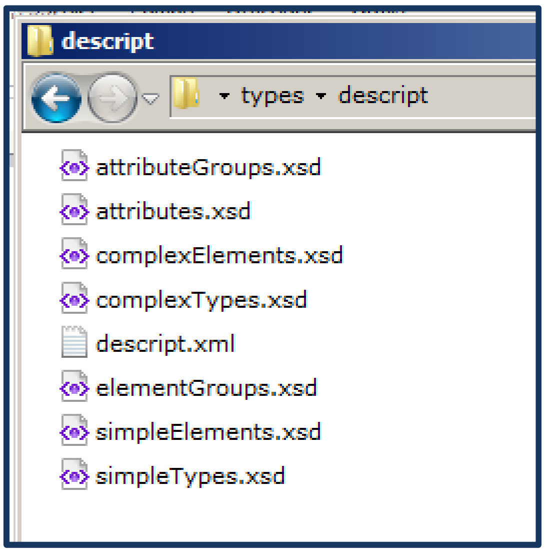
At least, the folder named types contains all the definitions of modules types (`descript`, `process`, `brex`, `crew`…) as shown per figure 12:



appliccrossreftable
brex
checklist
comment
condcrossreftable
container
crew
ddn
descript
dml
fault
ipd
learning
pm
prdcrossreftable
process
schedul
techrep
wrngdata
wrngflds

Figure 12: Content of folder `types`

Each subfolder represents here a type of module whose content is based on libraries of reusable objects:


descript

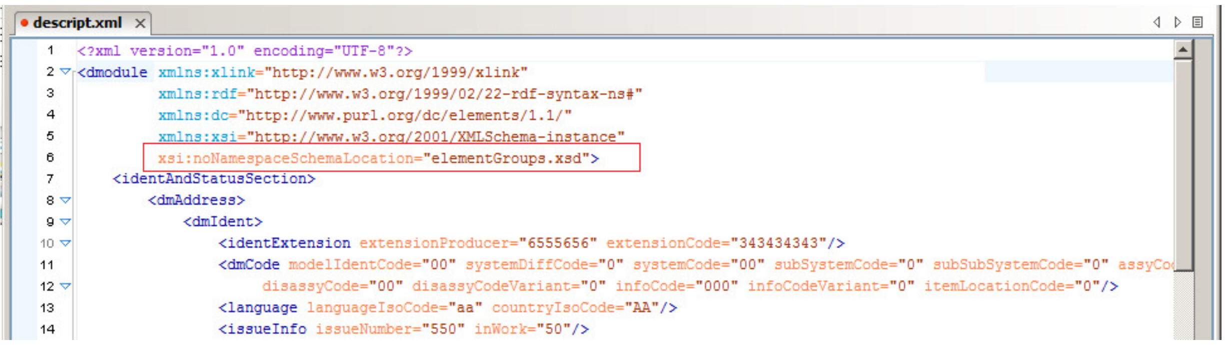
types ▸ descript

attributeGroups.xsd
attributes.xsd
complexElements.xsd
complexTypes.xsd
descript.xml
elementGroups.xsd
simpleElements.xsd
simpleTypes.xsd

Figure 13: Content of folder `descript`

Let's look at the details of the content of the XML and XSD files:

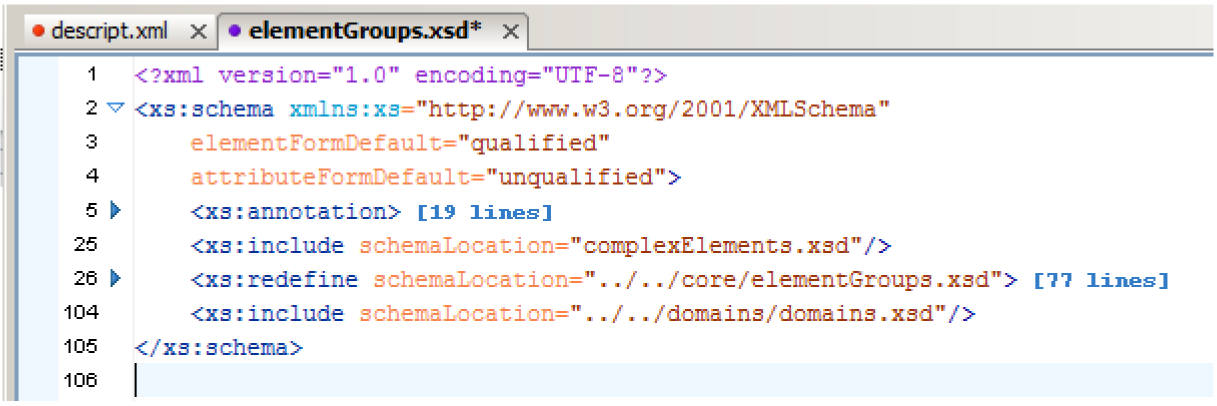
Here, the `descript.xml` instance refers to the schema `elementGroups.xsd` of type descript.



```
<?xml version="1.0" encoding="UTF-8"?>
<dmodule xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc="http://www.purl.org/dc/elements/1.1/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="elementGroups.xsd">
    <identAndStatusSection>
        <dmAddress>
            <dmIdent>
                <identExtension extensionProducer="6555656" extensionCode="343434343"/>
                <dmCode modelIdentCode="00" systemDiffCode="0" systemCode="00" subSystemCode="0" subSubSystemCode="0" assyCo
                    disassyCode="00" disassyCodeVariant="0" infoCode="000" infoCodeVariant="0" itemLocationCode="0"/>
                <language languageIsoCode="aa" countryIsoCode="AA"/>
                <issueInfo issueNumber="550" inWork="50"/>
```

Figure 14: Content of the xml instance `descript.xml`

That schema, on his turn, includes `complexElements.xsd` (the local one specific to type descript), `domains.xsd` and redefines `elementGroups.xsd` from the core class.

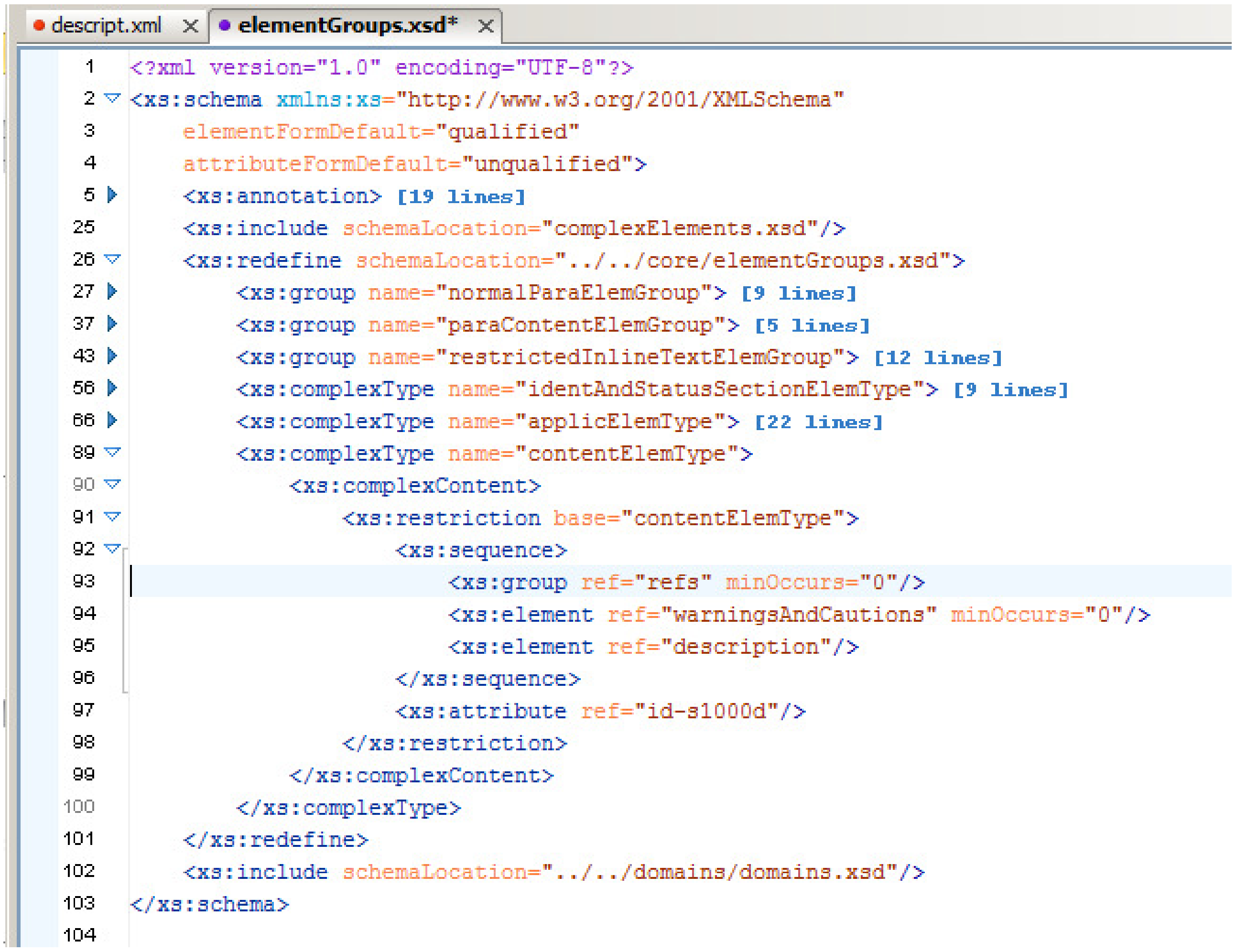


```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:annotation> [19 lines]
    <xs:include schemaLocation="complexElements.xsd"/>
    <xs:redefine schemaLocation="../../core/elementGroups.xsd"> [77 lines]
    <xs:include schemaLocation="../../domains/domains.xsd"/>
</xs:schema>
```

Figure 15: Links of schema `elementGroups.xsd` to other schemas

The purpose of `domains.xsd` is to include, or ignore, all the domains that a business application of s1000D has adopted.

The purpose of the redefinition of `core/elementGroups.xsd` is to redefine the content models that can be specific to the type descript. In particular, the complex type `contentElemType` is restricted to a sequence of the `refs`, `warningsAndCautions` and `descript` elements only.



descript.xml | elementGroups.xsd*

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:annotation> [19 lines]
    <xs:include schemaLocation="complexElements.xsd"/>
    <xs:redefine schemaLocation="../../core/elementGroups.xsd">
        <xs:group name="normalParaElemGroup"> [9 lines]
        <xs:group name="paraContentElemGroup"> [5 lines]
        <xs:group name="restrictedInlineTextElemGroup"> [12 lines]
        <xs:complexType name="identAndStatusSectionElemType"> [9 lines]
        <xs:complexType name="applicElemType"> [22 lines]
        <xs:complexType name="contentElemType">
            <xs:complexContent>
                <xs:restriction base="contentElemType">
                    <xs:sequence>
                        <xs:group ref="refs" minOccurs="0"/>
                        <xs:element ref="warningsAndCautions" minOccurs="0"/>
                        <xs:element ref="description"/>
                    </xs:sequence>
                    <xs:attribute ref="id-s1000d"/>
                </xs:restriction>
            </xs:complexContent>
        </xs:complexType>
    </xs:redefine>
    <xs:include schemaLocation="../../domains/domains.xsd"/>
</xs:schema>
```

Figure 16: Redefinition of `core/elementGroups.xsd` for a specific module type, in the schema `elementGroups.xsd` of the type

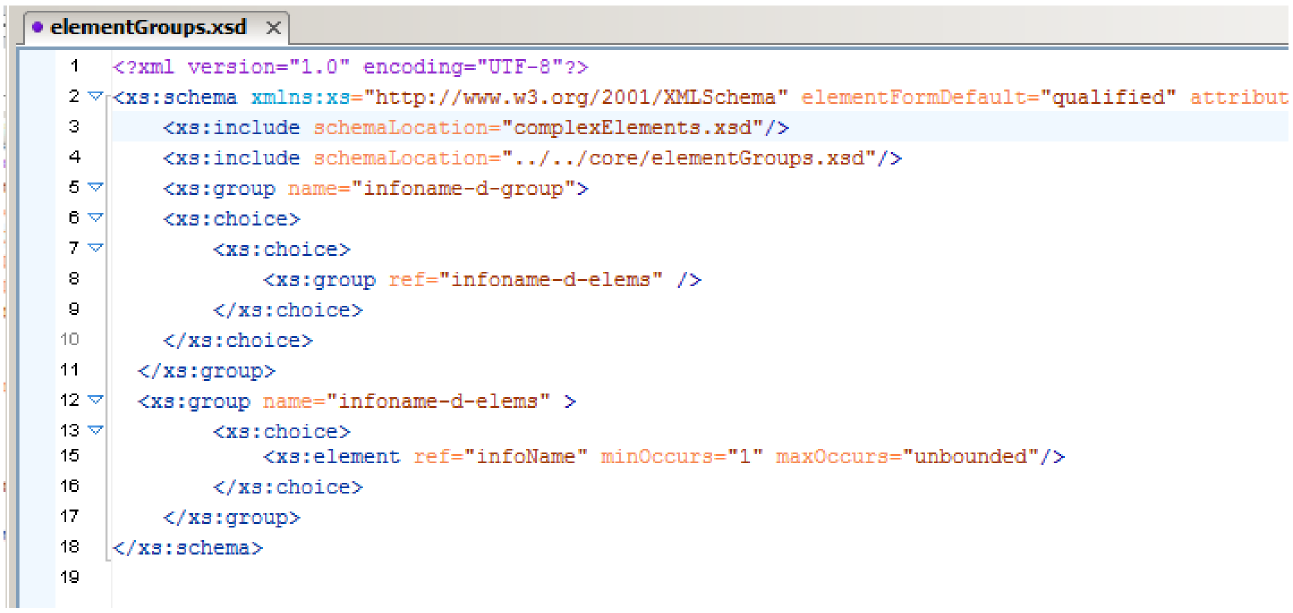
This architecture is exactly the same for any of the module type defined in class, or folder, “type”.

Now let’s look at the class domains (Figures 9 and 10).

The class domains is introducing flexibility in s1000D, providing the possibility to include or ignore some pre-defined tags sets. This mechanism could, for example, participate to the design of a modular s1000D.

In figure 10, we see a series of inclusion of the library `elementGroups.xsd` of each domain (for those that it was decided to include, and not to ignore – otherwise inclusions would be put in comments – , for one specific application of s1000D)

The typical content of such a library is shown in figure 17:


elementGroups.xsd

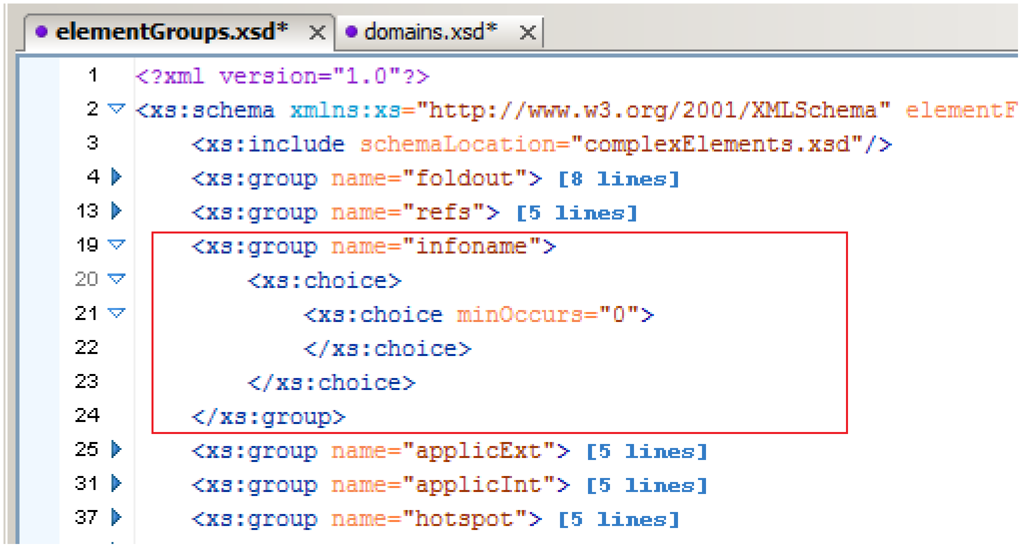
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attribut
    <xs:include schemaLocation="complexElements.xsd"/>
    <xs:include schemaLocation="../../core/elementGroups.xsd"/>
    <xs:group name="infoname-d-group">
    <xs:choice>
        <xs:choice>
            <xs:group ref="infoname-d-elems" />
        </xs:choice>
    </xs:choice>
  </xs:group>
  <xs:group name="infoname-d-elems" >
        <xs:choice>
            <xs:element ref="infoName" minOccurs="1" maxOccurs="unbounded"/>
        </xs:choice>
    </xs:group>
</xs:schema>
```

Figure 17: Typical content of schema `elementGroups.xsd` in a domain

This typical content is made of: a group of elements having the name of the domain (here `infoname`) followed by "`-d`". In our sample, this group has the name "`infoname-d`".

It also contains the definition of a group of the previous domain-d group (`infoname-d`), whose name is domain-d-elems (in our sample `infoname-d-elems`)

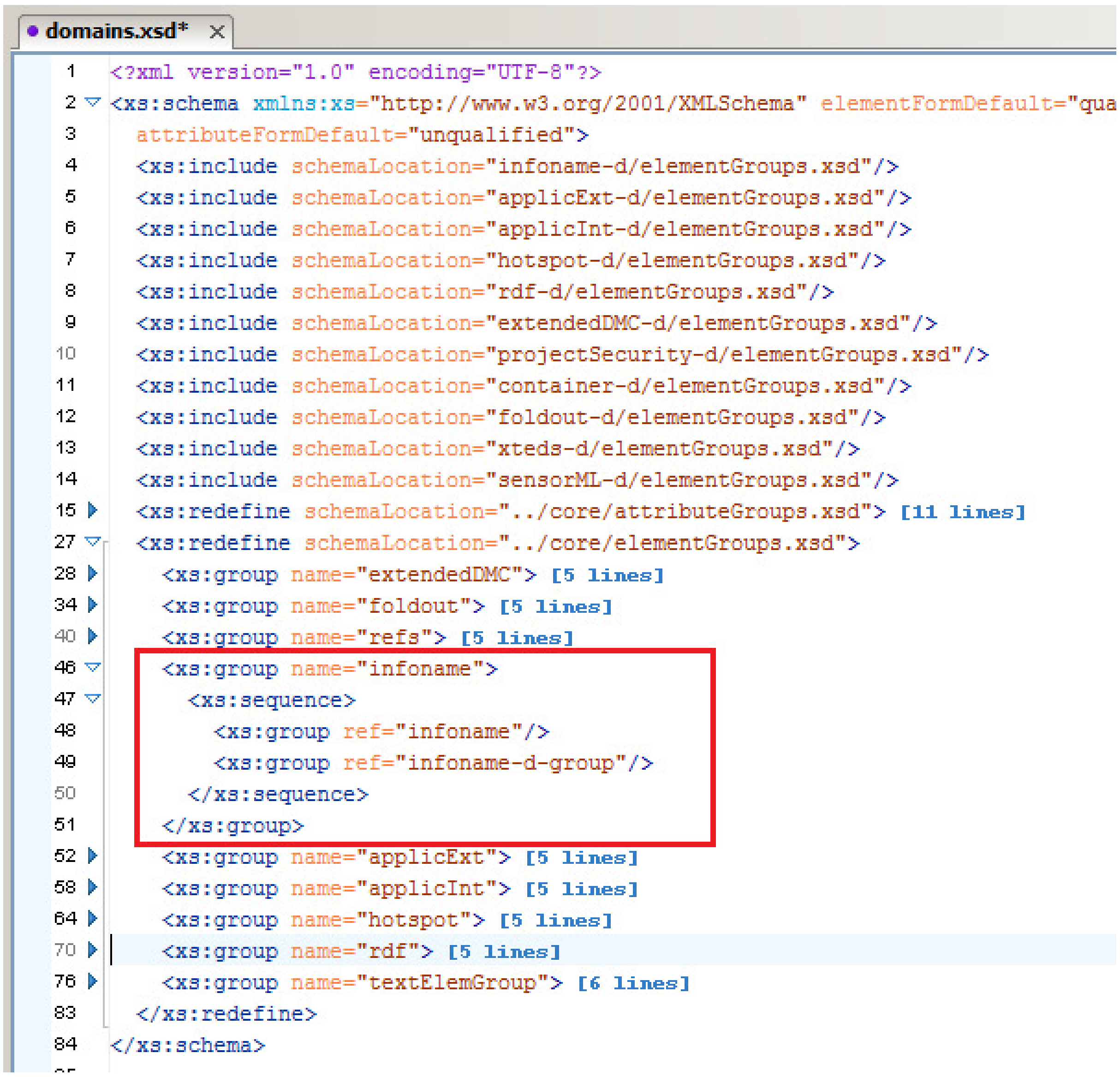
In class `core`, in `elementGroups.xsd`, a group `infoname` is defined but it is empty:


elementGroups.xsd* domains.xsd*

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementF
    <xs:include schemaLocation="complexElements.xsd"/>
    <xs:group name="foldout"> [8 lines]
    <xs:group name="refs"> [5 lines]
    <xs:group name="infoname">
        <xs:choice>
            <xs:choice minOccurs="0">
            </xs:choice>
        </xs:choice>
    </xs:group>
    <xs:group name="applicExt"> [5 lines]
    <xs:group name="applicInt"> [5 lines]
    <xs:group name="hotspot"> [5 lines]
```

Figure 18: initial definition of group `infoname`

In domains.xsd, the group infoname is redefined and now includes infoname-d-elems.



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qua
  attributeFormDefault="unqualified">
  <xs:include schemaLocation="infoname-d/elementGroups.xsd"/>
  <xs:include schemaLocation="applicExt-d/elementGroups.xsd"/>
  <xs:include schemaLocation="applicInt-d/elementGroups.xsd"/>
  <xs:include schemaLocation="hotspot-d/elementGroups.xsd"/>
  <xs:include schemaLocation="rdf-d/elementGroups.xsd"/>
  <xs:include schemaLocation="extendedDMC-d/elementGroups.xsd"/>
  <xs:include schemaLocation="projectSecurity-d/elementGroups.xsd"/>
  <xs:include schemaLocation="container-d/elementGroups.xsd"/>
  <xs:include schemaLocation="foldout-d/elementGroups.xsd"/>
  <xs:include schemaLocation="xteds-d/elementGroups.xsd"/>
  <xs:include schemaLocation="sensorML-d/elementGroups.xsd"/>
  <xs:redefine schemaLocation="../core/attributeGroups.xsd"> [11 lines]
  <xs:redefine schemaLocation="../core/elementGroups.xsd">
    <xs:group name="extendedDMC"> [5 lines]
    <xs:group name="foldout"> [5 lines]
    <xs:group name="refs"> [5 lines]
    <xs:group name="infoname">
      <xs:sequence>
        <xs:group ref="infoname"/>
        <xs:group ref="infoname-d-group"/>
      </xs:sequence>
    </xs:group>
    <xs:group name="applicExt"> [5 lines]
    <xs:group name="applicInt"> [5 lines]
    <xs:group name="hotspot"> [5 lines]
    <xs:group name="rdf"> [5 lines]
    <xs:group name="textElemGroup"> [6 lines]
  </xs:redefine>
</xs:schema>
```

Figure 19: Redefinition of group `infoname` in `domains.xsd`

The above architecture can be generalized to any group of s1000d elements that are considered by EPWG to be worth to group for potential flexibility of the model.

4.2 Chapter text changes

The list of chapter changes is not yet known, but mainly chapter 3, 4 and 7 are potentially affected.

4.3 Technical changes

Impacts on (master) schemas are described above.

Impact on BREX: no (at a first glance).

Impact on Bike Sample: very light (change of the referred schema in the header of the modules)

4.3.1 XML schema changes

See above.