QC / Compliance Observation Record (COR)

RIS Data Management Service, API specifications

PS/PN number:	EU RDMS Platina
WP number:	RDMS-API specs
Log number:	1

Item for review:	RIS Data Managent Service API specification V2.0 dated 12-07-2010 (working document).		
Received from:	IRIS-Europe II, Platina		
Review criteria:	n.a.		
Comments by:	RIS partners	Date:	Sep/Oct 2010
Corrections checked by:		Date:	

No	*Observations*	*Agreed corrective actions(s)*
1	Davor Hrg, 10 sep 2010: I had issues already with xsd for ERI with different styles for eachs xsd there. There are different ways to get same final xml with the xsd, but I had trouble with all of them when generating Java code. It took too much time to get from horrible code to usable code, by adding many rules to code generation engine. This xsd is worse than than from a programming perspective, over 2000 lines of xml definitions, multiple duplicate definitions overcomplicated translation definitions. It took me half a day just to get any code generated... Ibelieve that putting interfaceVersionNrType to every message is unnecessary. Another problem with this version attribute is that there are many elements that have an simpletype version tag. In xml this does not collide but in object code those would have to be on a same object and it is a naming collision of course. There are plenty of improvements to be made from my point of view, many can be done without affecting the resulting xml. One of the proposals is to use global types for everything instead of many inline defined types. Code generation engine can be forced to create top level classses for all of them but then there are many many collisions. This is only the top of the iceberg from what I have seen, if there is interest in improving this I can give further uggestions, as this email is getting too long .	The initial specification contained some implementations issues (“version” attribute was not unique), while using the XSD for implementation/realization of the web service. These version attribute issues (errors) were fixed already during implementation. Corrected version is published: The modified files have been published on the following URL's *URL* wsdl: http://reftool.risexpertgroups.org/refws/v1/RefWebService.wsdl The Data Managementr Tool is about managing separate reference tables. These items are implemented as individual tables (in fact no relations with each other, so it’s easy to add items/other tables). The viewpoint of separate tables is also used as the basis for the WS API design and specification. Furthermore not all the RIS reference and code tables contain the same languages, this is depending on the requirements and the various regulations.
2	Davor Hrg, 12 sep 2010: After few days, a shorter (calmer) version of the previous email.	About the remarks

QC / Compliance Observation Record (COR)

RIS Data Management Service, API specifications

PS/PN number:	EU RDMS Platina
WP number:	RDMS-API specs
Log number:	1

No	*Observations*	*Agreed corrective actions(s)*
	The XSD is OK as far as the requirements go, but I have issue is with the the structure of it. I took the XSD to implement a basic web service client, and the first step is generating Java code. Generating Java code from the provided XSD failed, and while fixing the basic errors. I've noticed unnecessary (but valid) constructs in the XSD. I like having clean code, so seeing the result of the code generation frustrated me to the extreme. Here are some basic suggestions to improve the XSD: -1 use only global types instead of defining types directly inside <xs:element>. -2 rename the duplicate property "version" (it is not duplicate in the resulting XML but results in duplicate property when generating Object code) -3 metadata common to many elements should not bey copy/pasted in the XSD but reused There also are issues with logic that is in the XSD, and moving it to the implementation can simplify the XSD ("use the right tool for the right task") -4 avoid using duplicate definition of same data just to handle mandatory fields (XSD can use the version with the optional fields and implementation can force the mandatory fields depending on usage). -5 generally avoid adding logic to the XSD that forces duplication (implementation should handle such logic) I can provide detailed suggestions if there is still time and will to modify the XSD. If not, this XSD is usable and I will find my way arround it when the time comes.	1) Use only global types: This is about the iversion attribute. It was a design choice to add the attribute directly to the implementation element (function def) of interface function (highest level), instead of the type definition. That is why “iversion” is an attribute of the <xs:element>. Moving the attribute to the Type def means that every element of that type will contain that attribute.The choice was to include this at the highest level (there are also remarks about the necessity of the iversion attribute). 2) Rename duplicate “version” property: Version property renamed to “iversion”. 3) Metadata should not be copy/pasted: The data management tool is designed / based on (individual) reference data tables (so the (design)choice was not to share elements between these tables). 4) Avoid using duplicate defs just to handle mandatory fields: It was required to have a full document style definition including a XSD with validations. 5) The choice was made (see above) to include validation in the XSD.
3	<u>Eric Smets 15 sep 2010:</u> In attachment (below) some general remarks and then more specific remarks about the risindex elements. In the definitions of the reference tool the specfic definitions used for the different sub types should be added or referenced (risindex). Remark the defintion of some key elements (functions) could be also added in the XSD or the examples should contain valid functions with the reference table in the annotation. <u>Attachment info</u>:	Initial a subset of the most used fields of the RIS index is used. It is possible to add extra fields in the RIS index structure. However, the basis used for the RIS index fields, is the RIS idx Excel template. Additional a reference to ENC id’s is already included.

No	*Observations*	*Agreed corrective actions(s)*

	1) All the required functions are probably available in the refTool, to request the data elements and the differences. One of the major issues about the reftool, is that the risindex is more than just some records with attributes and eventually some coordinates. Most of the elements have physical dimensions and are more than just point objects. In order to do so and give other applications benefits about this, one should add geographical definitions as they are used in the GML standards:
	
	ref http://schemas.opengis.net/gml/
	
	2) One should also add some use cases to check if all the data-elements and functions are available in the risindex, based on these use cases. One remark is also that the definitions of the types which are allowed should be described or taken from the ecdis defintions.
	
	Example: How to define if a lock bassin is out of order.
	
	If no definition of lock bassin is given in the risindex, NTS can probably not send some correct references. The result is that in function one needs lock and lock basin, the next question is how is this defined …
	
	3) It is correct that the content of the message is defined outside the reference tool manual, but the XSD definitions are delivered together with this document. As this is a new interface the geografical and structural elements can be added without to much additional cost..
	• Proposal 1: add geo-elements including the EPSG ex WGS 84 = 4326
	◦ GeometryPoint: this is always a point.
	◦ Geometry: This could be a line, multipoint ..., an area depending on the type of Object and could be implemented as a choice.
	
	Example of a Berth geometry
	`<gmgml:GEOMETRY>`
	  `<gml:LineString srsName="EPSG:4326">`
	    `<gml:posList srsDimension="2">50.9371935575031`
	        `5.46592999234001 50.9372388817075 5.46501432795451`
	    `</gml:posList>`
	  `</gml:LineString>`
	`</gmgml:GEOMETRY>`
	
	`<gmgml:GEOMETRYPOINT>`
	  `<gml:Point srsName="EPSG:4326">`
	    `<gml:pos>50.9372162204973 5.46547216036843</gml:pos>`
	  `</gml:Point>`
	`</gmgml:GEOMETRYPOINT>`
	
	• Proposal 2: Add also the reference to the XML opening and closing times of bridged, is missing in the definitions (Optional)
	• Proposal 3: Add waterwayauthority or responsible RISCenter per element
		1) I initial the RIS idx was based on (nautical) reference points related to the fairway (so no need for geo descriptions). Later on, also other objects such as lock were added, so we can understand your question. Maybe the relation to the geo description can be solved using the related ENC fields of that RIS idx.
		
		If there is a necessity to include geo description and/or other additional fields. Then this should first be discussed in the JWT on the RIS-index and approved by all involved.
		
		At this moment we propose to use/manage only the official fields in the RDMS as included at the moment.
		
		2) It is a good point to add/do these checks, but it's a little bit out of scope for this (technical) API specification (may be good item for the encoding guide).
		
		3) See remarks for item 1 (at the present time we support only the official RIS idx datafields asmentioned in the RIS-indec encoding guide?? that are available and agreed).

No	*Observations*	*Agreed corrective actions(s)*
4	Róbert Rafael 20 sep 2010	
	20100822 PLATINA RISDataManagement Service XSD description.doc As I see, this is a generated document, created with help of Altova XMLSpy. I think this document has a very low added value regarding to the interface itself. A few sentences about the goal of all (or at least the most important) types would be much more helpful. I neither want a description of the trivial things, nor the superfluously detailed, for that purpose there is the XSD itself, and the generated document. But some non-generated information would be helpful. Currently the generated document is so lengthy that it is very hard to oversee and therefore currently I cannot provide any valuable feedback.	20100822 PLATINA RISDataManagement Service XSD description.doc You are right, this document is generated by tooling and included for reference only. If the combination API specification and WSDL + XSD are enough then we will delete this additional document.
	20100822 PLATINA RISDataManagement Service API Interface.doc, page 6, figure 1 This is a very good diagram which shows the general overview of the system. Anyway, if you try to „read" it from left to right (as usual in our culture), at first it is hard to understand, as the information flow is basically the right to left (mutations →validations). Therefore I would suggest to "turn over" the picture (vertical mirror).	20100822 PLATINA RISDataManagement Service API Interface.doc, page 6, figure 1 OK, maybe a note will also clarify this. We will add a note to clarify the picture.
	20100822 PLATINA RISDataManagement Service API Interface.doc, pages 6-7, subchapter 1.2 *Within the current RIS Data Management Service environment the Service broker is not used. So RIS Data Management Service clients request the WSDL directly from the Server provider (in this case the RIS Data Management Service server). About the SOAP style to use, the Document/literal (wrapped) style will be used for the RIS Data Management Service . Using this style the Web service interface functions and data structures can be completely defined using a XSD.* I think that web services is general is out of the scope of this document, as web service is a general knowledge. Anyway, the last paragraph (see above) is very important, which indicates which approach will be used in this case from the several possibilities.	20100822 PLATINA RISDataManagement Service API Interface.doc, pages 6-7, subchapter 1.2 OK (action none) The general description is included to be complete and also used as an introduction to define what kind of web service the RDMS will use.
	20100822 PLATINA RISDataManagement Service API Interface.doc, page 13, rows 18-19 *The result list (maximum records) of this function is restricted because of performance up to 200 records*	20100822 PLATINA RISDataManagement Service API Interface.doc, page 13, rows 18-19 The result will be restricted and the first 200 items only will be returned (for the

No	*Observations*	*Agreed corrective actions(s)*
	(result will be limited to specified maximum records). Will paging be implemented? What happens if the number of matching items is more than 200?	mentioned functions). We have to add a recordcount in the result/returned data XML (not defined yet). Using this recordcount the requester can detect whether more records are available. We will add an extra recCount element.
	20100822 PLATINA RISDataManagement Service API Interface.doc, page 17, rows 30-35 *Specific version number to request the data for. Wildcards are possible ("%" will select all versions, also deleted ones). Empty string or not specified then only the actual record(s) are selected (last version). Otherwise the corresponding specified versions are selected (even if they were deleted).* I think that somehow the latest version should be handled with a specific code, e.g. LATEST or HEAD. This can be used for cases when the client wants to retrieve all the actual data.	20100822 PLATINA RISDataManagement Service API Interface.doc, page 17, rows 30-35 This specification text will be modified as follows: - 0 (zero) is wildcard, will return all versions (also erased ones). - not specified, is no specific version and will return only the latest (actual) one. - any other value >0 will return that specific version (if available).
	Typo's: • 20100822 PLATINA RISDataManagement Service API Interface.doc ○ Page 8, row 6: I think now it is better to use future tense (*will be*, instead of *can be*). ○ Page 10, row 36: close bracket (")") is missing at the end, before dot ("."). ○ Page 10, row 42: ***E**ach* (instead of ***e**ach*; capital E). ○ Page 20, row 1: ***other** tha**n** country specific* (instead of *the**n** country specific*). ○ Page 20, row 19: *re**q**uested* (instead of *re**g**uested*)	Typos will be corrected!
5	Davor Hrg 22 sep 2010 first suggestion there are 19 usages of <xs:extension 14 of those are extensions to add version attribute: <xs:attribute name="iversion" type="tns:interfaceVersionNrType" use="required"/> It is done in a way that displays no benefit from using extension	This is about the iversion attribute as mentioned before. It was a design choice

No	Observations	Agreed corrective actions(s)

	construct.
	
	To explain the problem I'll focus on one of them only (the pattern is the same for all 14).
	
	we have matchByCode element
	`<xs:element name="matchByCode">`
	that extends matchByCodeType only to add "iversion" attribute
	`<xs:extension base="tns:matchByCodeType">`
	`   <xs:attribute name="iversion"`
	`type="tns:interfaceVersionNrType" use="required"/>`
	`</xs:extension>`
	
	The matchByCodeType contains few elements
	reftype,code,version
	and is not used anywhere else in the xsd to be extended or otherwise.
	matchByCodeType is translated to object:
	`class MatchByCode {`
	`   ReftypeType reftype;`
	`   String code;`
	`   Long version;`
	`}`
	
	The extension creates an additional class in object mappings which is wastefull and useless.
	`class MatchByCodeType extends MatchByCode{`
	`   String interfaceVersion;`
	`}`
	
	Removing the extension and moving the attribute to matchByCodeType
	would result in singe Object for this.
	`class MatchByCode {`
	`   ReftypeType reftype;`
	`   String code;`
	`   Long version;`
	`   String iversion;`
	`}`
	
	This is repeated 14 times, so we would have 14 classes less, or 14 files less complicated code.
	
	This change would not affect the resulting element, just the resulting Object structure where XML is mapped to objects.
	This change brings no relevant change to the XML but is intended as improvement for those implementing it.
	
	If there is a reason not to do this, I'd like to know, so I can consider or argue it.
	
	OLD XSD CODE
	-- 	
		to add a version nr to each WS function and to add the attribute directly to the implementation element (the function def) instead of to the Type def.
		The iversion nr was included to be sure that the caller uses the correct endpoint (especially when different versions of the WS are operational).
		If we agree that this check/restriction is not needed (on the server side) or could easily be worked around (so check is useless), then we can remove this attribute. Removing this attribute will also resolve a lot of the other issues as mentioned by D. Hrg.
		
		We will remove the **iversion** attribute!
		Assuming the different endpoint url's (incl version id) will be enough to support different versions of the WS and that extra version checks if clients are using the correct endpoint, are not needed.

No	*Observations*	*Agreed corrective actions(s)*

```
   <xs:element name="matchByCode">
      <xs:annotation>
         <xs:documentation>Function decl: Find information based
on a Code (lookup code)</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="tns:matchByCodeType">
               <xs:attribute name="iversion"
type="tns:interfaceVersionNrType" use="required"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
...
...
   <xs:complexType name="matchByCodeType">
      <xs:annotation>
         <xs:documentation>type decl for matchByCode()
call</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="reftype" type="tns:reftypeType"/>
         <xs:element name="code" type="xs:string"/>
         <xs:element name="version"
type="tns:refrecVersionType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
---------------------------------------------------------------------------

NEW XSD CODE
---------------------------------------------------------------------------
   <xs:element name="matchByCode"
type="tns:matchByCodeType">
      <xs:annotation>
         <xs:documentation>Function decl: Find information based
on a Code (lookup code)</xs:documentation>
      </xs:annotation>
   </xs:element>
...
...
   <xs:complexType name="matchByCodeType">
      <xs:annotation>
         <xs:documentation>type decl for matchByCode()
call</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="reftype" type="tns:reftypeType"/>
         <xs:element name="code" type="xs:string"/>
         <xs:element name="version"
type="tns:refrecVersionType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="iversion"
```

No	*Observations*	*Agreed corrective actions(s)*
	type="tns:interfaceVersionNrType" use="required"/> </xs:complexType> --	
6	Davor Hrg 22 sep 2010: Another suggestion. Although I have sent a modification that involves interface version number that is sent back and forth, there is an issue that might make that change obsolete. My point is that interface version number is not needed and is redundant useless information. as WSDL clearly states the url for the service will be : /refws/v1/RefWebService <service name="v1/RefWebService"> <port name="RefWebPort" binding="tns:RefWebPortBinding"> <soap:address location="http://reftool.risexpertgroups.org:80/refws/v1/RefWebService"/> </port> </service> when new version is created my guess that the url will be different, for example: /refws/v2/RefWebService <service name="v2/RefWebService"> <port name="RefWebPort" binding="tns:RefWebPortBinding"> <soap:address location="http://reftool.risexpertgroups.org:80/refws/v2/RefWebService"/> </port> </service> I definitely would have different version on a different URL. With these facts, if two versions of the web-service would be active clients would connect to different url and different implementation would handle each URL. Hence there is no reason for interface version number to be sent back and forth, it is redundant. I can imagine PHP code that handles both versions by simply adding small logic to cover the difference in the XML structure. But the same php code could be easily be rewritten to use the url instead of the version number inside the XML itself to check the interface version.	See corrective actions at Observation #5. (iversion attribute will be removed).
7	-	-
8	Impl team (Linda), remarks during implementation 27/9: API functions (matchByCode and matchByName), version	Correct the function (version parameter) descriptions in de API sepcs as follows:

No	*Observations*	*Agreed corrective actions(s)*
	element could not contain "%" or could not be empty according the XSD.	-Version = "0" matches all versions of that records also the deleted ones. -Version item not specified (its optional) matches only the latest version (=actual record). -Other values = retriev that version (if available) even if its deleted.

9	Davor Hrg, 28 sep 2010:
	Dear All,
	another suggestion,
	remove empty <xs:extension and replace with type="" attribute.
	By looking at complextype:ris_idxType and complextype:ris_idxReqMutType
	it is clear that both are used and considered same.
	
	complextype:ris_idxType uses for risIdxCode:
	`<xs:element name="risidxCode">`
	`  <xs:annotation>`
	`    <xs:documentation>Be sure to ...</xs:documentation>`
	`  </xs:annotation>`
	`  <xs:complexType>`
	`    <xs:simpleContent>`
	`      <xs:extension base="tns:risCodeType"/>`
	`    </xs:simpleContent>`
	`  </xs:complexType>`
	`</xs:element>`
	
	and complextype:ris_idxReqMutType uses for risIdxCode:
	`<xs:element name="risidxCode" type="tns:risCodeType">`
	`  <xs:annotation>`
	`    <xs:documentation>Be sure to edit, </xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`
	Empty <xs:extension appears only 5 times, and this suggestion is just for cleaning xsd.
	This also means another 5 Java classes less, that xsd<->object binding would generate.
		This extra level of complextype is wrong (types in both ris_idx structures must be the same).
		Will be corrected.

10	Michal Chochula 29 sep 2010: We have checked xsd and wsdl you provided also the topics identified by D. Hrg We agree to his proposals in all topics: 1 Avoid xs:extension for iversion 2 Interface version number is not needed 3 Remove empty <xs:extension	See remarks of D. Hrg for the agreed corrective actions.

No	*Observations*	*Agreed corrective actions(s)*
	4 Duplicate data types 5 Duplicate definitions for translations Besides those, we do not have other objections or proposals for amendment/improvement of provided xsd/wsdl .	

11	Ron v/d Ven 30 sep 2010:
	`Looks good in general.`
	`My only comment is about the xsd coding in annex_3_Maint.doc`
	
	`In my experience, element names and type names must be kept in sync as much as possible to prevent confusion.`
	`Also I would consistently use camelCasing for element names and (complex) types.`
	
	`Some examples:`
	`- ris_idxReqMutType/wwsectCode`
	`The element name is wwsectCode, while the type is fwCodeType.`
	`My suggestion would be to rename wwsectCode to fwCode, because this field indicates the fairway, and not a section thereof.`
	
	`- element ris_idxReqMutType/objcode.`
	`to be consistent in the camelcasing, element name objcode, should be objCode.`
	`The type risobjCodeType should be objCodeType.`
	`Al ris prefixes I would leave out, as the parent element name/type already denominates the fact that this is a ris index attribute.`
	
	`- ris_idxReqMutType/hectomt`
	`hectomt could better be hectom or hectometer.`
	`kmCodeType could better be hectomType or hectometerType.`
	
	`- complexType ris_idxReqMutType`
	`All elements are optional, while the RIS Index mandates that some are mandatory.`
		Using the same name, type convention is a bit difficult in this situation.
		
		The XSD elements are based on general basic (simple)types. The XSD was designed in such a way that a simpletype definition can be used in different elements (within different structures).
		
		In that way its easy to change formats of basic element etc.
		
		The basic (simple)types could be seen as core components within this XSD. International std & recommendations promote the usage of such core components.
		
		The idea of the **ris_idxReqMutType** necessary for (manual) mutation request is: that most of the fields are optional and that the European Datamanager will have to (manually) complete these fields first (via GUI) before he accepts the request.
		
		In the ris_idxType most of the fields are mandatory. That structure is used when adding records directly tot the DB (that's why they are mandatory, while when requesting less mandatory fields are applicable because of the manual validation).

12	Linda Scherpbier sep 2010 (implementation remarks): **getDataXML()** p15: Add example in subcode parameter %AMS% matches all codes with letters AMS in it.	All these remarks and corrections will be

PS/PN number:	EU RDMS Platina
WP number:	RDMS-API specs
Log number:	1

No	Observations	Agreed corrective actions(s)
	All functions: **version** parameter is not a string (its defined as a long in the XSD) and it can not be empty. Also % is not allowed. Redefine and correct description: - 0 (zero) is wildcard, will return all versions (also erased ones). - not specified, is no specific version and will return only the latest (actual) one. - any other value >0 will return that specific version (if available). *Also **correct all function** definitions (parameter version = **long**) in the function descriptions in the document.* *Add **examples** for wildcards "%" (pattern) matching:* *NLRTM = exact match* *NLRTM% = everything beginning with NLRTM* *%00000 = everything ending with 00000* *%RTM% = everything with RTM in it (anywhere).* Appendix A4 AND codes description: XSD tag **maxWeightCodeInland** in eri_adnReqMutType and in eri_adncodeType must be renamed (2x) to **maxWeightCodeInland<u>BulkCont</u>** to be inline with the rest of the data. Descriptions in the different languages: Add extra note (footnote 5, p13) that languages that are not supported will be left blank (empty string). **risidxCode** in type ris_idxType is wrong must be simpltype instead of complextype (must be same type as ris_idxReqMutType). **eri_locationType** and **eri_locationReqMutType**, field **ivsCode** is ambiguous (inconsistent), because it matches an internal database field (with another meaning). So its better to rename this field in de XSD to **ivsVTS**. Add **recCnt** in **refDataReturnType** en **refDataReqMutType** structures. This counter indicates the found/available records. Less records may be returned in the XML due limitations. Inconsistencies found after complete rescan of XSD (differences between **refdataReturnType** and **refdataReqMutType**): -refdataReqMutType\reftype must be of type reftypeType. -refdataReqMutType\eri_hscode\name\loc must be of goodNameType -refdataReqMutType\flagsbulk + flagstank must be of type flagsType. Ris_idxType & ris_idxReqMutType <objname> M->O (accordance the available RIS idx data)	implemented in the XSD and the specification document.

QC / Compliance Observation Record (COR)

PS/PN number:	EU RDMS Platina
WP number:	RDMS-API specs
Log number:	1

RIS Data Management Service, API specifications

No	*Observations*	*Agreed corrective actions(s)*
13	Bernd Birklhuber sep 2010: I could not find UNDG, IMDG, NST or NST/R codes in the XSD and the API documentation. And I suppose that these documents should cover all the data in the reference tool. Will these codes be available in the reference tool?	At the moment only the following refdata (out of your list) is supported: AND(R), HS with link to NST 2007. Other data (tables) can be added but necessity has to be investigated in cooperation with other RIS partners. For now => no actions for RDMS impl. (investigate necessity first).
14		

Appendix A.1

REF tool XSD suggestions (summary)

Davor Hrg, CRUP Croatia. 28 sep 2010

1) Avoid xs:extension for iversion

there are 19 usages of <xs:extension
14 of those are extensions to add version attribute:
<xs:attribute name="iversion" type="tns:interfaceVersionNrType" use="required"/>
It is done in a way that displays no benefit from using extension construct.

To explain the problem I'll focus on one of them only (the pattern is the same for all 14)

we have matchByCode element
<xs:element name="matchByCode">
that extends matchByCodeType only to add "iversion" attribute

```
<xs:extension base="tns:matchByCodeType">
   <xs:attribute name="iversion" type="tns:interfaceVersionNrType" use="required"/>
</xs:extension>
```

The matchByCodeType contains few elements reftype,code,version
and is not used anywhere else in the xsd to be extended or otherwise.
matchByCodeType is translated to object:

```
class MatchByCode {
   ReftypeType reftype;
   String code;
   Long version;
}
```

The extension creates an additional class in object mappings which is wastefull and useless.

```
class MatchByCodeType extends MatchByCode{
   String interfaceVersion;
}
```

Removing the extension and moving the attribute to matchByCodeType
would result in singe Object for this.

```
class MatchByCode {
   ReftypeType reftype;
   String code;
   Long version;
   String iversion;
}
```

This is repeated 14 times, so we would have 14 classes less,
or 14 files less complicated code.

This change would not affect the resulting element, just the resulting Object structure where XML is mapped to objects.
This change brings no relevant change to the XML but is intended as improvement for those implementing it.

If there is a reason not to do this, I'd like to know, so I can consider or argue it.

OLD XSD CODE

```
-----------------------------------------------------------------------------------------------
  <xs:element name="matchByCode">
    <xs:annotation>
      <xs:documentation>Function decl: Find information based on a Code (lookup
code)</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="tns:matchByCodeType">
          <xs:attribute name="iversion" type="tns:interfaceVersionNrType" use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
...
...
  <xs:complexType name="matchByCodeType">
    <xs:annotation>
      <xs:documentation>type decl for matchByCode() call</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="reftype" type="tns:reftypeType"/>
      <xs:element name="code" type="xs:string"/>
      <xs:element name="version" type="tns:refrecVersionType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
-----------------------------------------------------------------------------------------------
```

NEW XSD CODE

```
-----------------------------------------------------------------------------------------------
  <xs:element name="matchByCode" type="tns:matchByCodeType">
    <xs:annotation>
      <xs:documentation>Function decl: Find information based on a Code (lookup
code)</xs:documentation>
    </xs:annotation>
  </xs:element>
...
...
  <xs:complexType name="matchByCodeType">
    <xs:annotation>
      <xs:documentation>type decl for matchByCode() call</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="reftype" type="tns:reftypeType"/>
      <xs:element name="code" type="xs:string"/>
      <xs:element name="version" type="tns:refrecVersionType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="iversion" type="tns:interfaceVersionNrType" use="required"/>
```

```
   </xs:complexType>
-----------------------------------------------------------------------------------------------
```

Corrective actions:

See corrective actions as mention for observation #1 + 2

(extension removed, because “iversion” attribute has been deleted)

2) Interface version number is not needed

Although I have sent a modification that involves interface version number that is sent back and forth, there is an issue that might make that change obsolete.

My point is that interface version number is not needed and is redundant useless information.

as WSDL clearly states the url for the service will be : /refws/v1/RefWebService

```
   <service name="v1/RefWebService">
      <port name="RefWebPort" binding="tns:RefWebPortBinding">
         <soap:address location="http://reftool.risexpertgroups.org:80/refws/v1/RefWebService"/>
      </port>
   </service>
```

when new version is created my guess that the url will be different
for example: /refws/v2/RefWebService

```
   <service name="v2/RefWebService">
      <port name="RefWebPort" binding="tns:RefWebPortBinding">
         <soap:address location="http://reftool.risexpertgroups.org:80/refws/v2/RefWebService"/>
      </port>
   </service>
```

I definitely would have different version on a different URL.

With these facts, if two versions of the web-service would be active clients would connect to different url and different implementation would handle each URL. Hence there is no reason for interface version number to be sent back and forth, it is redundant.

I can imagine PHP code that handles both versions by simply adding small logic to cover the difference in the XML structure. But the same php code could be easily be rewritten to use the url instead of the version number inside the XML itself to check the interface version.

Corrective actions:

See corrective actions as mention for observation #2 and #5

(iversion attribute deleted)

3) Remove empty <xs:extension

remove empty <xs:extension and replace with type="" attribute.

By looking at complextype:**ris_idxType** and complextype:**ris_idxReqMutType**
it is clear that both are used and considered same.

complextype:**ris_idxType** uses for **risIdxCode**:

```
<xs:element name="risidxCode">
<xs:annotation>
<xs:documentation>Be sure to ...</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:simpleContent>
<xs:extension base="tns:risCodeType"/>
</xs:simpleContent>
</xs:complexType>
</xs:element>
```

and complextype:**ris_idxReqMutType** uses for **risIdxCode**:

```
<xs:element name="risidxCode" type="tns:risCodeType">
<xs:annotation>
<xs:documentation>Be sure to edit, ...</xs:documentation>
</xs:annotation>
</xs:element>
```

Empty <xs:extension appears only 5 times, and this suggestion is just for cleaning xsd.
This also means another 5 Java classes less, that xsd<->object binding would generate.

Corrective actions:

See corrective actions as mention for observation #9.

4) Duplicate data types

avoid duplicating whole data types just to because one has few mandatory fields more.

The example I'll use is **ris_idxType** and **ris_idxReqMutType**.

Both type describe same data, but are differentce is that
ris_idxReqMutType has minoccurs="0" on following fields
- unlocCC
- unlocLC
- wwsectCode
- objcode
- hectomt
- lat

- lon
- startdate
- infodate
- version
- erased
- lastupdate

I've noticed multiple times desire to move as much validation as possible to the XSD. Like any other extreme measure I think this one needs to be loosened a bit to allow thinking of benefits/problems the extreme validation brings.

In this case to have validation for those properties being mandatory/not results in copy/paste data definition that has to be maintained in parallel, and will create additional binding class when XML<->Obj bindings are generated.

So to digress onto the implementation code:
I would end up with 2 classes with completely same fields just so XSD
checks the mandatory fields instead of me.
Hm, nice I do not have to check it, but hey! now I have to somehow handle these
2 classes as one in my code.
Ok, no problem I will create an Interface that both classes will implement,
so I do not have to implement all code concerning this twice.
Oh wait, not so nice ... I have 3 classes instead of one, and I still myself
have to check the mandatory fields or there will be errors when it is converted to XML.

I am not trying to undermine the effort of validation, but both XSD and code handle it.
Moving it to XSD complicates the code and hardens the maintenance.
I am aware of benefits but am also pointing out complications.

This pattern is repeated 8 times (*ReqMut*):
- ris_idxReqMutType
- eri_locationReqMutType
- eri_hscodeReqMutType
- eri_adncodeReqMutType
- eri_conttypeReqMutType
- eri_packtypeReqMutType
- eri_shiptypeReqMutType
- eri_countryReqMutType

The duplicated code is 538 lines, whole xsd is 2115 lines long removing it would make XSD 25% smaller or 1577 lines long.
I consider this a good thing.

From the implementation point of view 8 classes are needed to handle it in the simplified case, but in the current state there are
- 8 original classes
- 8 duplicates
- 8 interfaces to handle the duplicity

I am definitely for having 8 instead of 24 classes here.

To make my case stronger I will point out that there are already errors/inconsistencies in the current XSD caused by the duplication.

eri_locationType versus **eri_locationReqMutType**
eri_locationType has documentation for elements
 - ivsCode
 - exits
while eri_locationReqMutType does not
(not so important)

eri_hscodeType versus **eri_hscodeReqMutType**
element: name.Loc has different type definition
in eri_hscodeType:
<xs:element name="Loc" type="tns:**hsCodeType**"/>
in eri_hscodeReqMutType:
<xs:element name="Loc" type="tns:**goodNameType**"/>
(this is a problem)
also nstName has documentation in eri_hscodeType
(not so important)

... there are more incosistencies but I feel no need to list them all

XSD validation is good, duplication is bad.
Does duplication bring more problems that this particular validation solves ?
I say yes.

Corrective actions:

See corrective actions as mention for observation #2

It was required to have a full documentstyle definition including a complete XSD validation (especially for mandatory fields etc). This is the reason for these two structures (no action will be taken for this mandatory vs conditional issue).

5) Duplicate definitions for translations

there are two reappearing definitions for translations in the XSD:

A shorter one:
```
<xs:complexType>
<xs:sequence>
<xs:element name="Loc" type="tns:erilocNameType"/>
<xs:element name="NL" type="tns:erilocNameType"/>
<xs:element name="DE" type="tns:erilocNameType"/>
<xs:element name="FR" type="tns:erilocNameType"/>
<xs:element name="EN" type="tns:erilocNameType"/>
</xs:sequence>
</xs:complexType>
```
being 9 lines long

and used by:

- eri_locationType.name
- eri_locationType.termname
- eri_locationReqMutType.name
- eri_locationReqMutType.termname

... so instead of 9 lines we get 36 lines

A longer one:

```
<xs:complexType>
<xs:sequence>
<xs:element name="Loc" type="tns:goodNameType"/>
<xs:element name="NL" type="tns:goodNameType"/>
<xs:element name="DE" type="tns:goodNameType"/>
<xs:element name="FR" type="tns:goodNameType"/>
<xs:element name="EN" type="tns:goodNameType"/>
<xs:element name="BG" type="tns:goodNameType"/>
<xs:element name="CS" type="tns:goodNameType"/>
<xs:element name="DA" type="tns:goodNameType"/>
<xs:element name="EL" type="tns:goodNameType"/>
<xs:element name="ES" type="tns:goodNameType"/>
<xs:element name="ET" type="tns:goodNameType"/>
<xs:element name="FI" type="tns:goodNameType"/>
<xs:element name="HU" type="tns:goodNameType"/>
<xs:element name="IT" type="tns:goodNameType"/>
<xs:element name="LT" type="tns:goodNameType"/>
<xs:element name="LV" type="tns:goodNameType"/>
<xs:element name="PL" type="tns:goodNameType"/>
<xs:element name="PT" type="tns:goodNameType"/>
<xs:element name="RO" type="tns:goodNameType"/>
<xs:element name="SK" type="tns:goodNameType"/>
<xs:element name="SL" type="tns:goodNameType"/>
<xs:element name="SV" type="tns:goodNameType"/>
<xs:element name="HR" type="tns:goodNameType"/>
<xs:element name="RU" type="tns:goodNameType"/>
<xs:element name="SR" type="tns:goodNameType"/>
</xs:sequence>
</xs:complexType>
```

beinng 29 lines long

and used in:

- eri_hscodeType.name
- eri_adncodeType.name
- eri_adncodeType.syn
- eri_conttypeType.name
- eri_packtypeType.name
- eri_shiptypeType.name
- eri_countryType.name
- nts_dataType.name

... and of course in:

- eri_hscodeReqMutType.name
- eri_adncodeReqMutType.name
- eri_adncodeReqMutType.syn
- eri_conttypeReqMutType.name

- eri_packtypeReqMutType.name
- eri_shiptypeReqMutType.name
- eri_countryReqMutType.name

At very least this should be reduced by using top level type.so instead of 29 lines we get 464 lines

to sum it up, the xsd can be reduced from 2115 lines by 462 to 1653 lines (22% reduction).
In the implementation there would be 2 classes instead (20+2)

Another issue here is that translations have different restriction on the length. There may be valid reasons for this due to some legacy issues I am not aware of, but I still believe whatever the reasons when faced with the added complexity should be reconsidered.
Again longest restriction can be used in the XSD and specific restrictions for length can be enforced in the implementation.
I do not believe reducing length of few of the translations will have singificant impact on database size given today's memory and storage capacities.

After cleaning this, my personal preference would be not using a fixed structure for translations but using a list of translations elements having code+translation

```
<xs:sequence>
<xs:element name="code" type="tns:languageCode"/>
<xs:element name="Loc" type="tns:translation"/>
</xs:sequence>
```

where code is either "Loc" or ISO code for a language, this way adding languages would not require change in the XSD and by that in the implementations as well.

Corrective actions:

The datamodel contains so-called flat tables (not a relational model). One for each data type (so additional types could be added easily without affecting the datamodel to much). The XSD was based on this flat structure (no action will be taken for this issue).