Report

XML Encoding of S-52 Conditional Symbology Procedures

UTP1156-01 | 26 April 2009| Status: Draft 1
Submitted to: IIC

North Heath Lane, Horsham, West Sussex, RH12 5UX, United Kingdom
Tel: +44 01403 273 173 Email: info@envitia.com
www.envitia.com

TABLE OF CONTENTS

Authorisation

Approved	Name	Designation	Signature	Date
Author	A. Hughes	SW Engineer		
Project	J. Thomson	Project Manager		
Technical	N. Kirk	Technical		
Commercial	S. Little	Commercial		

Authorised	Name	Designation	Signature	Issue Date
QA	Karen Chrismas	QA Manager		

Accepted	Name	Designation	Signature	Date

Original to be held by QA Department, ENVITIA, Horsham.

Amendment Record

Issue	Pages	Description of Change	Date
01	-	First version	2009-04-23

1. INTRODUCTION

1.1. Purpose

The primary objective of this work is to translate S-52 Conditional Symbology Procedures (CSPs), which are currently compiled as Nassi-Shneiderman-Diagrams, into Extensible Markup Language (XML).

The XML encodings shall define the process flow and resulting S-52 symbology instructions. XML instance documents shall be provided for each of the CSPs defined in Appendix 2, Annex A, Part 1, Section 12 of S-52 [3]. The grammar and structure of these instance documents shall be specified by an accompanying XML Schema.

This document provides an overview of the CSP XML encoding and how it may be used by an application, such as a GIS or ECDIS, that displays ENC data according to the S-52 portrayal rules.

There are many ways that the process logic described in the S-52 CSPs may be encoded as XML. There appears to be no existing industry standards that address this problem directly. The approach therefore defines a bespoke XML grammar tailored to S-52 usage. However, relevant industry standards have been used as *guidance* in creating this XML grammar. In particular, the present implementation has been inspired by OGC Filter [1] for the constructs that are used for encoding expressions, and WS-BPEL [5] has been used as a basis for the process flow constructs.

1.2. Scope

The XML encoding is limited to the S-52 CSPs specified in Section 12 of the S-52 Presentation Library [3] (edition 3.4, January 2008). While the CSP XML encodings may be used to configure the CSP process flow of an application that portrays ENC data, these encodings do not constitute a complete portrayal specification for S-52. Other parts of S-52, such as colour tables, look-up tables and the symbol sets (vector and raster graphics) themselves, would need to be encoded using a separate XML encoding, such as OGC Symbology Encoding [2].

The CSP XML encodings represent a more formal, machine processable, encoding of the Nassi-Shneiderman-Diagrams in S-52. It is the responsibility of a system implementer to map the abstract functionality specified by the XML constructs to appropriate graphical rendering instructions on the target software.

1.3. Follow On Work

This is an overview of the present state of the work. Once initial feedback is provided it should be possible to finalise the XML grammar and provide a comprehensive description. For example, so that an application can map the XML constructs to graphical rendering instructions or internal display lists.

ISO 19117 is presently undergoing significant revision by ISO/TC211. ISO 19117 should be relevant for defining the conceptual model that underpins the concepts in the CSP encodings (portrayal mappings, symbology instructions, etc.). The alignment of the XML grammar with the ISO 19117 conceptual model is a valuable exercise that could be done as part of follow-on consultancy.

Since this work is somewhat novel and has an investigational component, this document also summarises known issues and some alternative XML encodings. Future revisions and the further development of the XML grammar could be dealt with as part of further consultancy, once initial feedback has been received.

1.4. Abbreviations

CSP	Conditional Symbology Procedure
ECDIS	Electronic Chart Display and Information System
ENC	Electronic Navigational Chart
GIS	Geographic Information System
IHO	International Hydrographic Organization
OGC	Open Geospatial Consortium
SE	Symbology Encoding
WS-BPEL	Business Process Execution Language for Web Services
XML	Extensible Markup Language

1.5. References

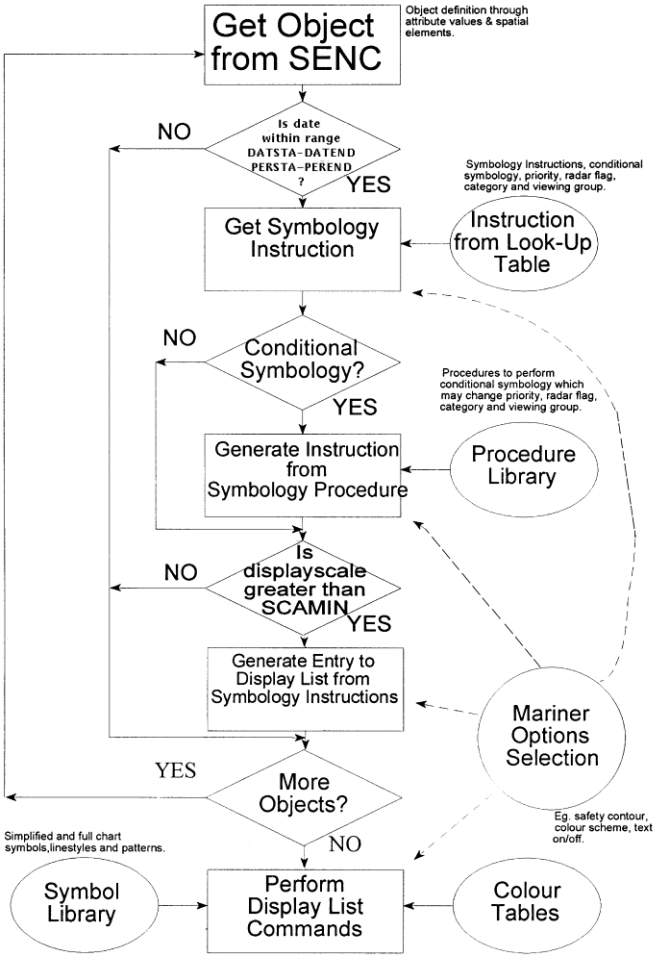
1. OGC 04-095, Filter Encoding Implementation Specification, v.1.1.0, 2007
2. OGC 05-077r4, Symbology Encoding Implementation Specification, v.1.1.0, 2006
3. S-52, Specifications for Chart Content and Display Aspects of ECDIS, IHO, 5th Edition, 1996 (amended 2008)
4. S-57, Transfer Standard for Digital Hydrographic Data, IHO, Edition 3.1, 2000
5. WS-BPEL, Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007
6. ISO/TC211 Draft Revision of ISO 19117, 2006-07-27

2. XML ENCODING OF S-52 CONDITIONAL SYMBOLOGY PROCEDURES

2.1. Overview

The rendering rules for certain S-57 objects are such that they cannot be determined by means of a simple lookup table. Instead, these objects require so-called Conditional Symbology Procedures (CSPs) to define their rendering. The CSPs in turn can rely on a library of external functions.

The process is shown in the diagram below:



The expression of the CSPs as XML documents requires the construction of a schema to define the rules of the grammar used. There do not appear to be any ideal pre-existing schemas however, so a new set of schemas has been generated.

The schema must allow expression of two main areas: the flow control grammar, and the grammar for expressions. Each of these components has been based on existing schemas:

WS-BPEL

WS-BPEL is concerned with the orchestration of web services, and thus defines various flow control structures. It was therefore used as the basis of the flow control grammar that we have developed.

However, as the name suggests, WS-BPEL is designed specifically for interacting with web services. So, whilst it features an invocation call for invoking a function, it is assumed that that function is resident on a remote server, accessible through a particular web address.

Furthermore, web services typically expect all their parameters to be supplied as a single block of data – the archetypal example being:

www.server.com/service.htm?variable1=foo&variable2=bar

The procedure calls made by the CSP routines are all local, and frequently make use of multiple parameters. To force multiple parameters through the one-parameter restriction imposed by the BPEL schema would involve the artificial step of concatenating all the parameters into a single string before invoking the function.

For this reason, we have used BPEL only as a starting point. We have removed all the mechanisms for remote invocation or querying of web services. We have however kept the basic flow control constructs, such as conditions, loops and assignments.

We have instead added a mechanism for calling a local function, that allows the passing of multiple parameters, and receiving information back from the called function through those same parameters where appropriate.

OGC Filter

OGC Filter is a standard for encoding expressions with appreciable support within the GIS industry. It does not however contain process flow constructs, and there are some issues with the mechanisms it uses for function call invocation.

In particular, it appears that OGC filter does not offer a means for naming the parameters expected by a function. Example 7 in the specification is as follows:

```
<Filter>
        <PropertyIsEqualTo>
                <Function name="SIN">
                        <PropertyName>DISPERSION_ANGLE</PropertyName>
                </Function>
                <Literal>1</Literal>
        </PropertyIsEqualTo>
</Filter>
```

From this we can see that there is no means for using the schema to cross-check that a function called "Sin" does in fact exist, nor to obtain the function's signature: in this case, a single value being used as input only. This is appropriate for general usage, but in the case of S-52, where the list of functions is a closed well-defined set, the schema can specify the function list.

We wished to be able to check that calls were only being made to known functions, and that we were supplying the appropriate parameters. Also, in some cases, functions are stated to return multiple different values, meaning that, in some cases, parameters must be able to return a value from a function as well as/instead of supplying it to the function.

For these reasons, we developed our own "call" syntax, along with a library detailing the function signatures of all functions called by the S-52 CSP routines.

The overall structure of an XML instance document that specifies a CSP is shown below.

2.2. Structure of the XML Instance Document

At the highest level, this document can be divided into 5 parts, in the following order:

1) processName – would be used to assign a function name for the CSP if the XML were to be transformed to executable code.
2) processDescription – optional text describing CSP.
3) parameters – block that describes all inputs to the CSP, and outputs from it.

4) variables – block within which to declare variables local to the CSP. Not strictly necessary in the context of the XML, but would likely be used if the XML were to be transformed to executable code.
5) sequence – block that contains the sequence of operations that comprise the CSP itself.

The sequence block in turn permits a number of sub-elements, in arbitrary order:

1) assignment – for assigning the result of an expression to a local variable.
2) if – conditional test. See below.
3) while – loop. See below.
4) call – function call. The S-52 CSPs make numerous references to functions such as DrawLine(), getFirstVertex(), etc, as well as the other CSP routines.

2.3. Process Flow Constructs

The "if" test comprises a condition block, followed by an actionSequence block should the condition evaluate to true. It also allows multiple optional elseif tests and a final else block. The actionSequence, elseif and else blocks can each contain an arbitrary mix of the 4 sequence sub-elements described above.

The "while" loop consists of a condition block followed by an actionSequence should the condition evaluate to true. The actionSequence can contain an arbitrary mix of the 4 sequence sub-elements described above (allowing, for instance, multiple nested conditions).

Expressions are found in the source of assignments, as arguments to function calls, and within the condition blocks of "if" and "while" statements. Expressions are composed of an arbitrary mix of variable references, literal values, S-52 enumerated values, operators, and function calls.

Operators are based on OGC Filter syntax, defining function calls in Reverse Polish Notation. We have attempted to keep the set of operators to only the minimum set directly required by the S-52 CSPs.

2.4. Procedures and Functions

The CSPs make use of a substantial number of external functions, to perform operations such as getting the first vertex of a line object, or the bearing from one point to another.

Sample prototypes for these functions have been defined in the S52_functions.xsd schema document. The prototypes name the expected parameters to each of the functions. Note however that there is no typing information defined for the parameters at this stage, ie: parameters are not defined as being of type "integer" or "string".

In fact, much of the typing used in the current version of the schemas and XML documents has been deliberately left very loose. For instance, the generic "value" type is used rather than distinguishing between integers, floats, etc. It is anticipated that the types can be more precisely defined as part of future work.

This is because the first step in specifying a precise set of types for the schemas and XML documents would be to establish what set of types to use, and how they interrelate.

The most likely candidate for a type model would appear to be ISO 19117. However, at the current point, this specification is still in draft form, and so it remains to be seen whether it will ultimately form a suitable model for the description of S-52/S-57 types.

2.5. Symbology Instructions

The CSPs also make use of a number of rendering commands – such as drawLine, drawRing, etc. It is assumed that the implementation of these functions will be system-specific. Also, their action may be deferred – figure 1 above describes a "display list" indicating that the actions of these rendering commands are deferred until all objects are ready to render.

As with the other functions referred to by the CSPs, basic function prototypes for the rendering functions have been placed in S52_functions.xsd, where each prototype describes the names of the expected parameters.

The rendering functions are passed S-57 objects, and it is assumed that they are able to automatically extract the spatial geometry components described below.

2.6. Handling Objects

We cannot assume that the S-57 objects are part of an object-oriented hierarchy. However, we need to assume some commonality between the different objects, as these assumptions are made by the CSP routines themselves.

We assume that we are able to pass an object of arbitrary type to a CSP, and that we can then classify the object's type at runtime: DEPARE, DRGARE, etc.

We assume that every object is comprised of geometry information and a set of attributes.

The geometry information can be identified at runtime as being of POINT, LINE, or AREA types.

We assume that we can query an object for a particular attribute by supplying the name of the attribute. If the object does not have an attribute with this name, or if its value is undefined, will result in the same thing – a return item that has the value "undefined".

We provide functions hasValue() and hasNoValue() that are assumed able to identify attributes or variables that are either uninitialized, unset, or absent – we assume these 3 states are considered identical.

We also provide a function clear() that will set an attribute or variable to the undefined state.

2.7. Recommendations for addition of objects/attributes to S-57

The statement of work contained a section "*Recommendations for adding any objects or attributes to those specified in the IHO S-57, IHO Transfer Standard for Digital Hydrographic Data, Appendix A, that would preclude the need for any of the CSPs listed above shall be made prior to the corresponding CSPs being translated.*"

At present we do not have any such recommendations. Instead, we have attempted to faithfully model the CSPs as given.

3. SUMMARY

The work carried out so far has focussed on attempting to eliminate (or at least identify) ambiguities in the diagrammatic CSPs. As such, this work had as its target the production of a series of XML documents that express the CSP logic in a rigorous and unambiguous manner.

Whilst the schemas and XML documents produced should meet the above goal, they do not represent the ideal end product for this process.

In particular, it is anticipated that further work could more tightly integrate the CSP schema with existing standards such as OGC Filter and ISO 19117.

However, this will require additional input from the customer – in particular, when there are discrepancies between the S-57 model, and ISO 19117. Such discrepancies will necessarily involve further design decisions.

3.1. Known Issues

General – applies to all CSPs

Linestyle function LS(...) treated as special case string

Symbol function SY(...) treated as special case string

The CSPs frequently reference the LS() and SY() functions, to set particular line styles or symbols. Whilst it appears that these are S-52 functions

Attributes referred to by name (instead of enum list of all S-57 attributes)

How to set variables to unknown - have "Clear" function.

Is "unknown" = "undefined" = "value not given"?

Area pattern IDs treated as strings

Action of pre-selected styles. Do they affect only next drawing command, or all subsequent commands until de-selected?

The "Is the attribute X given?" test is typically replaced with 2 steps:

1) Copy attribute to local variable

2) Test if local variable has a value

Need to check loop constructs carefully

Very loose typing. Using a generic "Value" type for all sorts of numbers.

Should probably ultimately be replaced with ISO 19117 hierarchy, but not clear how well that will fit with S-57/S-52 model.

IsolateDigit function - need to pass in number from which to pull digit.

Continuation blocks in original document. Could be replicated with "goto" construct in code, but presume that's not wanted.

Could just pull Continuation blocks into appropriate places (they are never referenced from multiple points), but that'll break link with graphical diagrams, making XML CSPs harder to verify, so leaving this re-ordering as late as possible.

We treat parameters and variables identically.

showSelectedSymbol can be replaced by showSymbol

Specific to Individual CSPs

DATCVR02

The comments at the start of this CSP state;

"Because the methods adopted by an ECDIS to meet the IMO and IHO requirements listed on the next page will depend on the manufacturer's software, and cannot be described in terms of a flow chart in the same way as other conditional procedures, this procedure is in the form of written notes."

For the same reasons, it is unclear if this CSP can be represented as XML.

DEPARE02

There is a section that contains the test:

"Is the spatial component shared by a 'DEPCNT' (Depth Contour) object?", folowed by

"Is the value of VALDCO given?"

Presume that VALDCO is attribute of shared-component DEPCNT object(s)?

What happens if spatial component is shared with multiple DEPCNT objects, some of which have the VALDCO attribute given, some not?

Unclear whether various attributes are being pulled from calling object, or objects that it shares a spatial component with.

DEPVAL02

Using clear() to set values to unknown. We're assuming that "unknown" = "undefined" = "not provided", and that the clear() function sets a parameter, variable or attribute so that it fulfils the above tests.

LEGLIN03

Unclear where DISTANCE_TO_RUN value sourced from

LIGHTS05

How to get "no sector" lights?

Is it possible to move the functionality of the fn:textGroupSelected function into the getMarinerEnteredValue function?

Needs work on the section that contains "object.SECTR1" references. Should be replaced with calls to extract SECTRx attribute from objects, copy into local variables, then use those.

Need select linestyle function that uses currently selected colour. At the moment we use the same LS() function as specified in the original CSPs to set the line style. That function however always requires 3 parameters – solidity, line width, colour. Not sure how to omit colour parameter except by using whole new function.

LITDSN

Not implemented. Currently exists as C source code, which performs various text manipulations. Unclear if this can feasibly be expressed in XML without defining multiple new functions – essentially just mirroring the various C function calls presently used.

OWNSHP02

Need function to get scaled ship's length

Need functions to scale outline of ship

Need functions to account for conning position

PASTRK01

Need function for repeated symbols along line

QUAPNT02

Need break loop construct

SAFCON01

We need to use the createSymbolName() function because of differing use of different-typed data. First 2 values treated as numbers and added together: 60 + 5 = 65.

Then result treated as string and appended to another string: SAFCON + 65 = SAFCON65.

The createSymbolName() function can be assumed to perform both operations in one step.

SLCONS03

Unclear how lookup tables presented in the original CSP diagram are meant to work.

SNDFRM03

Note use of concatenate operator, unlike SAFCON01, which uses createSymbolName(). This is because we're simply concatenating 2 strings this time.

SEABED01

Not sure if final instruction to "show it on top of the area's colour fill" needs an explicit instruction. When you select an area pattern, is it automatically drawn when the area is?

TOPMAR01

Needs some way of dealing with partial class identifiers. Maybe special function?

Note that we have assumed that the class of an object can be identified at run time. Identifying the class given only part of the class name may be more difficult however, depending on the exact implementation.

UDWHAZ04

DRVAL2 does not appear to be used, despite comments before original CSP

Needs new function to get next area object.

VESSEL02

Needs new drawing functions for scaled lines, and repeating symbols along lines.

Unclear conditions: eg: If vestat1 (activated) … else if vestat3 (dselected) …

Are vestat1 and vestat3 attributes. Can we assume that if vestat1 is set of true, then vestat3 is not?

VRMEBL02

Need to be able to get linestyle provided by manufacturer.

WRECKS04

Continuation blocks left in same position as on original CSP diagrams. We haven't provided a "goto" command. Since the continuation blocks are only ever referenced from a single point, they should be cut and pasted into that point. This will however affect the ordering of the instructions compared to the original diagrams, making verification of the XML logic harder. For this reason, the continuation blocks have been left in the order found in the original diagrams.