Submitted by: Hugh Astle – CARIS April 21, 2012
Reference to: review of newly proposed Portrayal specification at IHO joint TSMAD24/DIPWG4 meeting in Monaco 7-11 May 2012.

Contents

Intro

Recently (April 5) a new portrayal proposal was distributed for review.
There is a lot of good stuff in it and obviously a lot of effort was made to put it all together.

The proposed portrayal document can be used as a reference or example of desired functionality but we should give careful consideration as to whether the same functionality could be achieved by using existing standards and toolkits as opposed to defining things from scratch or in our own way. The document provides a good inventory of presentation capabilities, symbology, linestyles, patterns that are needed to implement a portrayal engine. We can compare it to other specs such as OGC SE, SVG and look for consistency or overlaps and adjust as necessary.

This document is meant as information and to stimulate discussion while we are reviewing the portrayal proposal.

Indigestion

One part in the existing portrayal proposal that is really hard to digest is the invention of a new programming language and the obfuscated encoding of that language into XML.

Do we really want every ECDIS manufacturer to be building their own interpreters, compilers and processing rules to generate a safe ECDIS display? The number of permutations and combinations would be nearly impossible to test and the safety of life at sea is at risk. If we do indeed need a programming language, why not use one that already exists, where open source code and standards are available with tools to test and debug the functionality.

Review

Goals for Portrayal system

Let's take a step back and consider some of our expectations or goals:
1. We want all the ECDIS to display navigational chart data in a consistent and safe way so that Mariners can understand and use any official ECDIS display for appropriate decision making.
2. We want to be able to deliver new data based on updated catalogues with new objects or attributes and to have the ECDIS display that data correctly (no question marks or missing objects).
3. There seems to be some consensus on allowing ECDIS providers to deliver SENCs directly to ECDIS but that the ECDIS system must also have the ability to ingest an officially encoded 000 file and updates in the event that SENC data is not available.
4. There may be some other hopes of having an ECDIS recognize and be able to open and display a new S-100 based product as an "overlay" with an ENC providing the product is defined from a recognized version of S-100.

S-52 portrayal capabilities

S-52, designed over 20 years ago, had a basic best match lookup table, symbols, linestyles, area patterns, color tables that could all be delivered and updated as a binary file. It required an implementer to provide some "built-in" conditional procedures that would be called for portrayal decision making too complex to be handled by the simple best match lookup table. What was missing or needs improvement:

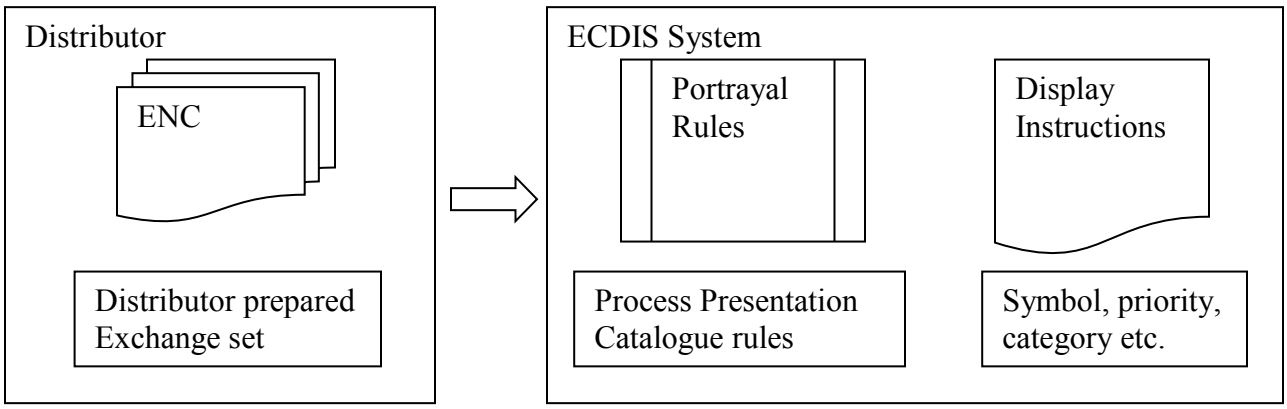
1. Lookup table only supported attribute value equals operator, limited logic
2. No recognition of new complex, hierarchical attribution and multiplicity of attributes.
3. Built in conditional procedures hard coded to one catalogue and software updates required to alter the logic.
4. Conditional procedures needed to handle portrayal related to spatial attribution.
5. Admin for managing versioning, dependencies with catalogues product specs and such was weak or non-existent.
6. Implementers were allowed to hard code as much as they wanted to providing they got the same results. Some hard coded the lookup tables and not just the conditional procedures. Different symbols used as well.
7. Some presentation capabilities, transparency, pattern support etc. could be improved.
8. Doesn't account for interrelated products and stacking of data coming from different products.
9. Colour palettes and colour definition rules need to be revisited based on environmental and display technology differences.

Reality Check

The S-52 portrayal was intended to be flexible such that rules (at least the simple best match), symbols and colours could be changed. One of the main reasons for this was that there was little practical experience with ECDIS and it was expected that with feedback from real usage the presentation would evolve. In the past 20 years there have been few changes in the display (maybe partially because it was frozen) and now we have 20+ years of experience to work with. Do we expect the portrayal of ENCs to be changing greatly over the next 20 years?

Portrayal Process and options to consider

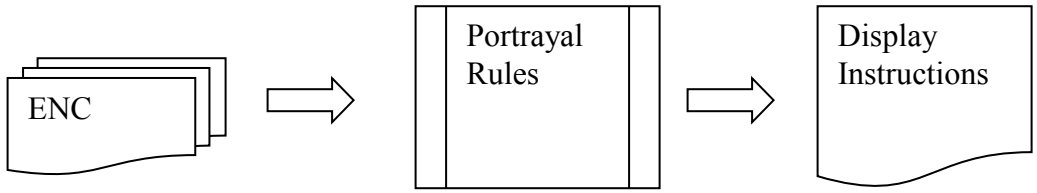
The portrayal process works on a dataset (ENC) and generates Display Instructions for each object.



We have boxed ourselves into thinking that the Portrayal rules need to be processed in the ECDIS. In the end, what the ECDIS operates on is typically the SENC which could be created by a distributor and delivered to the ECDIS. We want to maintain that an ECDIS can open an ENC directly in the case where a distributed SENC is not available. There are other options that could be considered such as delivering the Display Instructions along with an ENC as part of the ENC or as a simple XML format. If an ECDIS could read an ENC and apply the given Display Instructions then it could display data that it has no direct knowledge of.

What if…

Consider breaking the process into steps and whether the boundaries can be moved.



Consider the options of moving the boundaries of where things happen.

Symbol references within or delivered with ENC

What if the default portrayal instructions were generated with the ENC tested in the production shop and delivered with or as part of the ENC? Each object would be given a style code/identifier as a reference to a record in a library of styles. Each style record in the library would contain properties for symbol references, category, and priority etc. The library of styles would be delivered to the ECDIS as a separately maintained, updateable entity.

Pros:

1. As long as the ECDIS can process the style records it should be able to provide a default display of anything given to it.
2. For minimum mandatory carriage display the risk is minimal and results highly consistent.

3. We are already planning on including text placement information in the ENCs, only a small step to include style info.
4. Very little additional space needed.
5. Easy to change a symbol or priority on a specific style by updating the style library.
6. Minimum need for conditional processing on ECDIS. Would need a mechanism to test depth related features against mariner context parameters (e.g. Safety depth).
7. Style codes could be applied automatically by production systems and adjusted by cartographers as needed.
8. Display systems can still use other presentation rules to display data for purposes other than navigation.

Cons:

1. ENC exchange set slightly larger to carry style codes for each object.
2. Still need some conditional logic in ECDIS against mariner settings.
3. If the logic for what style to use on a given feature class needs to change all affected ENCs need to be updated.

Shared portrayal engine implementation

What if a generic portrayal engine (perhaps an open source implementation) could process rules on board the ship? The engine would be a black box that operates on an ENC and generates display instructions for each object.

Pros:

1. Improved consistency from ECDIS to ECDIS.
2. Allows for updating of portrayal logic without having to update ENC
3. Implementation of black box tested in common

Cons:

1. Interfacing or overhead of data in/out of black box could be undesirable
2. Would not work well if SENCs are delivered to ECDIS
3. Doesn't allow ECDIS manufacturers to gain unique efficiencies or optimizations.
4. Would require update/patch mechanism

Shared portrayal library module

What if the portrayal rules processor was implemented using a common or open source code base but with an API that allows it to be integrated into the ECDIS code? Like a windows DLL, Python Interpreter implementation or XSLT engine (libxslt). Implementation would need to be well defined and perhaps a profile or subset of functionality identified for portrayal use.

Pros:

1. Rules processing definable and testable in production shop or on the ship.
2. Common testing and debugging tools available.
3. Improved consistency

Cons:

1. Might be issues with interfacing and optimization
2. Still need to tackle exchange and update of rules.

Use or extend an existing portrayal model

Hybrid of old and new

Consider using simple conditional matches for as much as possible using a newer implementation of the old lookup table for most of the portrayal assignment. Then only use a programming language approach for the complex, conditional procedures.

It would be good to make some effort to see how far we could go with OGC SE and FE specs as a way to implement the basic matching of features to portrayal styles.

We should spend some time reviewing the existing S-52 conditional procedures and determine what functionality still needs to be implemented with complex rules once we have changed the data model to carry display criteria as related attributes and information rather than having to search through the dataset.

XSLT comparison

Following is an exercise comparing the custom designed rule language from the portrayal proposal with XSLT.

XSLT is a W3C Recommendation (since 1999).
http://www.w3.org/TR/xslt
It was designed as a way to transform XML from one schema to another or to generate HTML. XSLT includes: expressions, looping conditions, switch/choose statements, variables, math functions, string functions. XSLT is mature and included in various applications, web browser, word processors. It is available in open source libraries with implementations in C/C++ and in Java. There are off the shelf applications available for development, validation, step through debugging and so on.

If need be it could be profiled to define a subset applicable to portrayal and since it is written in XML it would be possible to define validation rules to validate against an IHO profile.

It use XPath (OGC Filter Expressions also uses XPath) as a way to navigate through hierarchical elements and attributes. XPath depends on navigation via a node tree. There are example implementations where one could integrate XPath and XSLT on top of a private data model which is exposed as a node tree. A concept also exists to write XSLT rules such that the processor is always going forward through the document nodes and so in theory one feature at a time could be processed.

The basic concept of XSLT is that a template or rule is matched to a node in the document and the rule is applied to generate the output.
For a quick overview of XSLT take a run through a tutorial such as
http://www.w3schools.com/xsl/default.asp
or do a web search for XSLT quickref.

Why look at XSLT and not Python or Lua or some other scripting language? One reason is that we are already using XML as a data format. We are already including Meta data and various ECDIS support files as XML. There will most likely be other S-100 products that use XML or GML as the exchange format. With the inclusion of multiplicity of attributes and complex attributes in S-100 we can end up with hierarchical data which is easy to match with XPath. XSLT works on XML and most likely the ECDIS systems will sooner or later incorporate XSLT processing engines anyway. Also the concept of matching data to templates and applying a style is what XSLT was designed to do and the pattern of matching and converting data for display has been in use in web browsers for some time now.

Following are some A-B comparisons against the proposed rule language.

Feature Rule for cardinal beacons copied from Proposed S-100 portrayal document, coded in a custom language. The example translation takes 556 lines of complex XML.

```
/* FEATURE RULE FOR CARDINAL BEACONS */
RULE BCNCAR01( IN FEATURE_TYPE feature ) : BeaconCardinal
{
  VAR SYMBOL symbol;
  VAR POINT_INSTRUCTION instruction;
  instruction.setFeature(feature);
  instruction.setViewingGroup("17020");
  instruction.setDisplayPlane(("O");
  instruction.setDisplayProiority(80);

  VAR INTEGER catcam := feature.attributes().integerValue("CATCAM", 0, 0);
  VAR STRING  objnam := feature.attributes().textValue("OBJNAM", 0, "");

  VAR INTEGER num := feature.geometryCount();
  VAR INTEGER i := 0;

  WHILE ( i < num )
  {
    VAR GM_OBJECT gobj := feature.geometry(i);
    IF (gobj.isPoint())
    {
      instruction.setGeometry(gobj);
      VAR STRING symbolName;
      SWITCH ( catcam )
      {
         CASE (1)
            symbolName := "BCNCAR01";
         CASE (2)
            symbolName := "BCNCAR02";
         CASE (3)
            symbolName := "BCNCAR03";
         CASE (4)
            symbolName := "BCNCAR04";
         DEFAULT
            symbolName := "BCNCAR13";
      } /* SWITCH */
      symbol := catalog.getSymbol(symbolName);
      instruction.setSymbol(symbol);
      display.addPointInstruction(instruction);

      IF ( length(objnam) > 0 )
      {
         VAR TEXT_INSTRUCTION textInstruction;
         textInstruction.setFeature(feature);
         textInstruction.setGeometry(gobj);
         textInstruction.setViewingGroup("17020");
         textInstruction.setDisplayPlane("O");
         textInstruction.setDisplayProiority(80);

         CALL TEXT01(textInstruction, concatenate("bn ", objnam));
      } /* IF ( length(objnam) > 0 )*/
    } /* IF (gobj.isPoint()) */
    i := i + 1;
  } /* WHILE */
} /* BCNCAR01 */
```

The same logic to match BCNCAR implemented using XSLT, already in XML.

```
<!-- Template to process a cardinal beacon BCNCAR -->
<xsl:template match="BCNCAR">
  <xsl:element name="pointInstruction"> <!—Output a pointInstruction element structure -->
    <xsl:element name="feature"><xsl:value-of select="@fid"/></xsl:element>
    <xsl:element name="viewingGroup">"17020"</xsl:element>
    <xsl:element name="displayPlane">"O"</xsl:element>
    <xsl:element name="displayPriority">"80"</xsl:element>

    <xsl:choose>
       <xsl:when test="CATCAM = '1'">
          <xsl:element name="symbolName">"BCNCAR01"</xsl:element>
       </xsl:when>
       <xsl:when test="CATCAM = '2'">
          <xsl:element name="symbolName">"BCNCAR02"</xsl:element>
       </xsl:when>
       <xsl:when test="CATCAM = '3'">
          <xsl:element name="symbolName">"BCNCAR03"</xsl:element>
       </xsl:when>
       <xsl:when test="CATCAM = '4'">
          <xsl:element name="symbolName">"BCNCAR04"</xsl:element>
       </xsl:when>
       <xsl:otherwise>
          <!-- spit out default symbol if no when clause is found  -->
          <xsl:element name="symbolName">"BCNCAR13"</xsl:element>
       </xsl:otherwise>
    </xsl:choose>

  </xsl:element>

  <xsl:if test="OBJNAM != " " ><!— if OBJNAM is not empty  -->

       <xsl:element name="textInstruction"> ><!— output textInstruction  -->
          <xsl:element name="feature"><xsl:value-of select="@fid"/></xsl:element>
          <xsl:element name="viewingGroup">"17020"</xsl:element>
          <xsl:element name="displayPlane">"O"</xsl:element>
          <xsl:element name="displayPriority">"80"</xsl:element>

          <!-- Use a template to fill in the std text info-->
          <xsl:call-template name="TEXT01">
          <xsl:with-param name="str" select="OBJNAM"/>
          </xsl:call-template>

       </xsl:element>
  </xsl:if>
</xsl:template>
```

Note the example calls a function called “TEXT01” and the XSLT has a similar capability to call another template with parameters. The custom language implementation and the XSLT equivalent follow:

```
RULE TEXT01( IN TEXT_INSTRUCTION instruction, IN STRING str)
{
  VAR COLOR textColor;
  textColor.setToken("CHBLK");

  VAR TEXT_ELEMENT textElement;
  textElement.setText(str);
  textElement.setForeground(textColor);
  textElement.setBodySize(3.5);

  VAR VECTOR offset;
  offset.setX(4.0);
  offset.SetY(-4.0);

  VAR TEXT text;
  text.setOffset( offset );
  text.setHorizontalAlignment( 1 ); /* LEFT ALIGNMENT */
  text.setVerticalAlignment( 2 );   /* BOTTOM ALIGNMENT */
  text.appendTextElement( textElement );

  instruction.setText(text);

  display.addTextInstruction( instruction );

} /* TEXT01 */
```

```
<!—XSLT Template to generate repetitive text parameters -->
<xsl:template name="TEXT01">
<xsl:param name="str"/> <!-- string to use in text output -->

   <xsl:if test="$str != '' ">
         <!-- build the text element -->
         <xsl:element name="text">
               <xsl:element name="offset">
              <xsl:element name="X">4.0</xsl:element>
              <xsl:element name="Y">-4.0</xsl:element>
          </xsl:element>

          <xsl:element name="horizontalAlignment">1</xsl:element> <!-- LEFT ALIGNMENT  -->
          <xsl:element name="verticalAlignment">2</xsl:element>   <!-- BOTTOM ALIGNMENT  -->

          <xsl:element name="textElement">
              <xsl:element name="string">bn <xsl:value-of select="$str"/></xsl:element>
              <xsl:element name="foregroundToken">CHBLK</xsl:element>
              <xsl:element name="bodySize">3.5</xsl:element>
          </xsl:element>

        </xsl:element>

   </xsl:if>

</xsl:template>
```

XSLT processing can be done on XML documents, which includes GML, or on any data that can be exposed to the API of the processor implementation as a node tree.

For the sample above, the XML input and output looks like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<featureCollection>
 <featureMember>
  <BCNCAR fid="1C_0000002115_00001">
   <BCNSHP>1</BCNSHP>
   <CATCAM>2</CATCAM>
   <COLOUR>7</COLOUR>
  </BCNCAR>
 </featureMember>
 <featureMember>
  <BCNCAR fid="1C_0000002118_00001">
   <BCNSHP>6</BCNSHP>
   <COLOUR>2</COLOUR>
   <OBJNAM>Fredericton</OBJNAM>
  </BCNCAR>
 </featureMember>

</featureCollection>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Instructions>
  <pointInstruction>
      <feature>1C_0000002115_00001</feature>
      <viewingGroup>"17020"</viewingGroup>
      <displayPlane>"O"</displayPlane>
      <displayPriority>"80"</displayPriority>
      <symbolName>"BCNCAR02"</symbolName>
  </pointInstruction>
  <pointInstruction>
      <feature>1C_0000002118_00001</feature>
      <viewingGroup>"17020"</viewingGroup>
      <displayPlane>"O"</displayPlane>
      <displayPriority>"80"</displayPriority>
      <symbolName>"BCNCAR13"</symbolName>
  </pointInstruction>
  <textInstruction>
      <feature>1C_0000002118_00001</feature>
      <viewingGroup>"17020"</viewingGroup>
      <displayPlane>"O"</displayPlane>
      <displayPriority>"80"</displayPriority>
      <text>
        <offset>
                  <X>4.0</X>
                  <Y>-4.0</Y>
        </offset>
        <horizontalAlignment>1</horizontalAlignment>
        <verticalAlignment>2</verticalAlignment>
        <textElement>
            <string>bn Fredericton</string>
            <foregroundToken>CHBLK</foregroundToken>
            <bodySize>3.5</bodySize>
        </textElement>
      </text>
  </textInstruction>
</Instructions>
```

Some other misc. functionality supported in XSLT

Binary operators: <=, <, >=, >, =, !=, and, or

Math Operators: +, -, *, div, mod and more
http://www.xml.com/pub/a/2001/05/07/xsltmath.html

For things operations like min, max, abs, sqrt, sin, cos…
http://www.exslt.org/math/

Number functions like: convert string to number with formatting, sum, floor, ceiling, round.

Functions are available to generating numbers and letters to label sequences.

A selection of nodes or elements can be sorted with available qualifiers before being processed.

String operations: All string operations mentioned in proposed portrayal doc are available in XSLT and more.

If you want to know how to do something in XSLT just do a web search there are countless forms on the web with people discussing tips and tricks. You can also find many books about XSLT and related technologies.
If you want to know how to implement something with the proposed custom portrayal language, contact the author of that language.