Using IccXML

To create and get information about profiles

ICC DevCon 2020 - The Future of Color Management

Max Derhak(PhD)
Principal Scientist
Onyx Graphics, Inc.

Representing profiles with RefIccMAX

- RefIccMAX supports two ways of representing iccMAX profiles
  - Binary
    - Implemented by IccProfLib library
    - Defined by iccMAX specification
    - Compact, embeddable format
  - XML
    - Implemented by IccLibXML
    - Implements derived classes from classes in IccProflib
    - Currently defined by implementation
    - Human readable / editable
- iccFromXML and iccToXML utilities allow for conversions between representations

Basic XML profile structure

```
<?xml version="1.0" encoding="UTF-8"?>
<IccProfile>
  <Header>
        <!-- Header fields -->
  </Header>
  <Tags>
        <!-- Tag definitions -->
  </Tags>
</IccProfile
```

iccMAX XML Header Fields Part 1

```
<PreferredCMMType>sig</PreferredCMMType>
<ProfileVersion>5.0</ProfileVersion>
<ProfileDeviceClass>sig</ProfileDeviceClass>
<ProfileDeviceSubClass>sig</ProfileDeviceSubClass>
<DataColourSpace>sig</DataColourSpace>
<PCS>sig</PCS>
<CreationDateTime>now</CreationDateTime>
<PrimaryPlatform>sig</PrimaryPlatform>
<ProfileFlags EmbeddedInFile="true" UseWithEmbeddedDataOnly="false"/>
<DeviceAttributes ReflectiveOrTransparency="reflective"
  GlossyOrMatte="glossy"  MediaPolarity="positive"  MediaColour="colour"/>
<RenderingIntent>value</RenderingIntent>
```

iccMAX XML Header Fields Part 2

```
<PCSIlluminant>
    <XYZNumber X="value" Y="value“ Z="value"/>
</PCSIlluminant>
<ProfileCreator>sig</ProfileCreator>
<ProfileID>1</ProfileID>
<SpectralPCS>sig</SpectralPCS>
<SpectralRange>
    <Wavelengths start="value" end="value" steps="value"/>
</SpectralRange>
<BiSpectralRange>
    <Wavelengths start="value" end="value" steps="value "/>
</BiSpectralRange>
<MCS>sig</MCS>
```

Basic XML tag structure

```
<tagName> <tagTypeName>
    <!-- data entries appropriate for Tag type -->
</tagTypeName> </tagName>
```

Where:

- The values of *tagName* match the sub-section titles for tags defined in section 9.2 of the iccMAX specification
- The values of *tagTypeName* match the sub-section titles for tag types in section 10.2 of the iccMAX specification

Sharing tag data between multiple tags

<*tagName2* SameAs="*tagName*"/>

- Used to link two tags to the same tag data in profile
- Note: The definition for *tagName* must be found before *tagName2* occurs in XML file

Private tag XML tag structure

<PrivateTag TagSignature=“sig” > <*tagTypeName*>

<!-- data entries appropriate for Tag type -->

</*tagTypeName*> </PrivateTag>

Where:

- The value of *sig* is the signature of the private tag
- The values of *tagTypeName* match the section names for tag types in section 10 of the iccMAX Specification

Example Tag Types

Encoding text tag types

```
<multiLocalizedUnicodeType>
   <TagSignature>sig</TagSignature>
   <LocalizedText LanguageCountry="enUS“
     ><![CDATA[text goes here]]></LocalizedText>
</multiLocalizedUnicodeType>

<textDescriptionType>
          <TextData<![CDATA[text goes here]]></TextData>    </textDescriptionType>

<utf8Type>
          <TextData<![CDATA[text goes here]]></TextData>
</utf8Type>
```

XML XYZType tag structure

```
<XYZType>
   <TagSignature>sig</TagSignature>
   <XYZNumber X="val" Y="val" Z="val"/>
</XYZType>
```

XML floating point array tag structure

```
<floatNumberType>
    <TagSignature>sig</TagSignature>
    <Data>numeric data values go here…</Data>
    ..or..
    <Data Filename="file" Format="text/binary"/>
</floatNumberType>
```

Where *floatNumberType* can be float16NumberType, float32NumberType, or float64NumberType

XML number array tag structure

```
<numberType>
   <TagSignature>sig</TagSignature>
   <Array>numeric data values go here…</Array>
</numberType>
```

Where *numberType* can be s15Fixed16NumberType, u16Fixed16NumberType, uInt16NumberType, uInt32NumberType, uInt64NumberType, or uInt8NumberType

XML spectralViewingConditions tag structure

```
<spectralViewingConditionsType>
   <TagSignature>sig</TagSignature>
   <StdObserver>Custom</StdObserver>
   <IlluminantXYZ X="val" Y="val" Z="val"/>
   <ObserverFuncs start="val" end="val" steps="val">
      numeric observer color matching function data goes here
   </ObserverFuncs>
   <StdIlluminant>Illuminant D50</StdIlluminant>
   <ColorTemperature>5000</ColorTemperature>
   <IlluminantSPD start="val" end="val" steps="val">
      numeric illuminant data goes here
   </IlluminantSPD>
   <SurroundXYZ X="val" Y="val" Z="val"/>
</spectralViewingConditionsType>
```

XML tagArrayType tag structure

```
<tagArrayType>
  <ArraySignature>sig</ArraySignature>
  <ArrayTags>
    <tagType>
      <!-- tag 1 data -->
    </tagType>
    …
    <tagType>
      <!-- tag N data -->
    </tagType>
  </ArrayTags>
</tagArrayType>
```

XML tagStructureType tag structure

```
<tagStructureType>
   <StructureSignature>sig</StructureSignature>
   <MemberTags>
      < tag_1_Tag > <tag_1_Type>
         <!-- tag 1 data -->
      </tag_1_Type> </tag_1_Tag>
      …
      <tag_N_Tag > <tag_N_Type>
            <!-- tag N data -->
      </tag_N_Type> </tag_N_Tag>
   </MemberTags>
</tagStructureType>
```

- Note: Uses same XML encoding for tag data sharing and private tags as for profile tags

XML multiProcessElementType tag structure

```
<multiProcessElementType>
   <TagSignature>sig</TagSignature>
   <MultiProcessElements InputChannels="in" OutputChannels="out">
      <!-- element 1 -->
      …
      <!-- element N -->
   </MultiProcessElements>
</multiProcessElementType>
```

Where elements can be any of the following:

Basic processing element structure

```
<elementTypeName InputChannels="val" OutputChannels="val">
   <!-- data entries appropriate for element type -->
</elementTypeName>
```

Where the values of *elementTypeName* match the sub-section titles for processing elements defined in section 11.2 of the iccMAX specification

curveSetElement structure

```
<curveSetElement InputChannels="val" OutputChannels="val">
   <SegmentedCurve>
      <!– Segment entries for curve 1 -->
   </SegmentedCurve
   …
 <SegmentedCurve>
      <!– Segment entries for curve N -->
   </SegmentedCurve
</ CurveSetElement >
```

Where segment entries can be any of the following:
SingleSampledSegment, FormulaSegment, SampledSegment

matrixElement structure

```
<matrixElement InputChannels="val" OutputChannels="val">
   <MatrixData>
      <!– matrix data with InputChannels columns and OutputChannels rows -->
   </MatrixData>
   <OffsetData>
      <!– offset data OutputChannels entries -->
   </OffsetData>
</matrixElement >
```

Note: OffsetData block is optional and can be omitted if zero

tintArrayElement structure

```
<tintArrayElement InputChannels="1" OutputChannels="val">
   <numberType>
      <!– numberType data with a multiple of OutputChannels values -->
   </numberType>
</TintArrayElement>
```

Were *numberType* can be any of the following:

float16NumberType, float32NumberType, or float64NumberType, s15Fixed16NumberType, u16Fixed16NumberType, uInt16NumberType, uInt32NumberType, uInt64NumberType, uInt8NumberType

calculatorElement structure

```
<calculatorElement InputChannels="val" OutputChannels="val">
   <SubElements>
      <!–- array of processing elements -->
   </SubElements>
   <MainFunction>
     <!–- main function script -->
   </MainFunction
</calculatorElement>
```

- The "main function script" executes a sequence of vector-based operations using a data stack (with access to a temporary memory array) to transform input channels (via the *in* operator) into output channels (via the *out* operator)
- Operators in the script can invoke transforms in sub-elements
- Annex F in iccMAX specdifiation describes text representation used in XML encoding of MainFunction

Extended Calculator Element XML Encoding

Introducing Calc++

CalculatorElement programming challenges

- As defined by iccMAX specification:
  - Temporary memory variables as well as input/output channels are indexed by position
  - Sub-elements are indexed by position
  - MainFunction is monolithic
    - Unwieldy without much consideration for code reuse
    - No concept of functional libraries
- Net result:
  - It is easy to confuse things and code is difficult to follow

Extending calculator elements

- XML parsing is separate from binary profile representation
- Extensions to parsing need not involve changes to profile format
- Additions to Calculator Element XML encoding:
  - Importing calculator "data & code" from separate files
  - Addressing of temporary memory and input/output channels as variable names
  - Named script macros
  - Addressing of sub-elements by name
- Note: This provides level of obfuscation in resulting binary ICC profile

Extended structure of CalculatorElemant XML

```
<CalculatorElement InputChannels=”in”
                   OutputChannels=”out”
                   InputNames=”x0 x1 …”
                   OutputNames=”y0 y1 …”>
     <Imports> … </Imports>
     <Variables> … </Variables>
     <Macros> … </Macros>
     <SubElements>… </SubElements>

     <MainFunction>
          Extended Representation of Operations
     </MainFunction>
</CalculatorElement>
```

Import Encoding

```
<Imports>
        <Import Filename=”file_specifier_1.xml”/>
        <Import Filename=”file_specifier_2.xml”/>
        …
</Imports>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<IccCalcImport>
        <Imports>…</Imports>
        <SubElements>…Named SubElements…</SubElements>
        <Variables>…</Variables>
        <Macros>…</Macros>
</IccCalcImport>
```

Variable encoding

```
<Variables>
        <Declare Name=”myVar”/>
        <Declare Name=”myVector” Size=6/>
        <Declare Name=”myStruct”>m1 m2[3] m3</Declare>
</Variables>
```

Accessing variables in calculator scripts:

```
tget{myVar}
tput{myVector}
tsav{myStruct}
tget{myVector[3]}
tput{myVector[4,2]}
tsav{myStruct.m3}
tsav{myStruct.m2}
tget{myStruct.m2[2]}
tput{myStruct.m2[1,2]}
```

XML encoding of calulator elements

Macro Encoding

```
<Macros>
		<Macro Name=”macro1”>Text defining macro1 operator sequence</Macro>
		<Macro Name=”macro2”>Text defining macro2 operator sequence</Macro>
		…
</Macros>
```

```
<CalculatorElement InputChannels=”1” OutputChannels=”1”>
		<Macros>
				<Macro Name=”odd”>1 3 5 5 3 1</Macro>
				<Macro Name=”evenoddeven”>2 4 6 #odd 6 4 2</Macro>
		</Macros>

		<MainFunction>{ in[0] call{evenoddeven} sum(13) out[0] }</MainFunction>
</CalculatorElement>
```

```
<CalculatorElement InputChannels=”1” OutputChannels=”1”>
		<MainFunction>{ in[0] 2 4 6 1 3 5 5 3 1 6 4 2 sum(13) out[0] }</MainFunction>
</CalculatorElement>
```

XML encoding of calulator elements

Local Variables in Macros

```
<Macros>
        <Macro Name="macro1" Local="var1 … varN">
                Text defining macro1 operator sequence
        </Macro>
</Macros>
```

Example macro definitions:

```
<Macro Name="first_clamp3" Local="lower upper">
        tput{@upper} tput{@lower} tget{@upper} copy[1,2] vmin(3) tget{@lower} copy[1,2] vmax(3)
</Macro>

<Macro Name="second_clamp3" Local="range[2]">
        tput{@range} tget{@range[1]} copy[1,2] vmin(3) tget{@range[0]} copy[1,2] vmax(3)
</Macro>
```

Named Sub-Elements

```
<SubElements>
        <CurveSetElement Name="applyGamma"
                InputChannels="3" OutputChannels="3"> …  </CurveSetElement>
        <MatrixElement Name="RGBtoXYZ"
                InputChannels="3" OutputChannels="3"> …
        </MatrixElement>
</SubElements>
```

```
<MainFunction>
        { in[3] curv{applyGamma} mtx{RGBtoXYZ} out[3]}
</MainFunction>
```

THE FUTURE OF COLOR MANAGEMENT

Thank You

Questions?