XML SERIALIZATION OF APL DATA ARRAYS

Summary:

APL data arrays can incorporate structure which cannot necessarily be represented in non-APL environments, such as some client and web server interactions. In addition the internal representation of data may differ in each implementation of the APL programming language.

Serializing an APL data array to an XML string makes that data accessible to any environment in a manner independent of the internal representation of the values in that APL data array. However, XML serializations are generally verbose.

When an APL data array is XML-serialized care must be taken to preserve the significant digits of numeric values. XML-serialized character data values must use a shared common representation, such as Unicode, so that the information is properly interpreted when de-serialized.

This document describes a methodology for transmitting information between APL+Win and VisualAPL using the VisualAPL-format for XML serialization of APL data arrays.

XML Serialization in VisualAPL:

VisualAPL includes an XML serialization/de-serialization feature to represent APL data arrays as an XML string. An APL data array can be serialized to an XML-format string which represents the values and structure of that APL data array.

As an XML-format string, the initial XML tag is the XML ‘declaration’, <?xml version="1.0" encoding="utf-8"?>. Since VisualAPL is a .Net programming language it is inherently Unicode-based, so the ‘utf-8’ encoding is used in the XML representation of the APL data array.

The XML ‘root’ element of the XML serialization is the <cvar>…</cvar> tag representing the dynamically-typed ‘Cielo variable’ data type of VisualAPL.

Subordinate to the ‘root’ ‘cvar’ element are two XML elements, ‘shape’ and ‘avar’.

The <shape>…</shape> tag describes the shape of the APL data array. Subordinate to the ‘shape’ tag are the <arrayofint><int>…</int>…</arrayofint> tags which specify the integer vector shape elements of the APL data array.

The values of the APL data array is enclosed in the <avar>…</avar> tag. Because APL data arrays can be nested, there can exist XML tags which are subordinate to the ‘avar’ tag including:

string
double
int
ArrayOfInt
ArrayOfDouble
ArrayOfcvar

Since VisualAPL supports arrays of .Net data types beyond those found in traditional APL, those .Net data types which are XML serializable may also be included in the 'avar' tag of an APL data array. Whereas the XML serialization of these data types can be received by APL+Win, their de-serialization to APL+Win objects will not be possible because APL+Win does not support these .Net data types.

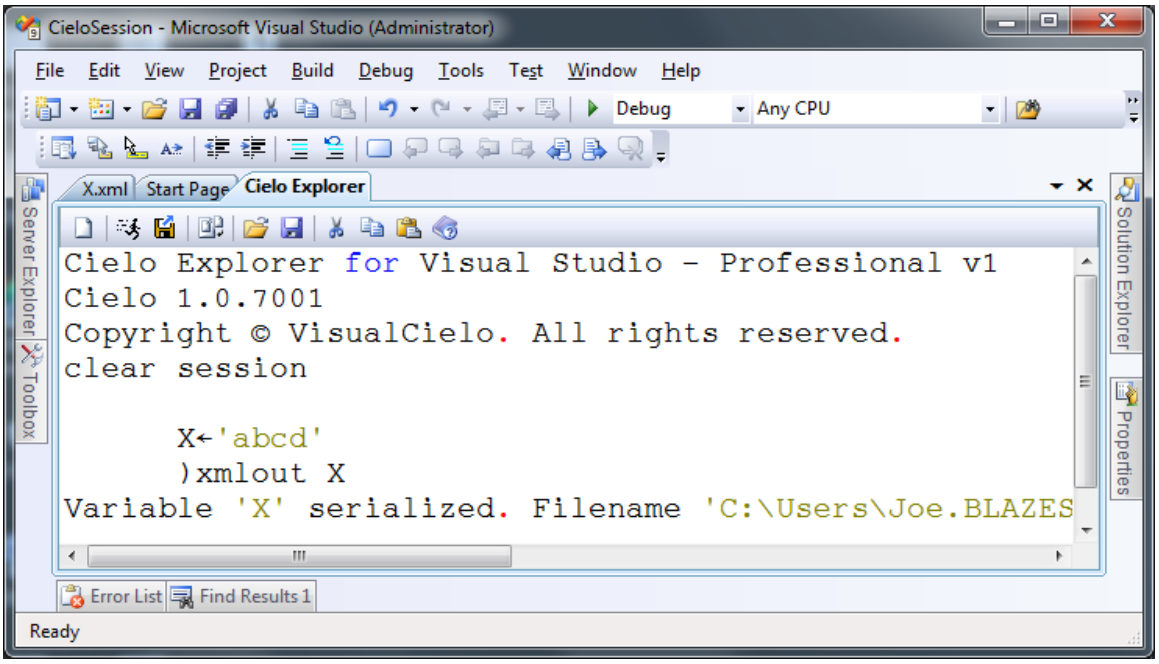
Thus an XML serialized APL data array in VisualAPL format has the structure:

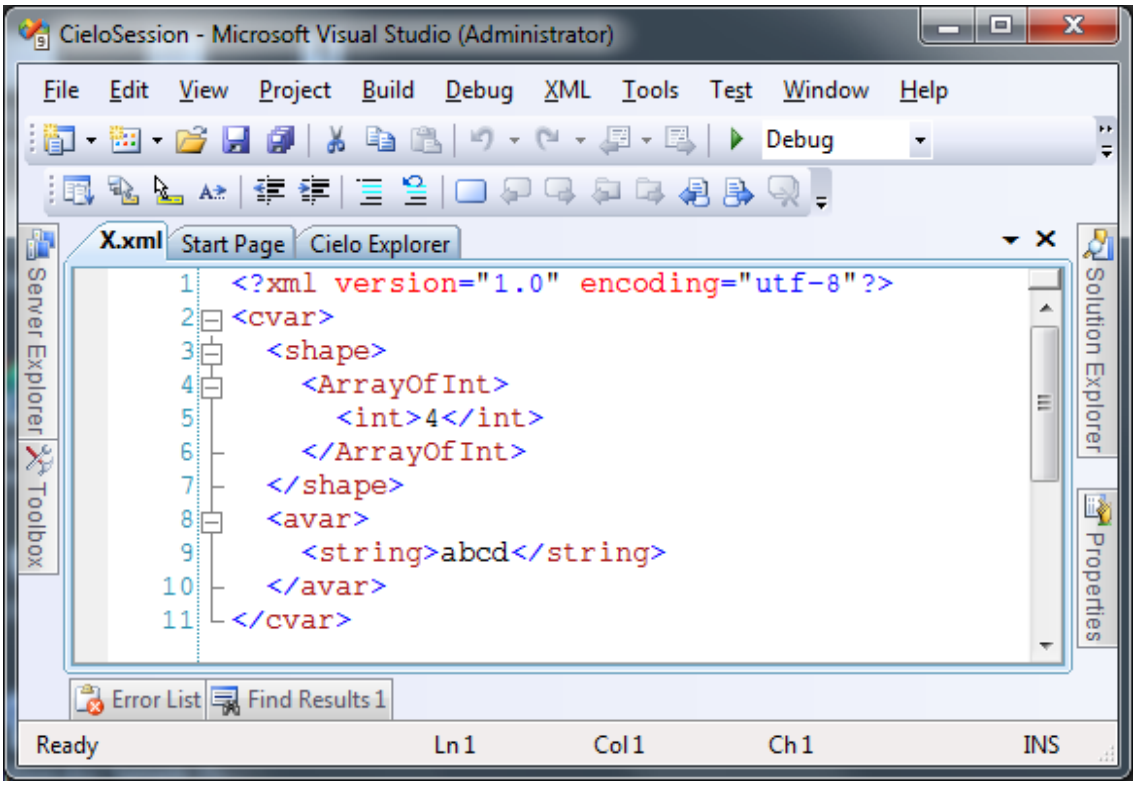
```
<cvar>
    <shape>
        <arrayofint>
            <int>…</int>
            …
        </arrayofint>
    </shape>
    <avar>
        …
    </avar>
</cvar>
```

VisualAPL XML-Serialization Examples:

The)xmlout system command available in the VisualAPL CieloExplorer interactive session supports the XML-serialization of APL variables. The XML-serialization facility can also be used under program control so that APL data objects can be exchanged between different implementations of APL. For further information on xml-format serialization in VisualAPL, refer to the APL2000 Forum topic: http://forum.apl2000.com/viewtopic.php?t=475. For simplicity of illustration, the following examples illustrate the VisualAPL ')xmlout' system command within the immediate mode Cielo Explorer session.

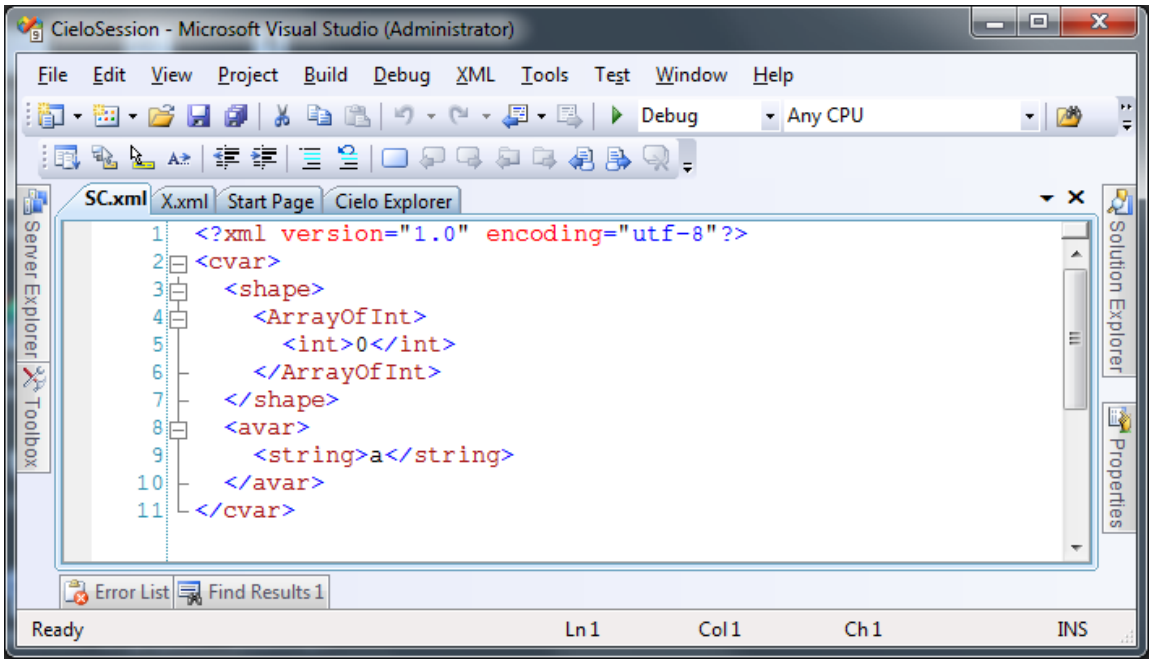
Example #1: APL string vector




CieloSession - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug XML Tools Test Window Help
Debug
X.xml
Start Page
Cielo Explorer
<?xml version="1.0" encoding="utf-8"?>
<cvar>
<shape>
<ArrayOfInt>
<int>4</int>
</ArrayOfInt>
</shape>
<avar>
<string>abcd</string>
</avar>
</cvar>
Server Explorer
Toolbox
Solution Explorer
Properties
Error List
Find Results 1
Ready
Ln 1
Col 1
Ch 1
INS

Example #2: Scalar character

```
SC←'a'
)xmlout SC
Variable 'SC' serialized. Filename …
```



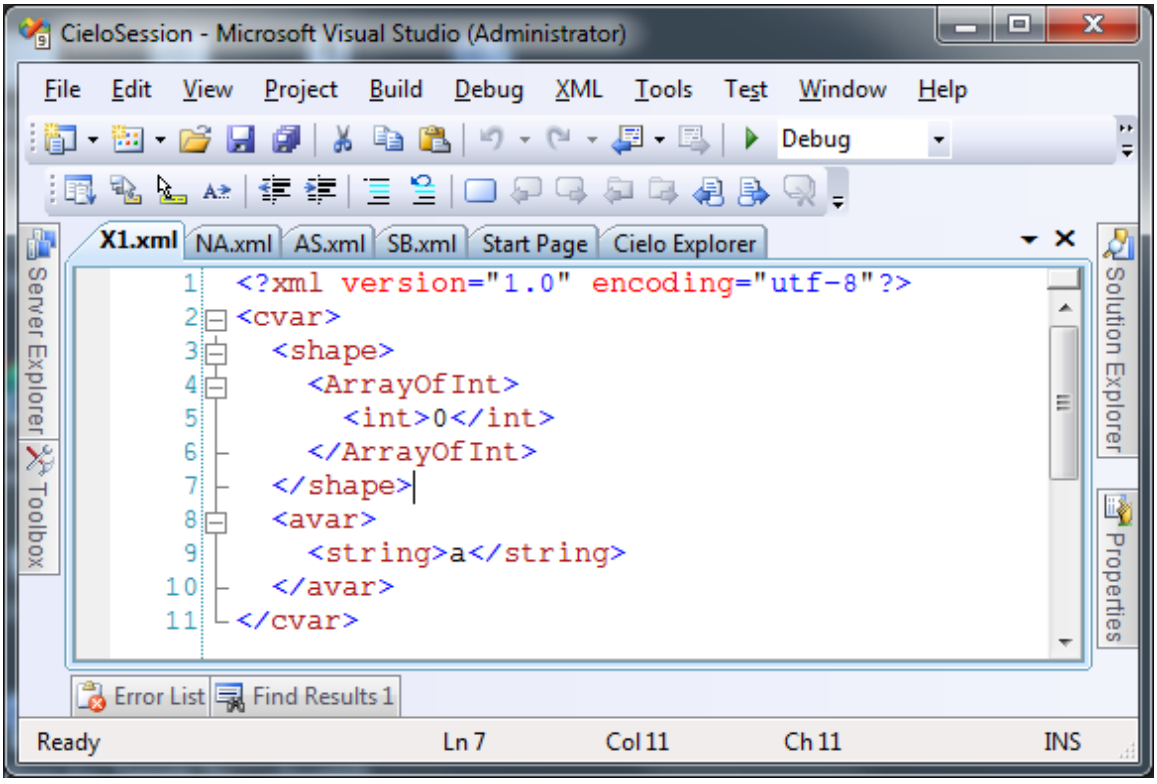
Example #3: Enclosed scalar character

This produces the same serialization result as scalar character

X1←⊂'a'

)xmlout X1

Variable 'X1' serialized. Filename…

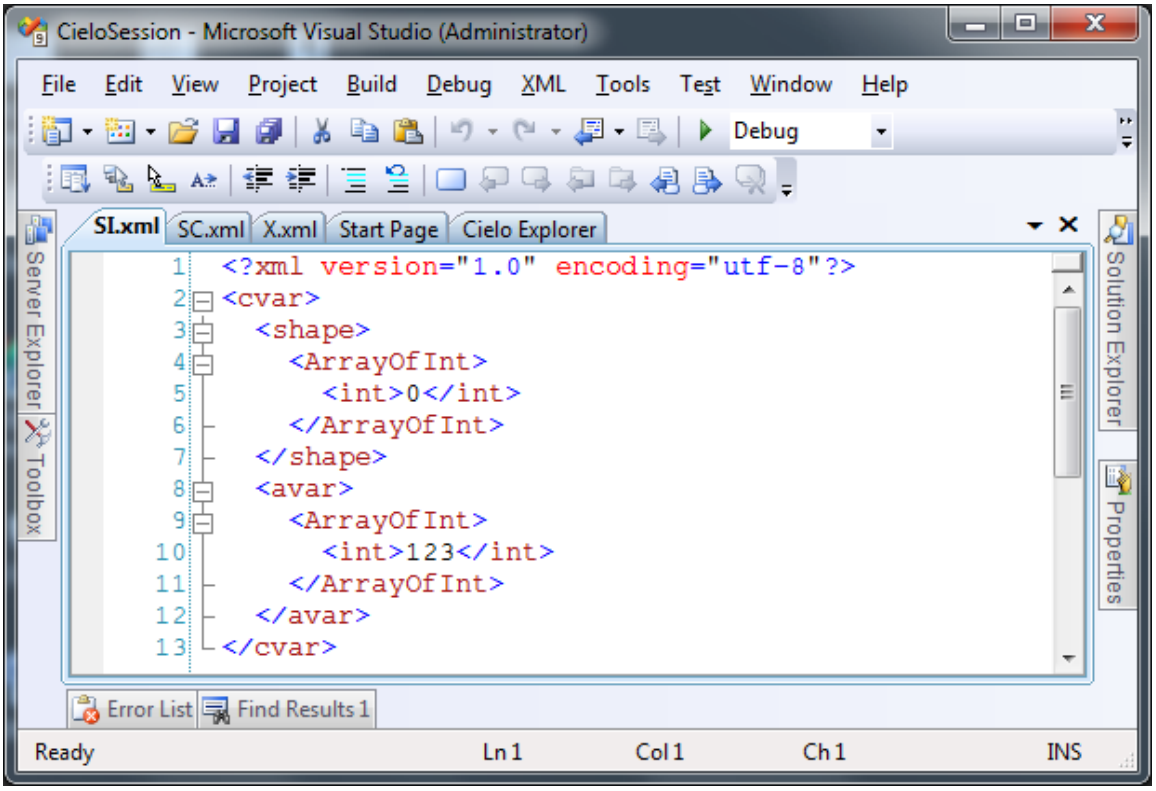


Example #4: Scalar integer

SI←123

)xmlout SI

Variable 'SI' serialized. Filename …

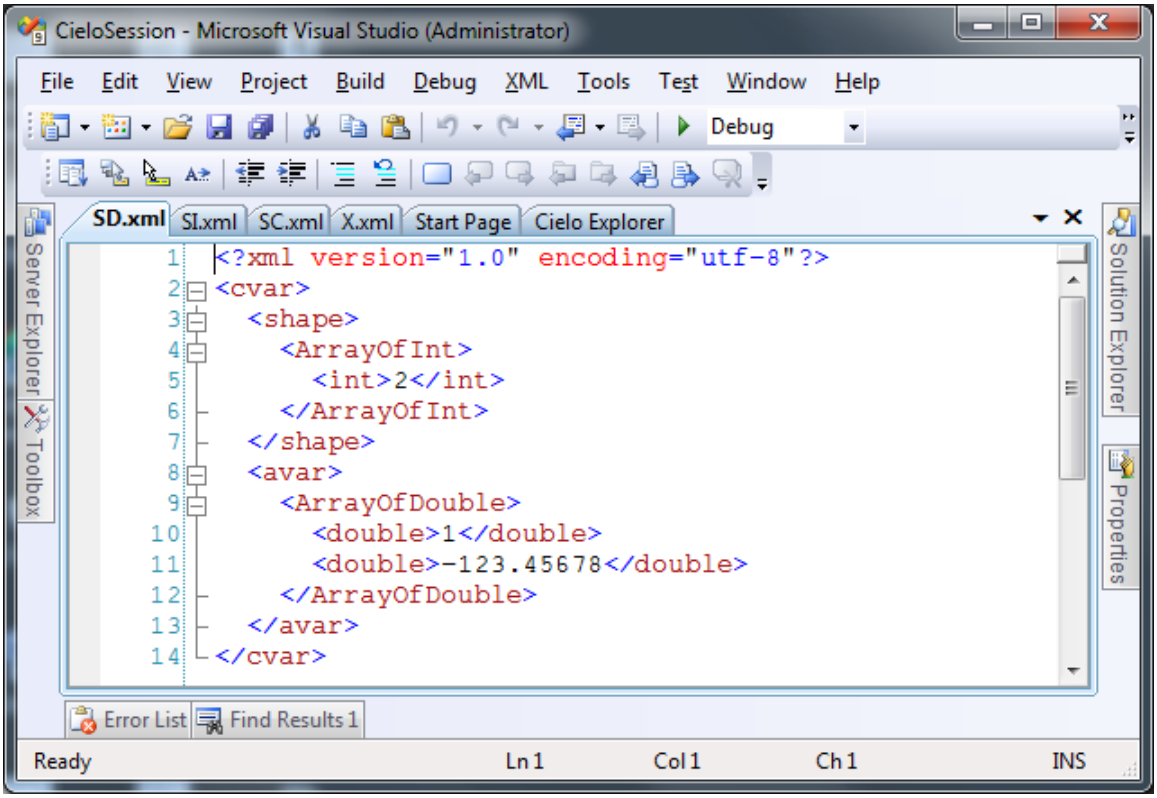


Example #5: Vector of doubles

SD←1 ¯123.45678

)xmlout SD

Variable 'SD' serialized. Filename…

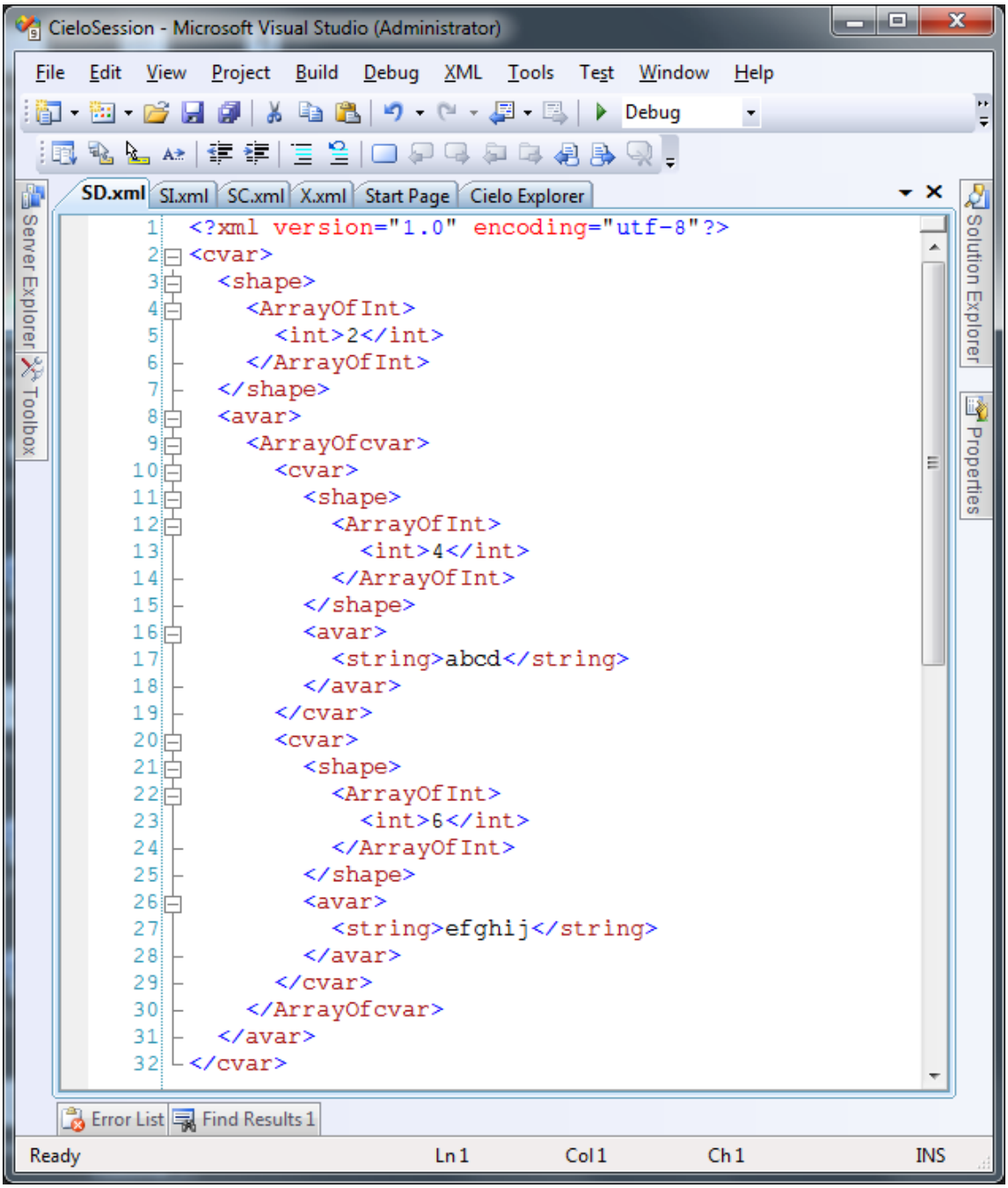


Example #6: Vector of string vectors

SD←'abcd' 'efghij'

)xmlout SD

Variable 'SD' serialized. Filename…

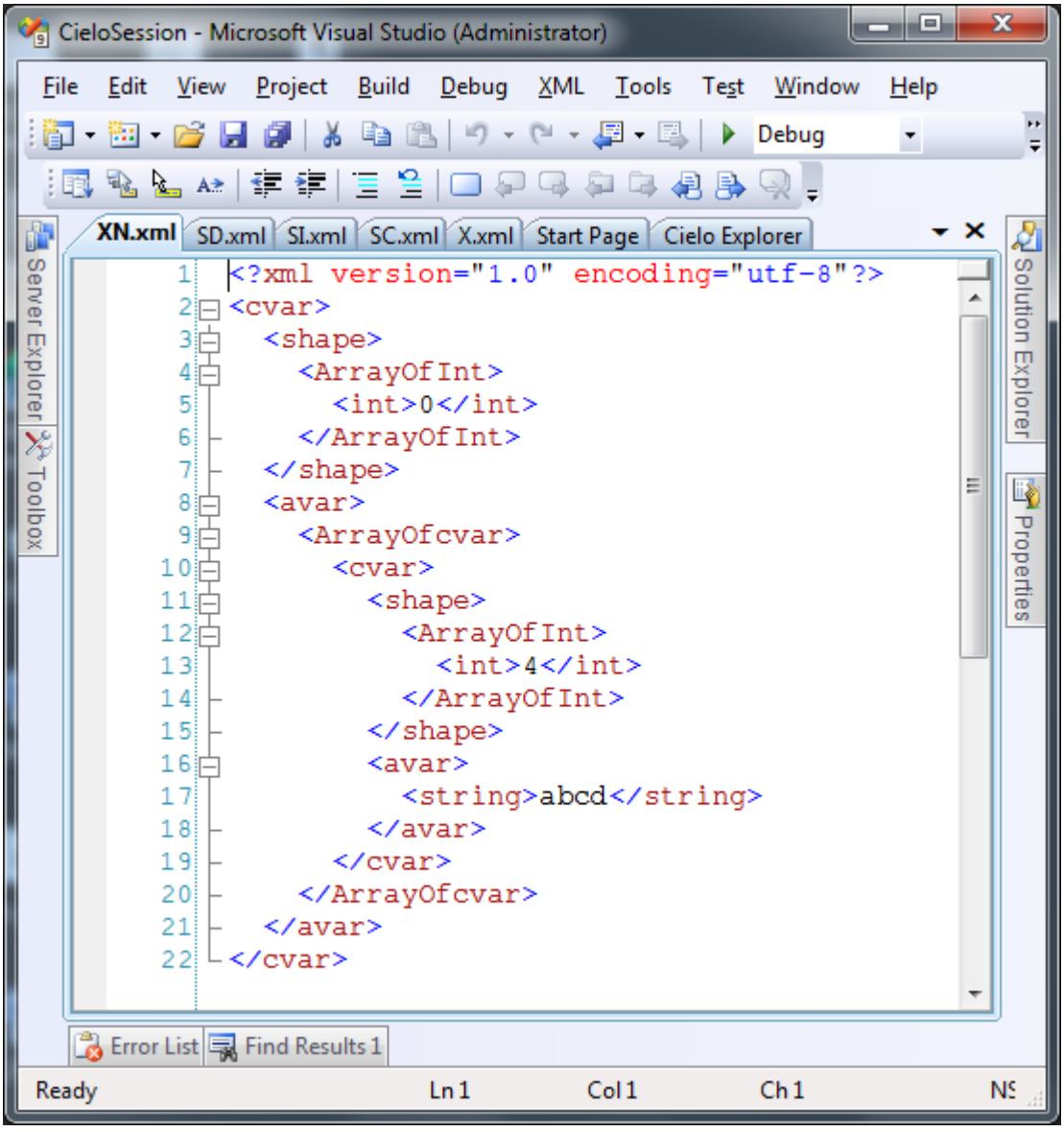


Example #7: Enclosed string vector

XN←⊂'abcd'

)xmlout XN

Variable 'XN' serialized. Filename...

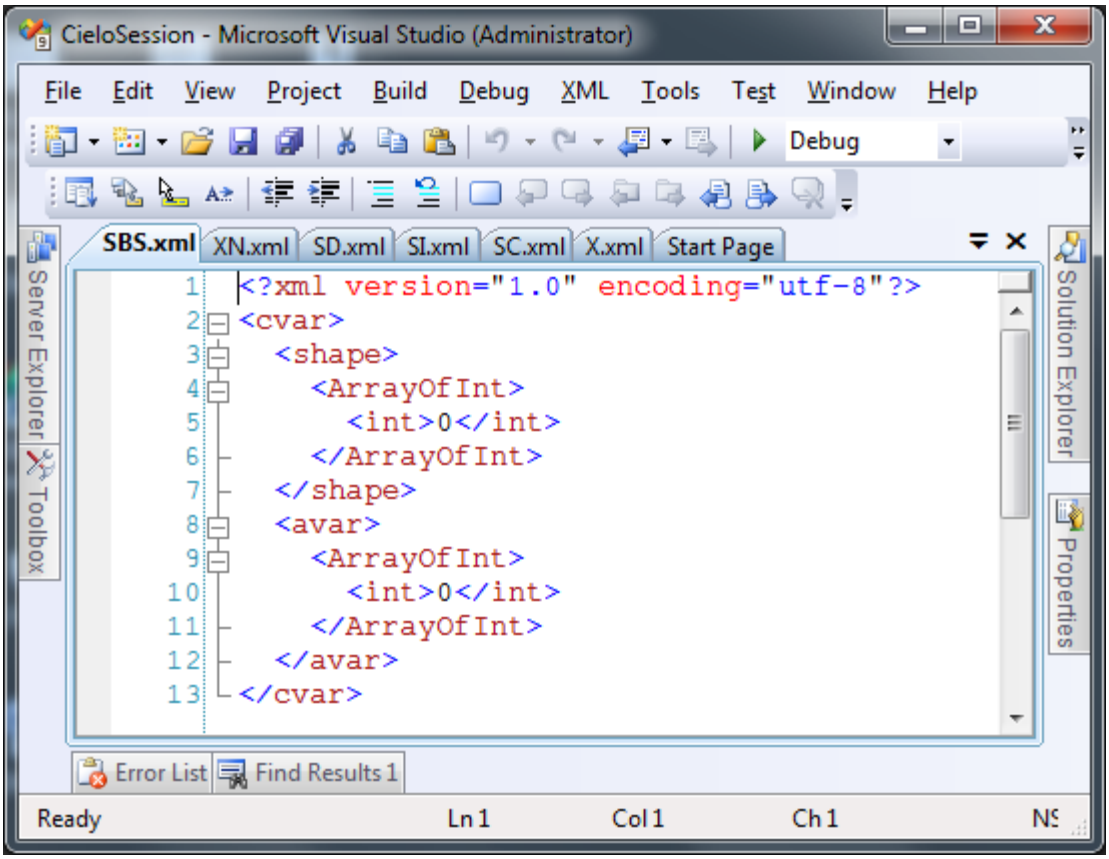


Example #8: Scalar APL Boolean serialized as scalar integer

SBS←0

)xmlout SBS

Variable 'SBS' serialized. Filename…

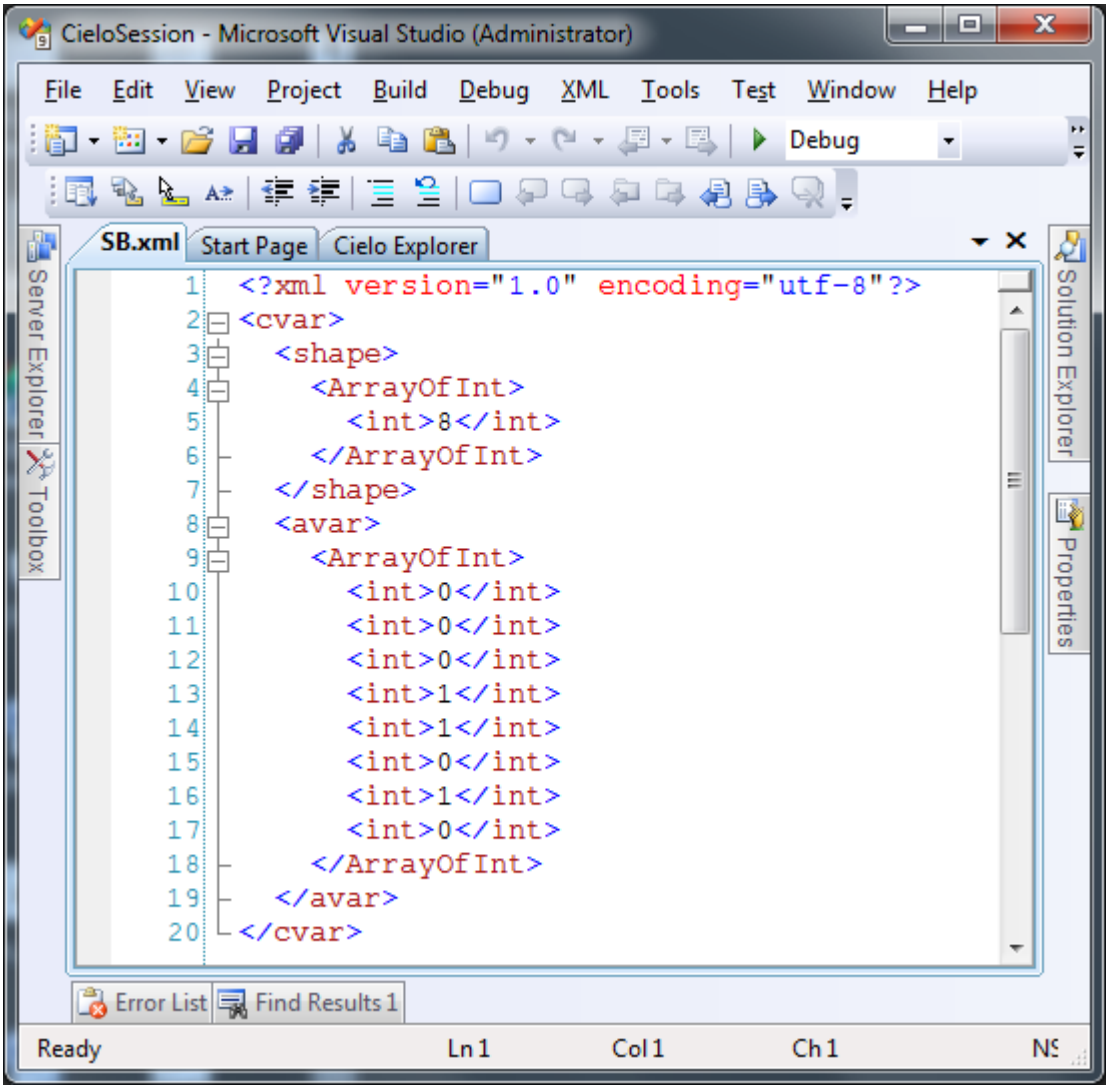


Example #9: Boolean vector serialized as integer vector

SB←0 0 0 1 1 0 1 0

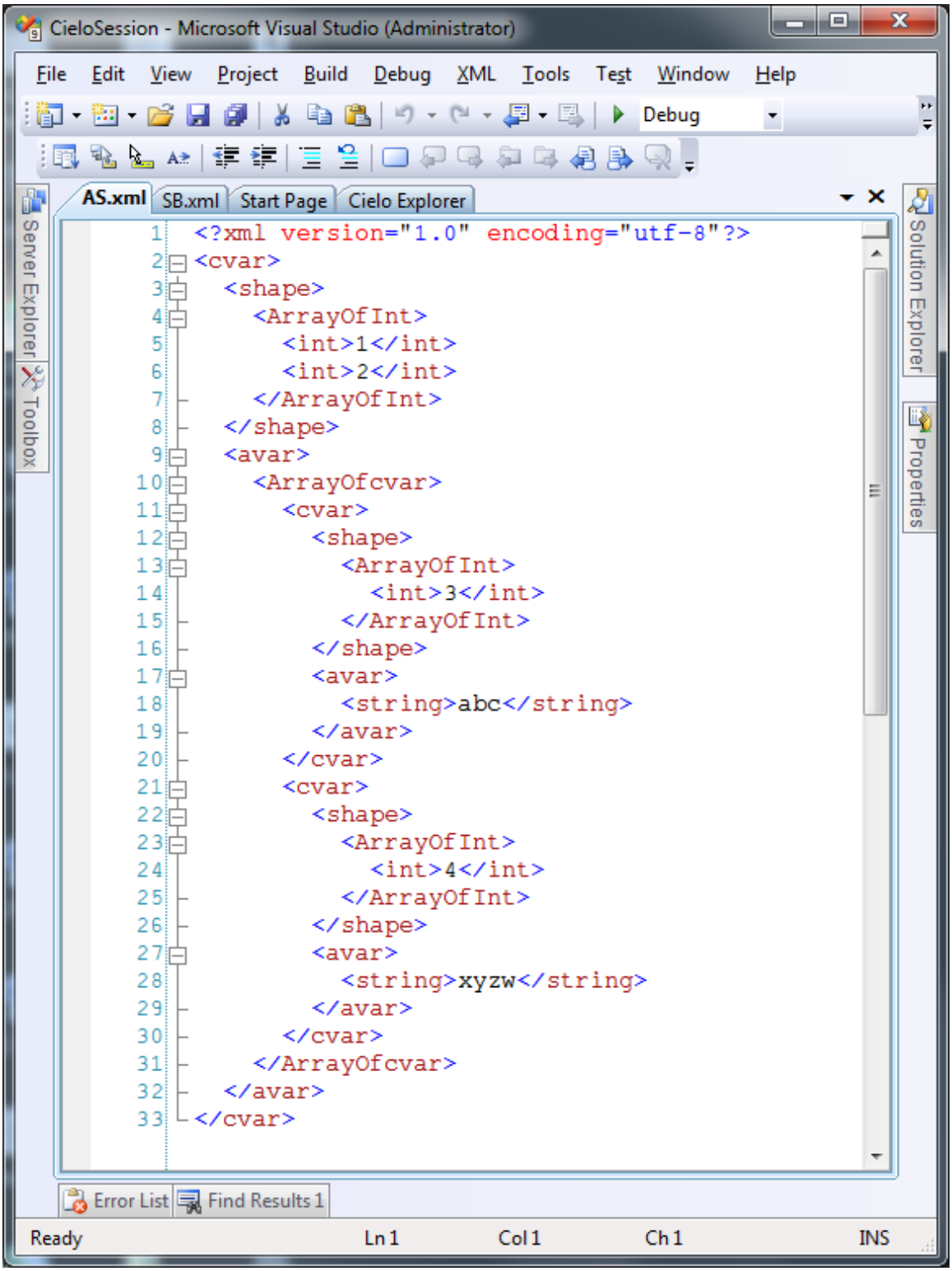
)xmlout SB

Variable 'SB' serialized. Filename…



Example #10: Array of string vectors

```
AS←1 2⍴'abc' 'xyzw'
)xmlout AS
Variable 'AS' serialized. Filename…
```


CieloSession - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug XML Tools Test Window Help
Debug
AS.xml SB.xml Start Page Cielo Explorer
Server Explorer
Toolbox
Solution Explorer
Properties
<?xml version="1.0" encoding="utf-8"?>
<cvar>
<shape>
<ArrayOfInt>
<int>1</int>
<int>2</int>
</ArrayOfInt>
</shape>
<avar>
<ArrayOfcvar>
<cvar>
<shape>
<ArrayOfInt>
<int>3</int>
</ArrayOfInt>
</shape>
<avar>
<string>abc</string>
</avar>
</cvar>
<cvar>
<shape>
<ArrayOfInt>
<int>4</int>
</ArrayOfInt>
</shape>
<avar>
<string>xyzw</string>
</avar>
</cvar>
</ArrayOfcvar>
</avar>
</cvar>
Error List
Find Results 1
Ready
Ln 1
Col 1
Ch 1
INS

Example #11: Nested array

```
NA←1.2345 ("abc" "xyzw" 3) (2 3⍴6)
)xmlout NA
Variable 'NA' serialized. Filename…

<?xml version="1.0" encoding="utf-8"?>
<cvar>
 <shape>
  <ArrayOfInt>
   <int>3</int>
  </ArrayOfInt>
 </shape>
 <avar>
  <ArrayOfcvar>
   <cvar>
    <shape>
     <ArrayOfInt>
      <int>0</int>
     </ArrayOfInt>
    </shape>
    <avar>
     <ArrayOfDouble>
      <double>1.2345</double>
     </ArrayOfDouble>
    </avar>
   </cvar>
   <cvar>
    <shape>
     <ArrayOfInt>
      <int>3</int>
     </ArrayOfInt>
    </shape>
    <avar>
     <ArrayOfcvar>
      <cvar>
       <shape>
        <ArrayOfInt>
         <int>3</int>
        </ArrayOfInt>
       </shape>
```

```
      <avar>
       <string>abc</string>
      </avar>
     </cvar>
     <cvar>
      <shape>
       <ArrayOfInt>
        <int>4</int>
       </ArrayOfInt>
      </shape>
      <avar>
       <string>xyzw</string>
      </avar>
     </cvar>
     <cvar>
      <shape>
       <ArrayOfInt>
        <int>0</int>
       </ArrayOfInt>
      </shape>
      <avar>
       <ArrayOfInt>
        <int>3</int>
       </ArrayOfInt>
      </avar>
     </cvar>
    </ArrayOfcvar>
   </avar>
  </cvar>
  <cvar>
   <shape>
    <ArrayOfInt>
     <int>0</int>
    </ArrayOfInt>
   </shape>
   <avar>
    <ArrayOfInt>
     <int>2</int>
    </ArrayOfInt>
   </avar>
  </cvar>
</ArrayOfcvar>
```

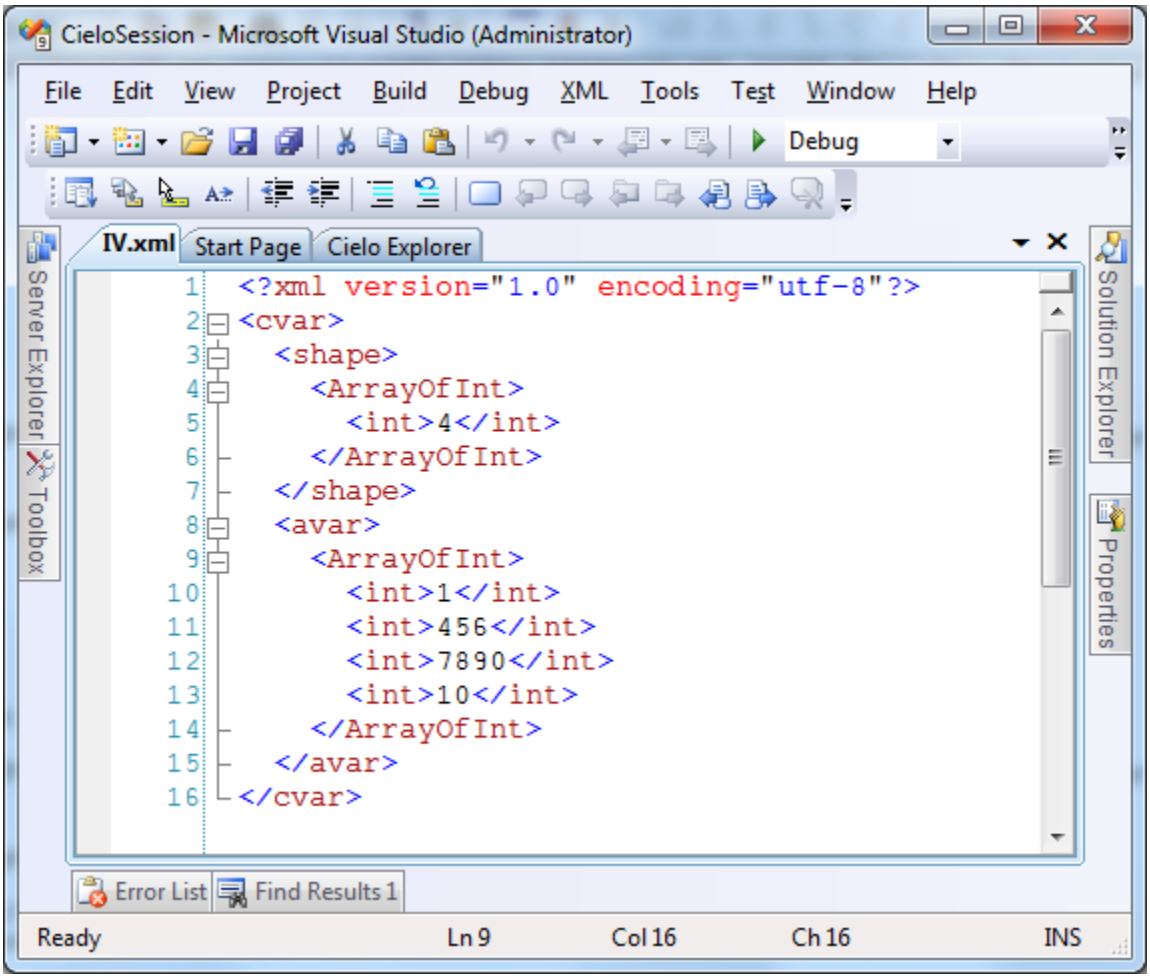
</avar>
</cvar>

Example #12: Integer vector

IV←1 456 7890 10

)xmlout IV

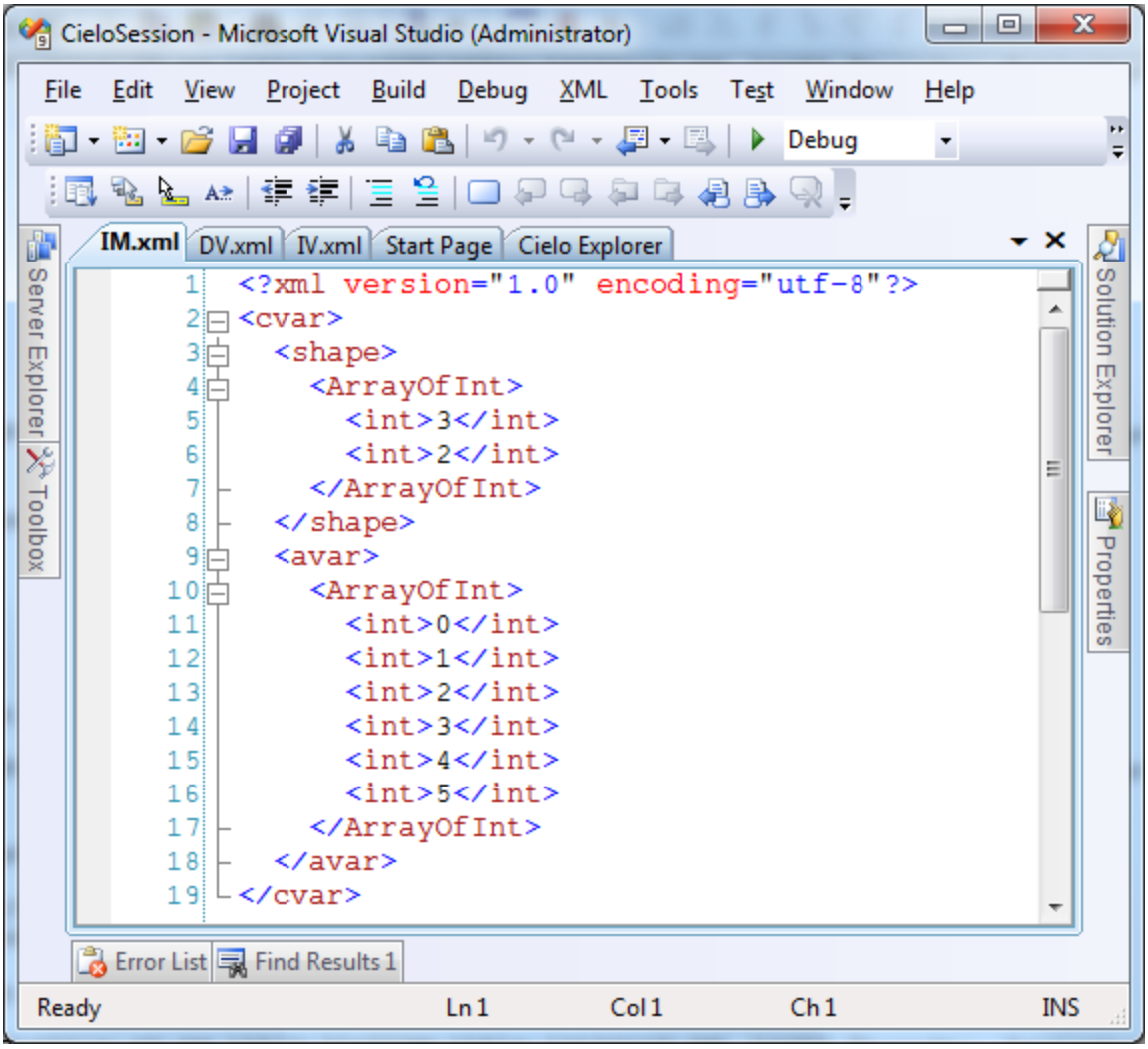
Variable 'IV' serialized. Filename…



Example #13: Array of integers

```
IM←3 2⍴⍳6
    )xmlout IM
```

Variable 'IM' serialized. Filename…



Example #14: Heterogeneous array

```
HET←1'abc' 2 3 4 'vb'
    □dr HET
807
    )xmlout HET
Variable 'HET' serialized. Filename…

<?xml version="1.0" encoding="utf-8"?>
<cvar>
 <shape>
  <ArrayOfInt>
   <int>6</int>
  </ArrayOfInt>
 </shape>
 <avar>
  <ArrayOfcvar>
   <cvar>
    <shape>
     <ArrayOfInt>
      <int>0</int>
     </ArrayOfInt>
    </shape>
    <avar>
     <ArrayOfInt>
      <int>1</int>
     </ArrayOfInt>
    </avar>
   </cvar>
   <cvar>
    <shape>
     <ArrayOfInt>
      <int>3</int>
     </ArrayOfInt>
    </shape>
    <avar>
     <string>abc</string>
    </avar>
   </cvar>
   <cvar>
    <shape>
```

```
    <ArrayOfInt>
     <int>0</int>
    </ArrayOfInt>
   </shape>
   <avar>
    <ArrayOfInt>
     <int>2</int>
    </ArrayOfInt>
   </avar>
  </cvar>
  <cvar>
   <shape>
    <ArrayOfInt>
     <int>0</int>
    </ArrayOfInt>
   </shape>
   <avar>
    <ArrayOfInt>
     <int>3</int>
    </ArrayOfInt>
   </avar>
  </cvar>
  <cvar>
   <shape>
    <ArrayOfInt>
     <int>0</int>
    </ArrayOfInt>
   </shape>
   <avar>
    <ArrayOfInt>
     <int>4</int>
    </ArrayOfInt>
   </avar>
  </cvar>
  <cvar>
   <shape>
    <ArrayOfInt>
     <int>2</int>
    </ArrayOfInt>
   </shape>
   <avar>
    <string>vb</string>
```

```
      </avar>
     </cvar>
    </ArrayOfcvar>
  </avar>
</cvar>
```

Example #15: Zilde

```
X0←⍬
     )xmlout X0
Variable 'X0' serialized. Filename…

<?xml version="1.0" encoding="utf-8"?>
<cvar>
  <shape>
    <ArrayOfInt>
      <int>0</int>
    </ArrayOfInt>
  </shape>
  <avar>
    <ArrayOfInt />
  </avar>
</cvar>
```

Example #16: Empty string

```
X99←''
     )xmlout X99
Variable 'X99' serialized. Filename…

<?xml version="1.0" encoding="utf-8"?>
<cvar>
  <shape>
    <ArrayOfInt>
      <int>0</int>
    </ArrayOfInt>
  </shape>
  <avar>
    <string />
  </avar>
</cvar>
```

Example #17: Unicode points specification of character data values

Consider the following XML string which represents the scalar character “A” as the Unicode code point ‘A’:

```
<?xml version="1.0" encoding="utf-8"?>
<cvar>
```

<shape/>
<avar>
<string>A</string>
</avar>
</cvar>

If this XML string is provided to VisualAPL it will be de-serialized to the scalar character “A”. The Unicode code point ‘&#....;’ specification in the XML string enables unambiguous representation of character values between different environments. This XML syntax is not unique to VisualAPL, but is an XML standard.

Example #18: XML serialization of APL+Win []av using the Unicode code point specification method

```
<?xml version="1.0" encoding="utf-8"?>
<cvar>
 <shape>
  <ArrayOfInt>
   <int>256</int>
  </ArrayOfInt>
 </shape>
 <avar>

<string>&#0000;&#0001;&#0002;&#9079;&#8900;&#0168;&#8592;&#0007;&#0008;&#0009;&#0010;&
#8834;&#0012;&#0013;&#8835;&#9055;&#0016;&#0017;&#0018;&#9067;&#0020;&#0242;&#9068;&#
9077;&#8593;&#8595;&#8594;&#0027;&#8867;&#8866;&#9035;&#9042;&#0032;&#0033;&#0034;&#0
035;&#0036;&#0037;&#0038;&#0039;&#0040;&#0041;&#0042;&#0043;&#0044;&#0045;&#0046;&#00
47;&#0048;&#0049;&#0050;&#0051;&#0052;&#0053;&#0054;&#0055;&#0056;&#0057;&#0058;&#005
9;&#0060;&#8776;&#0062;&#0063;&#0064;&#0065;&#0066;&#0067;&#0068;&#0069;&#0070;&#0071;
&#0072;&#0073;&#0074;&#0075;&#0076;&#0077;&#0078;&#0079;&#0080;&#0081;&#0082;&#0083;&
#0084;&#0085;&#0086;&#0087;&#0088;&#0089;&#0090;&#0091;&#0092;&#0093;&#8743;&#0095;&#
0096;&#0097;&#0098;&#0099;&#0100;&#0101;&#0102;&#0103;&#0104;&#0105;&#0106;&#0107;&#0
108;&#0109;&#0110;&#0111;&#0112;&#0113;&#0114;&#0115;&#0116;&#0117;&#0118;&#0119;&#01
20;&#0121;&#0122;&#0123;&#0124;&#0125;&#8764;&#0127;&#0128;&#0129;&#0130;&#0131;&#013
2;&#0133;&#8800;&#0135;&#0136;&#0137;&#0138;&#0139;&#0140;&#8968;&#0142;&#8970;&#0144;
&#8710;&#0215;&#0147;&#0148;&#9109;&#0150;&#9054;&#9017;&#0153;&#0154;&#0155;&#0156;&
#0157;&#9066;&#0159; &#0161;&#0162;&#0163;&#0164;&#0165;&#9053;&#9024;&#8803;&#
9015;&#0170;&#0171;&#0172;&#0173;&#0174;&#0061;&#0176;&#0177;&#0178;&#0179;&#0180;&#0
181;&#0182;&#0183;&#0184;&#0185;&#0186;&#0187;&#0188;&#0189;&#0190;&#0191;&#0192;&#01
93;&#0194;&#0195;&#0196;&#0197;&#0198;&#0199;&#0200;&#0201;&#0202;&#0203;&#0204;&#020
5;&#0206;&#0207;&#0208;&#0209;&#0210;&#0211;&#0212;&#0213;&#0214;&#8778;&#0216;&#0217;
&#0218;&#0219;&#0220;&#0221;&#0222;&#0223;&#9082;&#0225;&#9075;&#0227;&#0228;&#9073;&
#8869;&#8868;&#9021;&#8854;&#9074;&#9023;&#8711;&#9033;&#8714;&#8745;&#8801;&#9049;&#
8805;&#8804;&#9045;&#9038;&#0247;&#0402;&#8728;&#9675;&#8744;&#9076;&#8746;&#0175;&#8
739;&#0255;</string>
 </avar>
</cvar>
```

There are 256 elements to []av as indicated by the shape tag of the serialization.

XML Serialization in APL+Win:

An APL+Win v10.1 workspace has been prepared which performs the XML serialization and de-serialization of APL data arrays in a manner analogous to that of VisualAPL. There are potentially an infinite number of alternative XML serialization methodologies, but supporting the VisualAPL method in APL+Win facilitates convenient exchange of complex APL arrays.

APL+Win (v10.1) functions in the SERIALIZETOVAPLXML.W3 workspace have been implemented to duplicate the XML serialization (SerializeToVAPLXML) and de-serialization (DeserializeFromVAPLXML) methodology of VisualAPL. These functions are inverses of each other. These APL+Win functions can be used to XML serialize APL+Win data arrays in VisualAPL format so that the resulting XML strings can be passed to VisualAPL for de-serialization to the equivalent VisualAPL arrays.

The operation of sending data serialized as XML from VisualAPL to APL+Win is also possible. Generally the 'HEX' or 'UC' left arguments to the APL+Win functions should be used. However, this operation is not recommended for sending VisualAPL function representations to APL+Win.

Exchanging text strings which are XML serializations of APL arrays between APL+Win and VisualAPL is easily accomplished using the APL+Win ActiveX Server interface because VisualAPL as a .Net language can utilize ActiveX components. To see examples of using this interface, go to: http://forum.apl2000.com/viewtopic.php?t=633. Using native files or a TCP/IP connection are other alternatives for exchanging such text strings between APL+Win and VisualAPL.

APL+Win is not natively Unicode-based. For most character values in the traditional 256-element ASCII character set, this presents no problem. However if the character values to be serialized include APL+Win 'special' characters, such as rho or epsilon, instead of representing these characters as a single glyph in the serialization, they are represented by the XML standard Unicode 'escape string', such as '&#9076' (for rho) or '∊' (for epsilon). This technique is used because APL+Win was developed when only 256 code points in a character set were available and glyphs for all 256 such code points had already been defined (e.g. ASCII), so the APL+Win 'special' characters were implemented by overloading certain ASCII code points which humans would not ordinarily type.

The SERIALIZETOVAPLXML.W3 APL+Win functions are not designed to pass APL+Win function representations to VisualAPL. The '\APL2UNICODE\' subdirectory installed with VisualAPL contains a workspace and associated ActiveX component which may be used for conversions of APL+Win functions to VisualAPL functions. To see examples of converting APL+Win functions to VisualAPL go to: http://forum.apl2000.com/viewtopic.php?t=455.

The APL+Win 'SerializeToVAPLXML' and 'DeserializeFromVAPLXML' functions have an optional left argument used to specify the serialization option for character data values. The options are:

'HEX' in which case the hex escape strings, e.g. ‘&xHH;’, will be used for ‘special’ ASCII characters.

'UC' in which case the Unicode escape strings, e.g. ‘&#UUUU;’, will be used for ‘special’ ASCII characters.

'VAPL' in which case the Unicode escape strings, e.g. ‘&#UUUU;’, will be used for all ASCII characters. This is the default value

The ‘special’ ASCII characters are those in []AV[1 – 32] and []AV[128 - 256] using APL+Win index origin 1.

Generally, APL+Win arrays serialized to XML strings using ‘HEX’ or ‘UC’ as the left argument to the ‘SerializeToVAPLXML’ function will be successfully de-serialized by VisualAPL when any string data contains standard ASCII characters, such as ‘0-9’, ‘a-z’, ‘A-Z’ and punctuation.

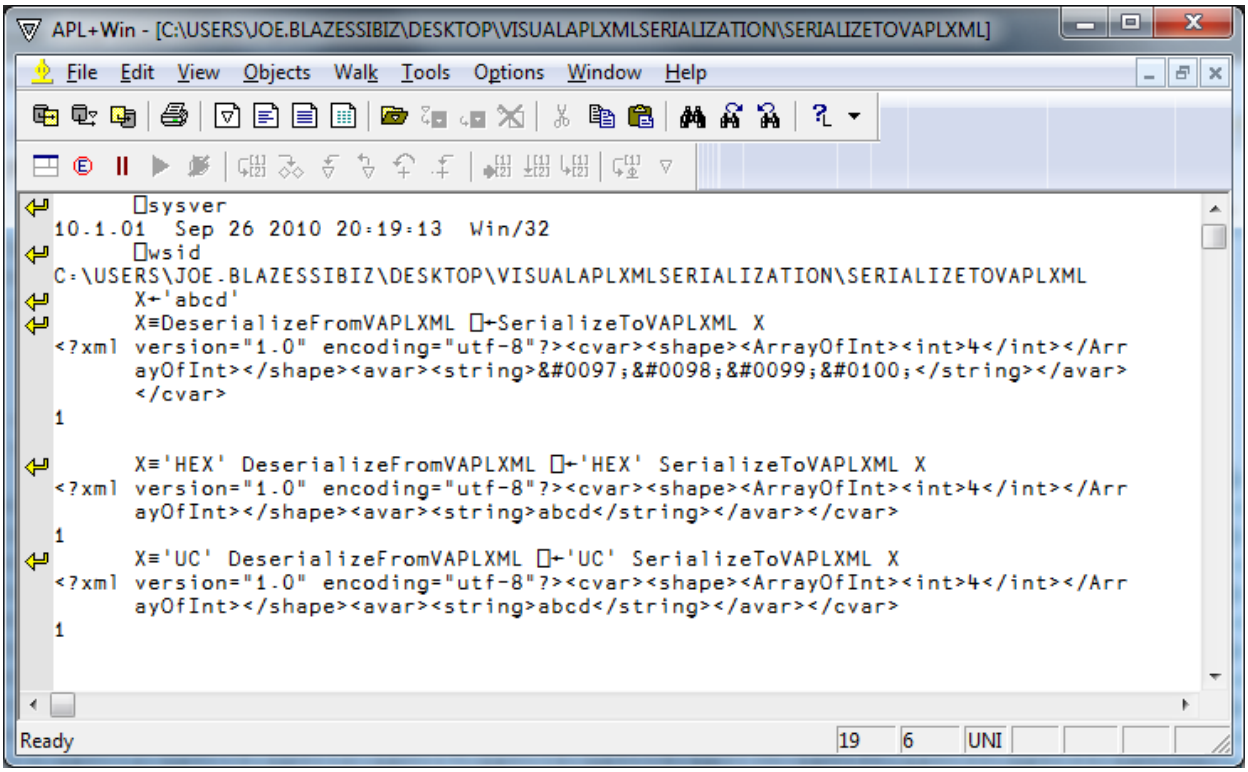
When ‘VAPL’ is used as the left argument to the ‘SerializeToAPLXML’ function, because it explicitly specifies the VisualAPL Unicode code points for any element of ‘[]av’, VisualAPL will be able to properly de-serialize all string (character) data values in an APL+Win data array.

VisualAPL is inherently Unicode-based, so there is no need for the encoding options provided in the APL+Win analogues of the VisualAPL XML serialization methods. Because of the special encoding options necessary in APL+Win, in some cases the APL+Win XML serialization will not be the same as the VisualAPL serialization. Nonetheless the will both result in the same APL array upon de-serialization.

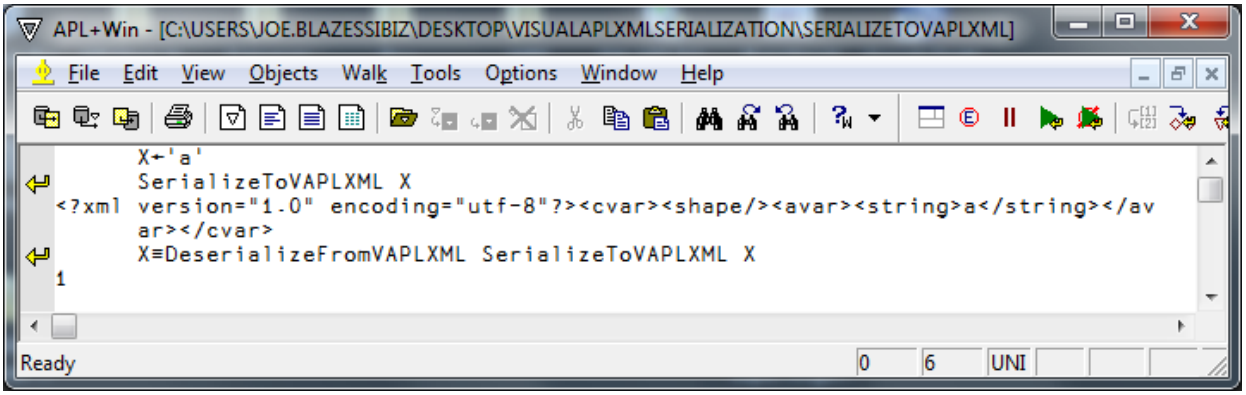
APL+Win XML Serialization Examples

These examples illustrate the APL+Win functions in the SERIALIZETOVAPLXML.W3 workspace.

Example #1: APL string vector

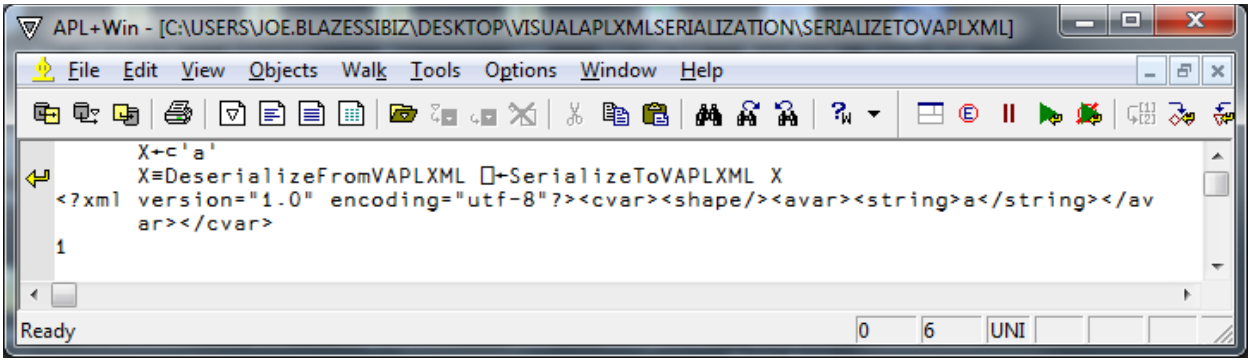


Example #2: Scalar character

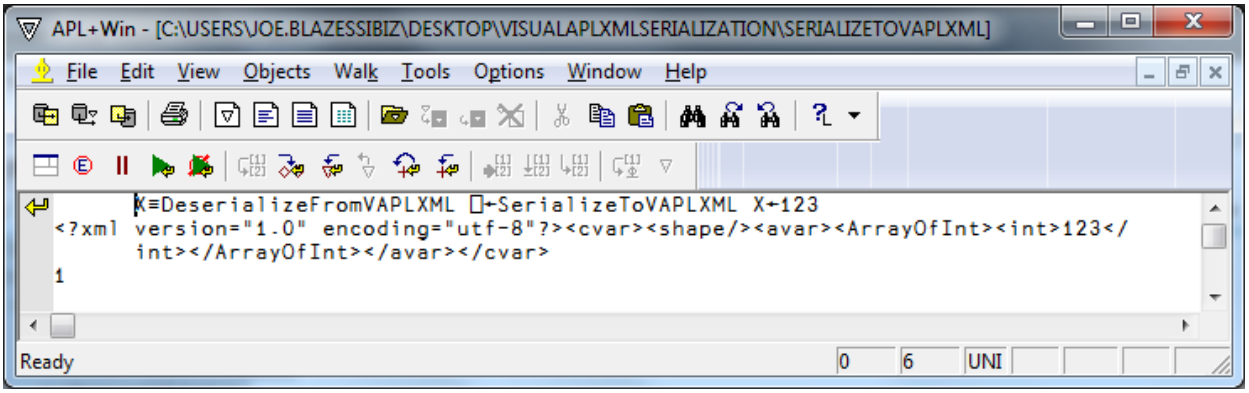


Example #3: Enclosed scalar character

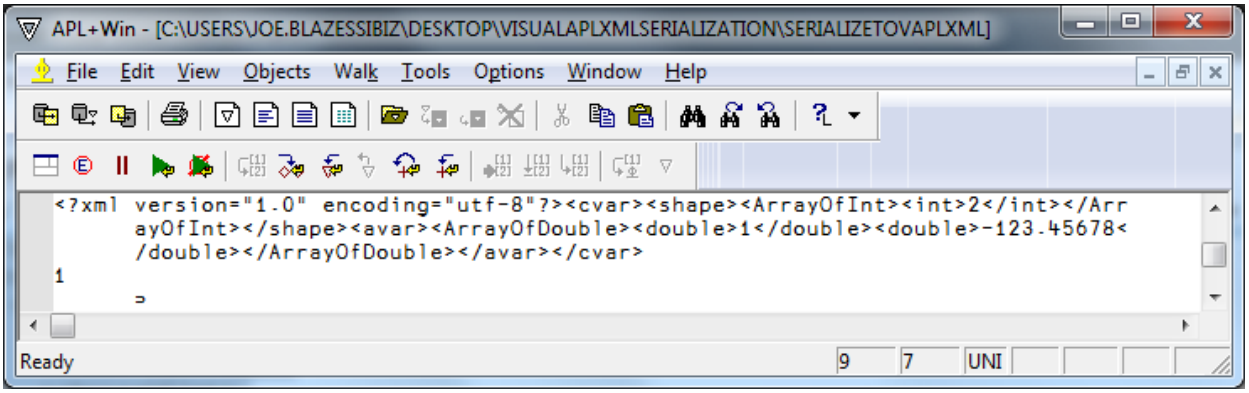
This produces the same serialized result as scalar character



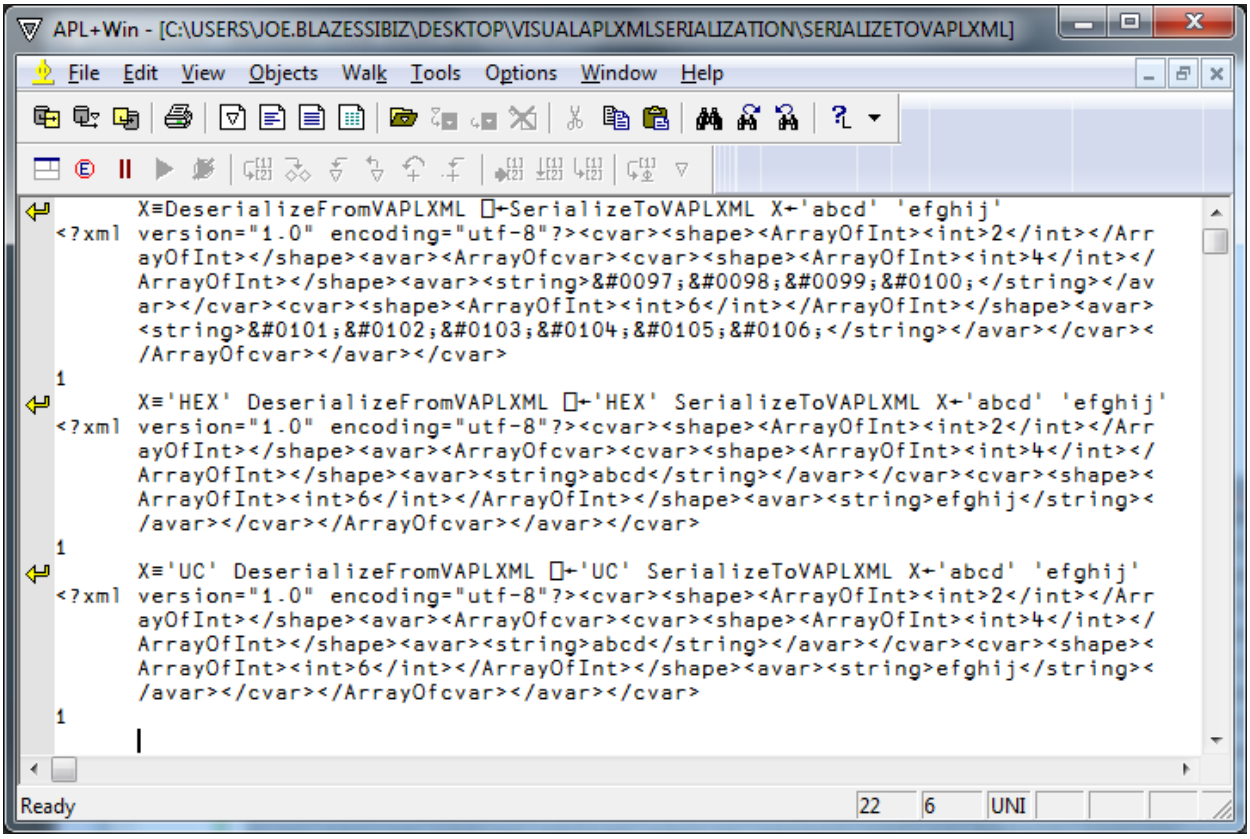
Example #4: Scalar integer



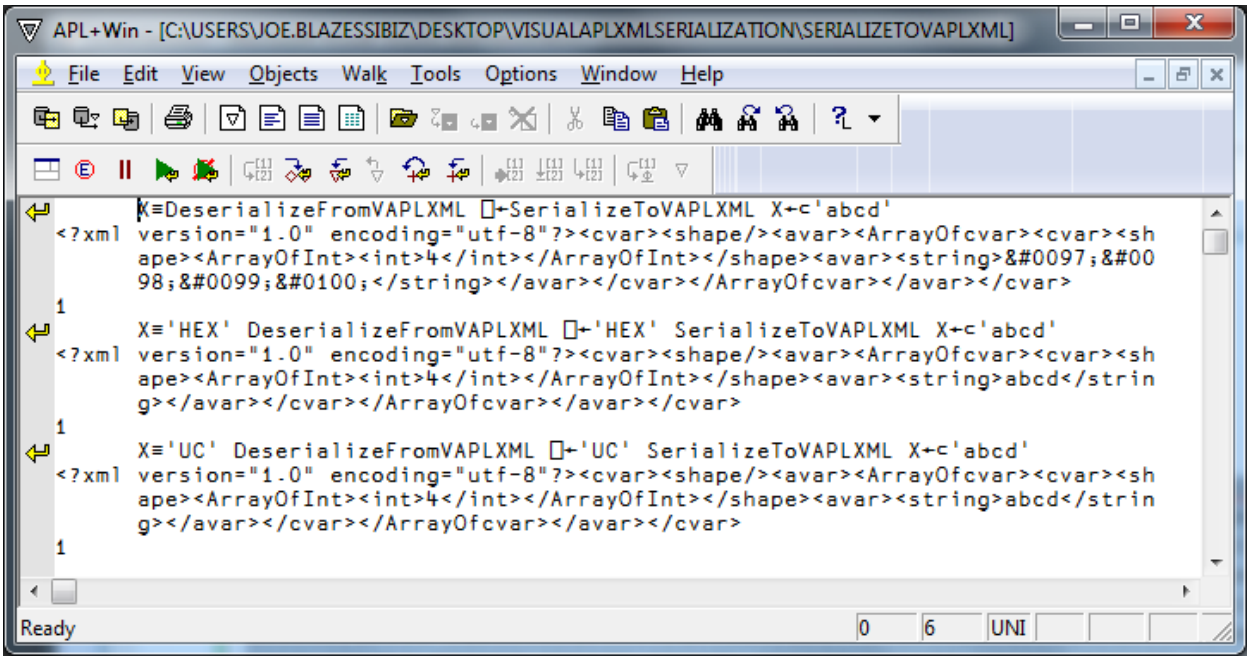
Example #5: Vector of doubles



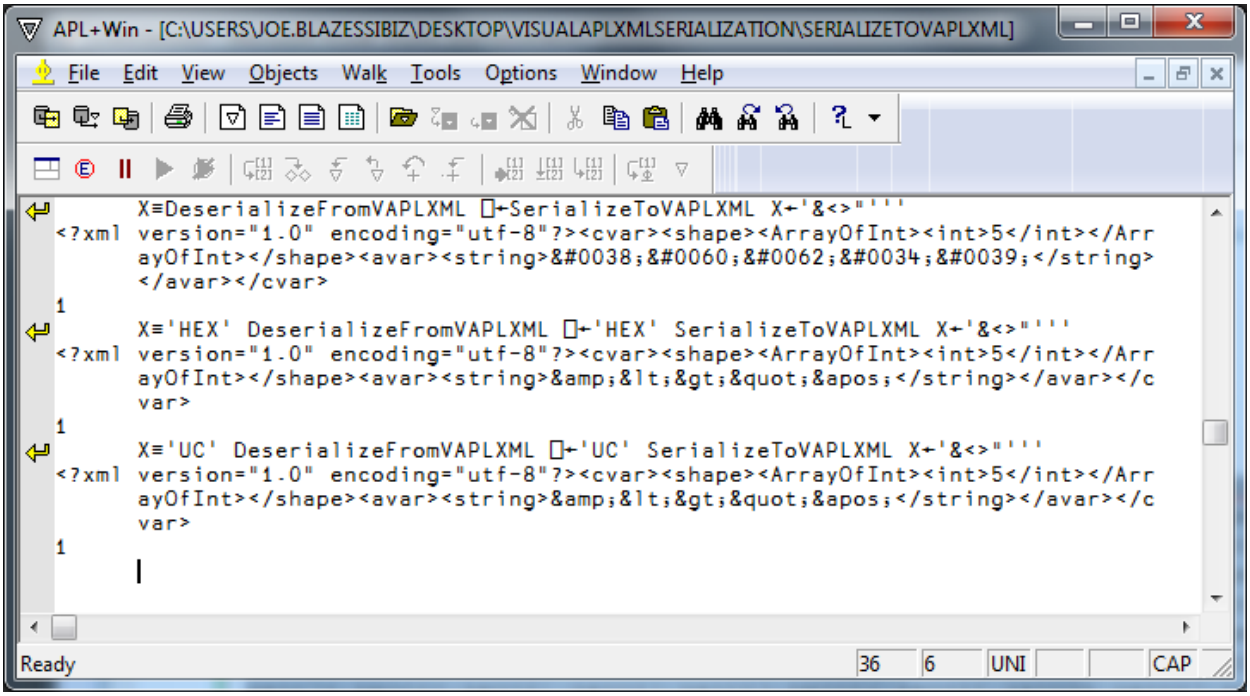
Example #6: Vector of string vectors



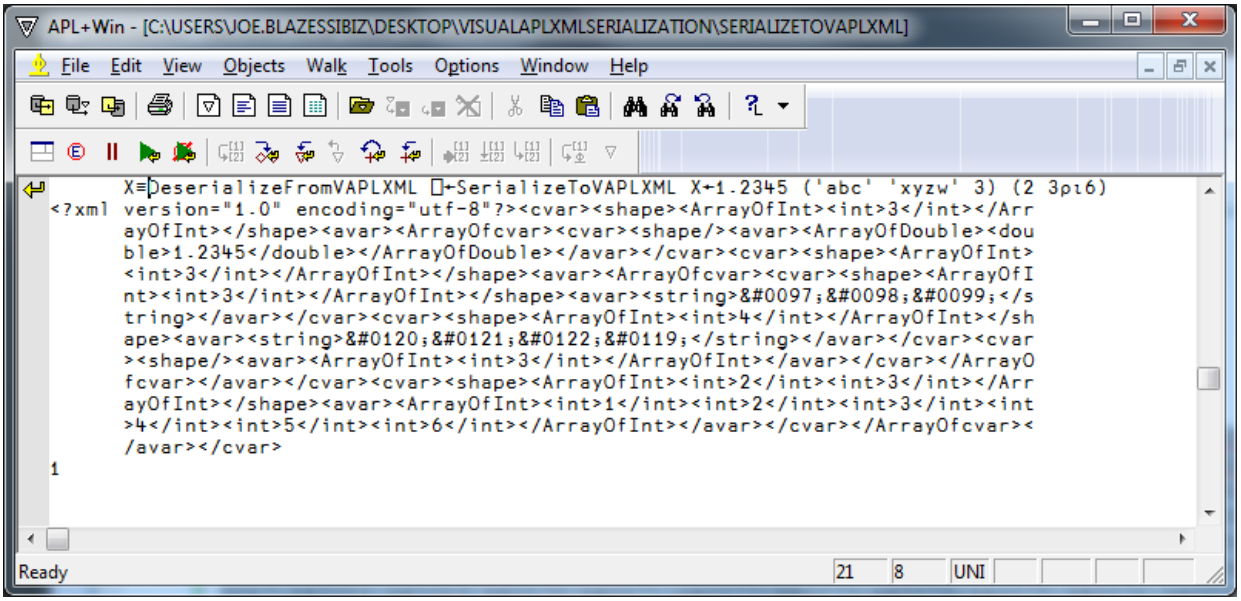
Example #7: Enclosed string vector



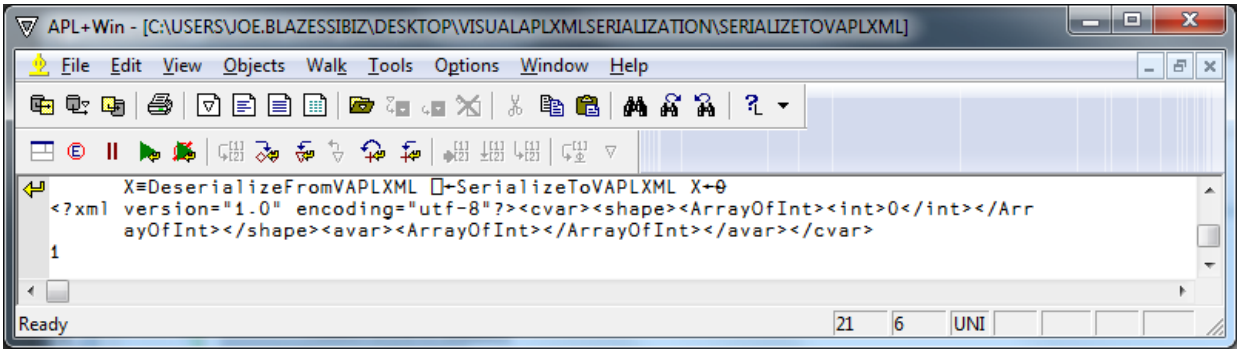
Example #8: Character vector containing XML 'special' characters



Example #9: Nested array



Example #10: Zilde



Example #11: Empty string