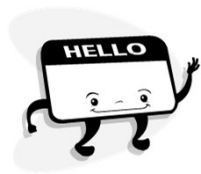


Looking Inside the Developer's Toolkit:

Introduction to Processing XML with RPG and SQL Too!



Charles Guarino

Central Park Data Systems, Inc.
@charlieguarino

About The Speaker

With an IT career spanning over 30 years, Charles Guarino has been a consultant for most of them. Since 1995 he has been founder and President of Central Park Data Systems, Inc., a New York area based IBM midrange consulting and corporate training company. In addition to being a professional speaker across the United States and Europe, he is a frequent contributor of technical and strategic articles and webcasts for the IT community. He is a member of COMMON's Speaker Excellence Hall of Fame and also Long Island Software and Technology Network's Twenty Top Techies. In 2015 Charles became the recipient of the Al Barsa Memorial Scholarship Award. Additionally, he serves as a member of COMMON's Strategic Education Team (SET) and is also a past president and monthly Q&A host of LISUG, a Long Island IBM i User's Group www.lisug.org.

Charles can be reached at cguarino@centralparkdata.com.

LinkedIn - <http://www.linkedin.com/in/guarinocharles>

Twitter - @charlieguarino

What We'll Cover

- **Essential Definitions**
- **XML – A First Look**
- **A Review of Parsers**
- **XML-INTO: A Closer Look**
- **XML-SAX: A Closer Look**
- **XML and SQL**
- **Wrap-up**

ESSENTIAL DEFINITIONS

- 1) What is XML and why use it?
- 2) Element
- 3) Attribute
- 4) Parsers – 2 types!
- 5) Well formed document
- 6) Where is XML typically stored on the IBM i?

What We'll Cover

- **Essential Definitions**
- **XML – A First Look**
- **A Review of Parsers**
- **XML-INTO: A Closer Look**
- **XML-SAX: A Closer Look**
- **XML and SQL**
- **Wrap-up**

If you are already familiar with HTML...

The screenshot shows the w3schools.com website. At the top, there's a navigation bar with links for HOME, HTML, CSS, JAVASCRIPT, JQUERY, XML, ASP.NET, PHP, SQL, and MORE... There's also a search bar and a language selector. The main content area is titled "HTML Tutorial - (HTML5 Compliant)". It features a sidebar on the left with a list of HTML topics, including HTML HOME, HTML Introduction, HTML Editors, HTML Basic, HTML Elements, HTML Attributes, HTML Headings, HTML Paragraphs, HTML Formatting, HTML Links, HTML Head, HTML CSS, HTML Images, HTML Tables, HTML Lists, HTML Blocks, HTML Layout, HTML Forms, HTML Iframes, HTML Colors, HTML Colnames, HTML Colorvalues, HTML JavaScript, HTML Entities, HTML URL Encode, HTML Quick List, HTML Summary, and HTML XHTML. The main content area has a heading "HTML Tutorial - (HTML5 Compliant)" and a sub-heading "« W3Schools Home". It includes a paragraph: "With HTML you can create your own Web site. This tutorial teaches you everything about HTML. HTML is easy to learn - You will enjoy it." Below this, there's a section titled "Examples in Each Chapter" which states: "This HTML tutorial contains hundreds of HTML examples. With our online HTML editor, you can edit the HTML, and click on a button to view the result." An example of HTML code is shown in a text area:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

 Below the code area is a button labeled "Try it yourself »". At the bottom, there's a note: "Click on the 'Try it yourself' button to see how it works".

Then XML will be a snap!

The screenshot shows the w3schools.com website with the XML Tutorial page. The page has a navigation bar at the top with links to HOME, HTML, CSS, XML, JAVASCRIPT, ASP, PHP, SQL, and MORE... There is a search bar and a TRANSLATE button. The main content area is titled "XML Tutorial" and includes a "W3Schools Home" link and a "Next Chapter" link. The tutorial text explains that XML stands for eXtensible Markup Language, is designed to transport and store data, and is important to know and very easy to learn. It includes a "Start learning XML now!" link. Below the text is an "XML Document Example" showing a sample XML document with a root element <?xml version="1.0"?>, a <note> element containing a <to>Tove</to> element, a <from>Jani</from> element, a <heading>Reminder</heading> element, and a <body>Don't forget me this weekend!</body> element. There are also links for "XML Examples" and "XML Quiz Test". On the left side, there is a sidebar with links to various XML topics, including XML Basic, XML JavaScript, XML Advanced, and XML Examples. On the right side, there is a sidebar with links to various web hosting and building services, including WEB HOSTING, WEB BUILDING, W3SCHOOLS EXAMS, W3SCHOOLS BOOKS, and STATISTICS.

Document "citydata1.xml"

The screenshot shows a file explorer window with a sidebar on the left and a main pane on the right. The sidebar shows a tree view of the file system, including "IFS Files", "File systems", "Root file system", "Home", and "xml docs". Under "xml docs", there are three files: "citydata.xml", "citydata1.xml", and "citydata2.xml". The "citydata1.xml" file is selected. The main pane shows the contents of "citydata1.xml" in a text editor. The XML document is structured as follows: <?xml version="1.0"?> <Cities> <CityData> <CityName>New York</CityName> <Region>Northeast</Region> <MonthlyData> <Month>May</Month> <Low>53</Low> <High>71</High> </MonthlyData> <MonthlyData> <Month>June</Month> <Low>63</Low> <High>81</High> </MonthlyData> <MonthlyData> <Month>July</Month> <Low>68</Low> <High>81</High> </MonthlyData> </CityData> <CityData> <CityName>Chicago</CityName> <Region>Midwest</Region> <MonthlyData> <Month>May</Month> <Low>51</Low> <High>70</High> </MonthlyData> <MonthlyData> <Month>June</Month> <Low>61</Low> <High>80</High> </MonthlyData> <MonthlyData> <Month>July</Month> <Low>66</Low> <High>84</High> </MonthlyData> </CityData> </Cities>

What We'll Cover

- **Essential Definitions**
- **XML – A First Look**
- **A Review of Parsers**
- **XML-INTO: A Closer Look**
- **XML-SAX: A Closer Look**
- **XML and SQL**
- **Wrap-up**

Which RPG parser to use?

Examine the XML document.

Does it have a consistent structure, with repetitive groups or many different structures, and/or complex structures?

Will you know in advance how the XML is actually formatted?

How will you be using the data?
Is it a particularly large document?

These important answers will direct you to which parser to use.

DOM = Document Object Model

Allows an application to read and update XML data directly in memory. In the DOM implementation, data is moved into arrays and data structures.

Programmers need to be sensitive to how large an XML document is so as to not exceed the available system storage. Optionally, a 'handler' can be used to deal with array overflow.

RPG implementation  "XML-INTO"

SAX = Simple API for XML

The SAX parser walks (runs?) through an entire XML document, one element at a time, and returns control to the handler as each new 'event' is encountered. The complexity of the document is not relevant.

Also, since system memory is not consumed, the document size is not relevant.

RPG implementation  "XML-SAX"

Where can you find sample XML? There are lots of documents to be downloaded. Take this fairly simple one.

<http://www.w3schools.com/xml/simple.xml>

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Edited by XMLSpy@ -->
- <breakfast_menu>
- <food>
  <name>Belgian Waffles</name>
  <price>$5.95</price>
  <description>two of our famous Belgian Waffles with plenty of real maple syrup</description>
  <calories>650</calories>
</food>
- <food>
  <name>Strawberry Belgian Waffles</name>
  <price>$7.95</price>
  <description>light Belgian waffles covered with strawberries and whipped cream</description>
  <calories>900</calories>
</food>
- <food>
  <name>Berry-Berry Belgian Waffles</name>
  <price>$8.95</price>
  <description>light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
  <calories>900</calories>
</food>
- <food>
  <name>French Toast</name>
  <price>$4.50</price>
  <description>thick slices made from our homemade sourdough bread</description>
  <calories>600</calories>
</food>
- <food>
  <name>Homestyle Breakfast</name>
  <price>$6.95</price>
  <description>two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
  <calories>950</calories>
</food>
</breakfast_menu>
```

This complex XML document was computer-generated, 75 pages long.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE XMLFileIn (View Source for full doctype...)>
<!-- XML ORDER EXPORT FILE V1.0 -->
- <XMLFileIn>
- <OrderV2>
  <Attribute Name="Dealer">10912</Attribute>
  <Attribute Name="Homeowner">000002</Attribute>
  <Attribute Name="ProjectName">New Construction 529</Attribute>
  <Attribute Name="CreatedBy">11714</Attribute>
  <Attribute Name="CustPONumber">Verbal</Attribute>
  <Attribute Name="JobNumber" />
  <Attribute Name="PriceMultiplier">.83</Attribute>
  <Attribute Name="SalesTaxPercent">8.65</Attribute>
  <Attribute Name="SalesTaxAmount">0.00</Attribute>
- <CustomerInfo>
  <Attribute Name="Name" />
  <Attribute Name="CustomerID" />
- <Billing>
  <Attribute Name="Contact" />
  <Attribute Name="Address1" />
  <Attribute Name="Address2" />
  <Attribute Name="City" />
  <Attribute Name="State" />
  <Attribute Name="ZipCode" />
  <Attribute Name="Phone" />
</Billing>
- <Shipping>
  <Attribute Name="Contact" />
  <Attribute Name="Address1" />
  <Attribute Name="Address2" />
```

This document contains over 150 different complex structures. SAX is clearly the way to go!

What We'll Cover

- **Essential Definitions**
- **XML – A First Look**
- **A Review of Parsers**
- **XML-INTO: A Closer Look**
- **XML-SAX: A Closer Look**
- **XML and SQL**
- **Wrap-up**

Introducing...

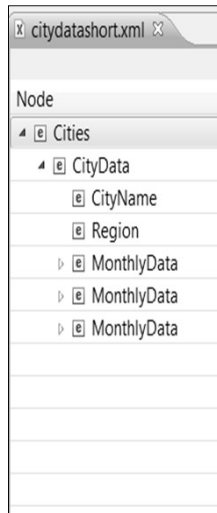


XML-INTO !!!

You need to know the structure format in advance

- Data is mapped into data structures
- Can parse an entire document at one time
- Can use an optional handler for every large documents
- Recommended for less complex structured documents

When using XML-INTO, an XML editor can help you define your data structures



```
dcl-ds cities qualified;
  CityData   likes(CityDataDS) dim(2);
end-ds;

dcl-ds CityDataDS qualified;
  CityName   char(20);
  Region     char(20);
  State      char(20);
  MonthlyData likes(MonthlyDataDS) dim(3);
end-ds;

dcl-ds MonthlyDataDS;
  Month      char(9);
  Low        char(3);
  High       char(3);
end-ds;
```

It's also helpful to use the "tag", "data", "tag" approach

```
C:\cgonly\CPDS workspace\RemoteSy...
<Cities>
- <CityData>
  <CityName>New York</CityName>
  <Region>Northeast</Region>
  - <MonthlyData>
    <Month>May</Month>
    <Low>53</Low>
    <High>71</High>
    </MonthlyData>
  - <MonthlyData>
    <Month>June</Month>
    <Low>63</Low>
    <High>81</High>
    </MonthlyData>
  - <MonthlyData>
    <Month>July</Month>
    <Low>68</Low>
    <High>81</High>
    </MonthlyData>
  </CityData>
</Cities>
```

```
dcl-ds cities qualified;
  CityData   likes(CityDataDS) dim(2);
end-ds;

dcl-ds CityDataDS qualified;
  CityName   char(20);
  Region     char(20);
  State      char(20);
  MonthlyData likes(MonthlyDataDS) dim(3);
end-ds;

dcl-ds MonthlyDataDS;
  Month      char(9);
  Low        char(3);
  High       char(3);
end-ds;
```

Our first XML parsing program “CITYDATA1G”

```

ctl-opt option(*nodebugio);

dcl-f citydata1 disk(*ext) usage(*output) rename(citydata1:citydata1);

dcl-ds cities qualified;
  CityData likes(CityDataDS) dim(2);
end-ds;

dcl-ds CityDataDS qualified;
  CityName char(20);
  Region char(20);
  State char(20);
  MonthlyData likes(MonthlyDataDS) dim(3);
end-ds;

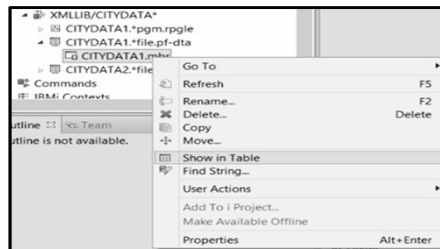
dcl-ds MonthlyDataDS;
  Month char(9);
  Low char(3);
  High char(3);
end-ds;

dcl-s options1 char(100);
dcl-s filename varchar(25) inz('/xml/docs/citydata1.xml');
dcl-s x packed(2:0);
dcl-s y packed(2:0);

// Setup parsing options
options1 = 'doc=file +
allowextra=yes allowmissing=yes case=any';

```

“Show in Table” view of parsed data as it resides in physical file CITYDATA1



*RCNBR	CITYNAME	REGION	MONTHNAME	LOW	HIGH
1	New York	Northeast	May	53	71
2	New York	Northeast	June	63	81
3	New York	Northeast	July	68	81
4	Chicago	Midwest	May	51	70
5	Chicago	Midwest	June	61	80
6	Chicago	Midwest	July	66	84

XML parsing program CITYDATA2G using a handler

```
ctl-opt dftactgrp(*no)      option(*nodebugio);

// Note - Since this is using free form this is for use on a 7.1 system
// normally when calling a local subprocedure the prototype is optional

dcl-pr Xmlhandler    int(10);
      MyCommArea      char(100);
      CityData        likeds(citydata) dim(3) const;
      CitiesParsed    int(10) value;

dcl-ds Cities qualified;
      CityData        likeds(CityData) dim(3);
end-ds;

dcl-ds CityData qualified;
      CityName        char(20);
      Region          char(20);
      MonthlyData      likeds(MonthlyDataDS) dim(3);
end-ds;

dcl-ds MonthlyDataDS;
      Month           char(9);
      Low             char(3);
      High            char(3);

dcl-s options1      char(100);
dcl-s filename      varchar(25)      inz('/xml/docs/citydata2.xml');
```

Variations on a theme – not originally parse-able with RPG

A more typical XML example, where the element “state” has associated attributes placed before the actual text data. The text data found here is the actual state name.

```
<states>
  <state abbreviation="AL" capital="Montgomery"> ALABAMA </state>
  <state abbreviation="AK" capital="Juneau"> ALASKA </state>
  <state abbreviation="AZ" capital="Phoenix"> ARIZONA </state>
  <state abbreviation="AR" capital="Little Rock"> ARKANSAS </state>
  <state abbreviation="CA" capital="Sacramento"> CALIFORNIA </state>
  <state abbreviation="CO" capital="Denver"> COLORADO </state>
  <state abbreviation="CT" capital="Hartford"> CONNECTICUT </state>
  <state abbreviation="DE" capital="Dover"> DELAWARE </state>
</states>
```

Data sub fields

```

STATES.RPGLE
Line 1      Column 1      Replace
.....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8..
000700
002805      D states      ds      qualified
002806      D state      ds      likeds(states) dim(99)
002807
002809      D stateds      ds
002813      D abbreviation  2
002814      D capital      20
002815      D statename    30
003400
005400
005500      D options1      s      100a
005601      D filename      S      25a      inz('xmldocs/states.xml')varying
005700
005800
005900      /free
006000
006001      options1 = 'doc=file +
006002      allowextra=yes allowmissing=yes case=any datasubf=statename';
009700
009800      // Capture header data
009900      xml-into states %xml(filename:options1);
010000
010001      *inlr = *on;
012700
013900

```

Sample XML with namespaces – how do you deal with colons?

```

<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>

```

Source: http://www.w3schools.com/xml/xml_namespaces.asp

RPG can *still* parse this with namespace support

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

```
dcl-ds TableDS      qualified;
  Name              char(20);
  Width             char(3);
  Length            char(4);
end-ds TableDS;
```

ns = remove

```
dcl-ds TableDS      qualified;
  f_Name            char(20);
  f_Width           char(3);
  f_Length          char(4);
end-ds TableDS;
```

ns = merge

Summary of XML-INTO %xml Options

Original V5R4 options

doc – XML file exists in a document (versus a datastring)
allowextra – don't stop with error if undeclared tags are found
allowmissing – don't stop with error if expected data is missing
path – specifies starting element where to start parsing XML document
case - (any, lower, upper) – specifies the case of the data elements to map
ccsid – (best, job, ucs2) – specifies which CCSID to use
trim – specified whether to include whitespace mapped into your variables

V6R1 PTFs, 7.1 and beyond

Datasubf – names the DS subfield to capture the text data
Countprefix – specifies the prefix for selected array fields where you need to capture the number of elements return. Can replace allowmissing=yes.
case (convert using *LANGIDSHR table, convert alphabetic characters)

ns (remove=only use matching subfield, merge=join namespace with subfield)
nsprefix declares prefix of new field to capture actual namespace

What We'll Cover

- **Essential Definitions**
- **XML – A First Look**
- **A Review of Parsers**
- **XML-INTO: A Closer Look**
- **XML-SAX: A Closer Look**
- **XML and SQL**
- **Wrap-up**

Introducing...



XML-SAX !!!

- You DO NOT need to know the structure format in advance.
- Data is mapped anywhere you want at runtime.
- Reads the document the same way we do, left to right from top to bottom.
- Each new piece of encountered data is an event, which is passed to an event handler.
- Recommended for more complex structured documents.

This document contains both XML elements and attributes, and the data is inconsistent. It is however well-formed.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE XMLFileIn (View Source for full doctype...)>
<!-- XML ORDER EXPORT FILE V1.0 -->
<XMLFileIn>
  <OrderV2>
    <Attribute Name="Dealer">10912</Attribute>
    <Attribute Name="Homeowner">000002</Attribute>
    <Attribute Name="ProjectName">New Construction 529</Attribute>
    <Attribute Name="CreatedBy">11714</Attribute>
    <Attribute Name="CustPONumber">Verbal</Attribute>
    <Attribute Name="JobNumber" />
    <Attribute Name="PriceMultiplier">.83</Attribute>
    <Attribute Name="SalesTaxPercent">8.65</Attribute>
    <Attribute Name="SalesTaxAmount">0.00</Attribute>
  <CustomerInfo>
    <Attribute Name="Name" />
    <Attribute Name="CustomerID" />
  <Billing>
    <Attribute Name="Contact" />
    <Attribute Name="Address1" />
    <Attribute Name="Address2" />
    <Attribute Name="City" />
    <Attribute Name="State" />
    <Attribute Name="ZipCode" />
    <Attribute Name="Phone" />
  </Billing>
  <Shipping>
    <Attribute Name="Contact" />
    <Attribute Name="Address1" />
    <Attribute Name="Address2" />
  </Shipping>
</OrderV2>
</XMLFileIn>
```

The sax parser reads the same way you read a book; one word at a time, left to right, top to bottom.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE XMLFileIn (View Source for full doctype...)>
<!-- XML ORDER EXPORT FILE V1.0 -->
<XMLFileIn>
<OrderV2>
<Attribute Name="Dealer">10912</Attribute>
<Attribute Name="Homeowner">000002</Attribute>
```

20	=	Start_Document	
25	=	Version_InfoVersion	1.0
10	=	Encoding_Decl	UTF-8
9	=	Doctype_Decl	XMLFileIn
6	=	Comment	XML Order Export File V1.0
21	=	Start_Element	XMLFileIn
5	=	Chars	*blank
21	=	Start_Element	OrderV2
5	=	Chars	*blank
21	=	Start_Element	Attribute
2	=	Attr_Name	Name
4	=	Attr=Chars	Dealer
26	=	End_Attr	Name
5	=	Chars	10912
13	=	End_Element	Attribute

Tables SAXCTL and SAXDATA

SAXDATA is populated in program SAXPARSES and processed sequentially in the P.O. creation program.

SAXCTL

PROCESSED_PATH	CHAR(20)
PROCESSED_DOC	CHAR(20)
PROCESSED_FLAG	CHAR(1)
PROCESSED_DATTIM	TIMESTAMP

SAXDATA

XMLDATATYPE	CHAR(20)
XMLDATA	CHAR(256)
XMLDOCPATH	CHAR(20)
XMLDOCNAME	CHAR(20)

We already know which pieces of data we want to capture. Each time the event handler is called, if the event code identifies one of these known values, we capture it. Otherwise we simply ignore it.

Program Listing SAXPARSES

```
ctl-opt dftactgrp(*no) debug(*xmlsax) option(*nodebugio);

// To compile this program
// CRTSQLRPGI OBJ(XMLLIB/SAXPARSES)
// SRCFILE(XMLLIB/QRPGLESRC)
// COMMIT(*NONE) DBGVIEW(*SOURCE)

dcl-c num_elements const(500);
dcl-s parsingstatus char(1);
dcl-s parseddata char(256);
dcl-s xmldoc char(50);
dcl-s options char(20) inz('doc=file');
dcl-s currentdate date inz(*sys);

dcl-ds saxctlds extname('SAXCTL') qualified end-ds;

// Communications area definition
dcl-ds MyCommArea;
  namecount int(10);
  elementdata dim(num_elements);
  name char(64) overlay(elementdata);
  startcount int(10) overlay(elementdata:*next);
  endcount int(10) overlay(elementdata:*next);
end-ds;

dcl-pr EventHandler int(10);
  commArea likes(MyCommArea);
  event int(10) value;
  pstring pointer value;
  stringlen int(20) value;
  exceptionID int(10) value;
```


*XMLSAX debugging option

```
ctl-opt dftactgrp(*no) debug(*xmlsax) option(*nodebugio);
```

- The ***XMLSAX debug option** generates a special array named **_QRNU_XMLSAX** into your program. Since the **SAX** parser returns just a numeric value, it is helpful to identify what each value represents.
- In the event handler we can test for each event and take the appropriate action. We will capture certain pieces of data, based on the event code that is passed.

All possible sax event codes in data structure _QRNU_XMLSAX

```

  • _QRNU_XMLSAX
    • _QRNU_XMLSAX(1) = ATTR_UCS2_REF
    • _QRNU_XMLSAX(2) = ATTR_NAME
    • _QRNU_XMLSAX(3) = ATTR_PREDEF_REF
    • _QRNU_XMLSAX(4) = ATTR_CHARS
    • _QRNU_XMLSAX(5) = CHARS
    • _QRNU_XMLSAX(6) = COMMENT
    • _QRNU_XMLSAX(7) = UCS2_REF
    • _QRNU_XMLSAX(8) = PREDEF_REF
    • _QRNU_XMLSAX(9) = DOCTYPE_DECL
    • _QRNU_XMLSAX(10) = ENCODING_DECL
    • _QRNU_XMLSAX(11) = END_CDATA
    • _QRNU_XMLSAX(12) = END_DOCUMENT
    • _QRNU_XMLSAX(13) = END_ELEMENT
    • _QRNU_XMLSAX(14) = END_PREFIX_MAPPING
    • _QRNU_XMLSAX(15) = EXCEPTION
    • _QRNU_XMLSAX(16) = PI_TARGET
    • _QRNU_XMLSAX(17) = PI_DATA
    • _QRNU_XMLSAX(18) = STANDALONE_DECL
    • _QRNU_XMLSAX(19) = START_CDATA
    • _QRNU_XMLSAX(20) = START_DOCUMENT
    • _QRNU_XMLSAX(21) = START_ELEMENT
    • _QRNU_XMLSAX(22) = START_PREFIX_MAPPING
    • _QRNU_XMLSAX(23) = UNKNOWN_ATTR_REF
    • _QRNU_XMLSAX(24) = UNKNOWN_REF
    • _QRNU_XMLSAX(25) = VERSION_INFO
    • _QRNU_XMLSAX(26) = END_ATTR
```

This is how the parsed data looks in file SAXDATA.
A second program will read these records sequentially and process the data.

Event Codes	Start_Element	XMLFileIn
21	Start_Element	OrderV2
2	Start_Element	Attribute
4	Attr_Name	Name
5	Attr_Chars	Dealer
13	XML_Chars	10912
	End_Element	Attribute
	Start_Element	Attribute
	Attr_Name	Name
	Attr_Chars	Homeowner
	XML_Chars	000002
	End_Element	Attribute
	Start_Element	Attribute
	Attr_Name	Name
	Attr_Chars	ProjectName
	Start_Element	FinishedGood
	Start_Element	PartNumber
	XML_Chars	1631516
	End_Element	PartNumber
	Start_Element	Attribute
	Attr_Name	Name
	Attr_Chars	Order_Quantity
	XML_Chars	4
	End_Element	Attribute
	Start_Element	Attribute
	Attr_Name	Name
	Attr_Chars	MfgNumber
	XML_Chars	HG-5522GH
	End_Element	Attribute
	Start_Element	Attribute

Code snippet of how to process this captured sequential data

```

If datatype = 'Attr_Chars';
Select;
When data = 'Dealer';
    flag = 'Dealer';
...

If datatype = 'XML_Chars';
Select;
When flag = 'Dealer';
    dealerno = %trim(data);
When flag = 'Homeowner';
    homeownerdata = %trim(data);
...

```

➤ Sets flag

➤ Map suitable data

Parsed data shown in SAXDATA using Table View

192.168.2.21:XMLLIB/SAXDATA(SAXDATA)					
DATATYPE	DATA		DOCP...	DOC...	
Start_Element	XMLFileIn	...	xmldocs	...	sample2...
Start_Element	OrderV2	...	xmldocs	...	sample2...
Start_Element	Attribute	...	xmldocs	...	sample2...
Attr_Name	Name	...	xmldocs	...	sample2...
Attr_Chars	Dealer	...	xmldocs	...	sample2...
XML_Chars	10912	...	xmldocs	...	sample2...
End_Element	Attribute	...	xmldocs	...	sample2...
Start_Element	Attribute	...	xmldocs	...	sample2...
Attr_Name	Name	...	xmldocs	...	sample2...
Attr_Chars	Homeowner	...	xmldocs	...	sample2...
XML_Chars	000002	...	xmldocs	...	sample2...
End_Element	Attribute	...	xmldocs	...	sample2...
Start_Element	Attribute	...	xmldocs	...	sample2...
Attr_Name	Name	...	xmldocs	...	sample2...
Attr_Chars	ProjectName	...	xmldocs	...	sample2...
XML_Chars	New Construction 529	...	xmldocs	...	sample2...
End_Element	Attribute	...	xmldocs	...	sample2...
Start_Element	Attribute	...	xmldocs	...	sample2...
Attr_Name	Name	...	xmldocs	...	sample2...
Attr_Chars	CreatedBy	...	xmldocs	...	sample2...
XML_Chars	11714	...	xmldocs	...	sample2...
End_Element	Attribute	...	xmldocs	...	sample2...
Start_Element	Attribute	...	xmldocs	...	sample2...
Attr_Name	Name	...	xmldocs	...	sample2...
Attr_Chars	CustPONumber	...	xmldocs	...	sample2...
XML_Chars	Verbal	...	xmldocs	...	sample2...
End_Element	Attribute	...	xmldocs	...	sample2...
Start_Element	Attribute	...	xmldocs	...	sample2...
Attr_Name	Name	...	xmldocs	...	sample2...

What We'll Cover

- **Essential Definitions**
- **XML – A First Look**
- **A Review of Parsers**
- **XML-INTO: A Closer Look**
- **XML-SAX: A Closer Look**
- **XML and SQL**
- **Wrap-up**

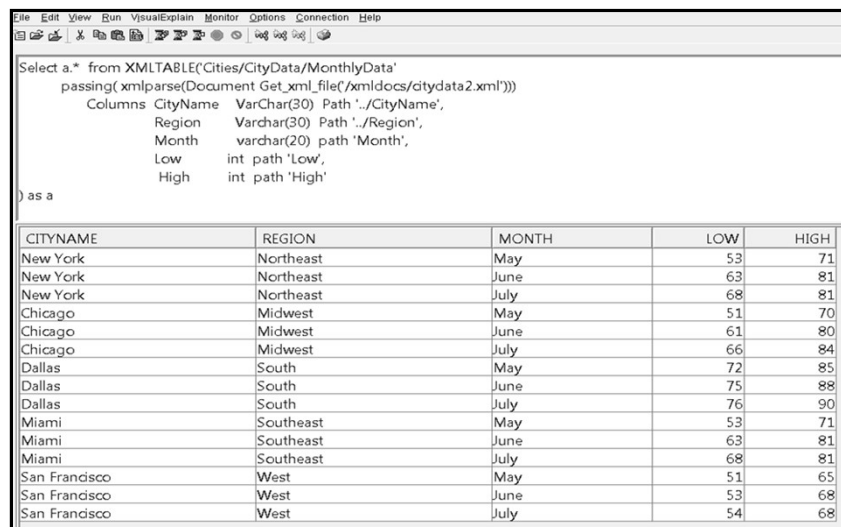
New SQL functionality makes parsing XML a snap

```

Select a.* from XMLTABLE('Cities/CityData/MonthlyData'
    passing(xmlparse(Document
        Get_xml_file(l/xmldocs/citydata2.xml)))
    Columns CityName  VarChar(30) Path '../CityName',
           Region    Varchar(30) Path '../Region',
           Month     varchar(20) path 'Month',
           Low       int path 'Low',
           High      int path 'High'
    ) as a;

```

Running SQL Script in Navigator **



```

Select a.* from XMLTABLE('Cities/CityData/MonthlyData'
    passing(xmlparse(Document Get_xml_file('/xmldocs/citydata2.xml')))
    Columns CityName  VarChar(30) Path '../CityName',
           Region    Varchar(30) Path '../Region',
           Month     varchar(20) path 'Month',
           Low       int path 'Low',
           High      int path 'High'
    ) as a

```

CITYNAME	REGION	MONTH	LOW	HIGH
New York	Northeast	May	53	71
New York	Northeast	June	63	81
New York	Northeast	July	68	81
Chicago	Midwest	May	51	70
Chicago	Midwest	June	61	80
Chicago	Midwest	July	66	84
Dallas	South	May	72	85
Dallas	South	June	75	88
Dallas	South	July	76	90
Miami	Southeast	May	53	71
Miami	Southeast	June	63	81
Miami	Southeast	July	68	81
San Francisco	West	May	51	65
San Francisco	West	June	53	68
San Francisco	West	July	54	68

****To run Connection > Use Temporary JDBC settings
 Set Isolation Level to Cursor Stability (*CS)**

Add a SQL INSERT and you've got a working program!

```
ctl-opt option(*nodebugio) dftactgrp(*no);

// To compile use this command:
// CRTSQLRPGI OBJ(XMLLIB/CITYDT2SQL) SRCFILE(XMLLIB/QRPGLESRC) DBGVIEW(*SOURCE)
// Table CITYDATA2 must be journaled to execute this program

// XMLTABLE - defines the stream file and node to start
// XMLPARSE - checks validity of XML data and parses XML
// Get_XML_file - points to actual stream file
// Columns - specifies which columns to return

exec sql
insert into xmllib/citydata2 (cityname, region, monthname, low, high)
Select a.* from XMLTABLE('Cities/CityData/MonthlyData'
passing( xmlparse(Document Get_xml_file('/xmldocs/citydata2.xml')))
Columns CityName VarChar(20) Path '../CityName',
Region Varchar(20) Path '../Region',
Month varchar(9) path 'Month',
Low int path 'Low',
High int path 'High')
as a;

*inlr = *on;
return;
```

For Further Reading...

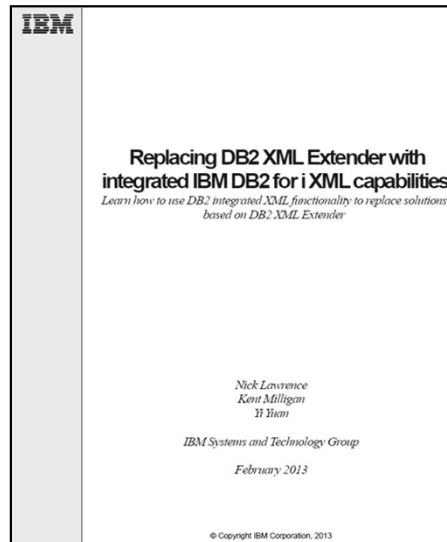
Be sure to visit IBM's RPG Café wiki.

LOTS of good examples of using the native RPG parsers
along with information about other recent RPG
enhancements.

<https://www.ibm.com/developerworks/ibmi/rpg/welcome>

Want more information, directly from IBM?

<https://ibm.biz/XMLandDB2fori>



What We'll Cover

- **Essential Definitions**
- **XML – A First Look**
- **A Review of Parsers**
- **XML-INTO: A Closer Look**
- **XML-SAX: A Closer Look**
- **XML and SQL**
- **Wrap-up**

Wrap-up

- XML-INTO is appropriate for less complex complicated structures.
 - XML parsing was first introduced into RPG at V5R4.
 - It parses the XML data into data structures, therefore you **MUST** know the names of the tags in advance.
 - If you are missing tags – or – define too many tags in your data structure and do not specify “allow-missing” or “allow-extra” you will receive an RPG parsing error at run-time.
 - PTF SI34938, which became available in early 2009 enhanced XML-INTO, allowing it the ability to parse more complex XML structures.
 - Additional PTFs are now available to parse XML documents that contain namespaces – read the PDF!

Wrap-up (continued)

- XML-SAX is appropriate for any type of XML structure.
 - It is an event driven parser, which reads your document the same way you read a regular document – left to right, top to bottom.
 - An event handler is called for each event that is encountered.
 - ▶ Along with each event is an event code.
 - “Allow-missing” and “allow-extra” are meaningless to the SAX parser since you do not pre-define your tags.
 - You do not need to handle or capture the event code every time the handler is called. Only capture what you need.

CITYDATA1 Table

```
CREATE TABLE XMLLIB/CITYDATA1  
  
(CITYNAME CHAR (20 ) NOT NULL WITH DEFAULT,  
REGION CHAR (20 ) NOT NULL WITH DEFAULT,  
MONTHNAME CHAR (9) NOT NULL WITH DEFAULT,  
LOW CHAR (3 ) NOT NULL WITH DEFAULT,  
HIGH CHAR (3 ) NOT NULL WITH DEFAULT)
```

CITYDATA2 Table

```
CREATE TABLE XMLLIB/CITYDATA2  
  
(CITYNAME CHAR (20 ) NOT NULL WITH DEFAULT,  
REGION CHAR (20 ) NOT NULL WITH DEFAULT,  
MONTHNAME CHAR (9) NOT NULL WITH DEFAULT,  
LOW CHAR (3 ) NOT NULL WITH DEFAULT,  
HIGH CHAR (3 ) NOT NULL WITH DEFAULT)
```


SAXCTL Table

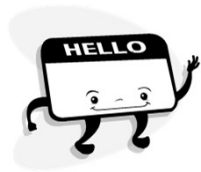
```
CREATE TABLE XMLLIB/SAXCTL  
  
(PROCESSED_PATH FOR COLUMN PRCDOPATH CHAR  
(20 ) NOT NULL WITH DEFAULT,  
  
PROCESSED_DOC FOR COLUMN PRCDOCNAME CHAR  
(20 ) NOT NULL WITH DEFAULT,  
  
PROCESSED_FLAG FOR COLUMN PRCFLAG  
CHAR (1 ) NOT NULL WITH DEFAULT,  
  
PROCESSED_DATTIM FOR COLUMN PRCDATTIM  
TIMESTAMP NOT NULL WITH DEFAULT)
```

SAXDATA Table

```
CREATE TABLE XMLLIB/SAXDATA  
  
(XMLDATATYPE CHAR ( 20) NOT NULL WITH DEFAULT,  
  
XMLDATA CHAR (256 ) NOT NULL WITH DEFAULT,  
  
XMLDOPATH FOR COLUMN DOPATH CHAR (20 ) NOT  
NULL WITH DEFAULT,  
  
XMLDOCNAME FOR COLUMN DOCNAME CHAR (20 ) NOT  
NULL WITH DEFAULT)
```

Looking Inside the Developer's Toolkit:

Introduction to Processing XML with RPG and SQL Too!



Charles Guarino

THANK YOU !!!