
XML vs JSON - Battle Royale
@RileyMajor
1

SQL Community

- PASS
- Twitter
  - #SQLSatMadison
  - #tsql2sday
  - @RileyMajor
- Blogs
  - http://TSQLTuesday.com/
  - http://SQLServerCentral.com/
  - http://LessThanDot.com/
  - http://DBA.StackExchange.com/
  - http://blogs.SentryOne.com/
  - http://scribnasium.com/
- Minnesota
  - PASSMN – Twin Cities
    - @PASSMN
    - http://MNSSUG.org
- Wisconsin
  - FoxPASS - Appleton, WI
  - MADPASS - Madison, WI
  - Microsoft BI Professionals - Wisconsin: Greendale, WI
  - Western Wisconsin PASS - Eau Claire, WI
  - WausauPASS - Wausau, WI
  - WI SSUG - Waukesha, WI

Background

XML

- eXtensible Markup Language
- Introduced in 1998.
- Derived from SGML (parent of HTML) by W3C.
- Human & Machine Readable
- Elements and Attributes
- T-SQL Support in 2000

JSON

- JavaScript Object Notation
- Hints in 1996. More like 2002. RFC 4627 in 2006.
- Formalized by ECMA (makers of JavaScript) in 2013.
- Human & Machine Readable
- Name/Value Pairs.
- T-SQL Support in 2016

XML:

https://www.w3.org/XML/
https://www.w3.org/TR/2008/REC-xml-20081126/
http://docstore.mik.ua/orelly/xml/xmlnut/ch01_04.htm

JSON:

http://www.json.org/
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
http://www.ietf.org/rfc/rfc4627.txt
https://www.youtube.com/watch?v=-C-JoyNuQJs

Originally, the Internet and regular applications were in such dire need of a standard for information exchange that even those who hated XML supported its use. Eventually, the simplicity of JSON and its popularity on the web (due to it being a subset of JavaScript) propelled its adoption.

XML is more complex– supporting elements, attributes, namespaces, validation, control language, comments, and more.

XML was created through a standards body. JSON arose organically out of JavaScript itself. Multiple people had similar ideas but Douglas Crockford is credited with standardizing the concept.

Basic Structure

XML

```
<?xml version="1.0"?>
<element
  attribute="value"
>
  Character Data
</element>
```

JSON

```
{
  "name":"value"
}
```


The XML version information is optional but recommended, so I included it here. I included attributes as they are commonly used, but they are also optional. Namespaces are optional, too, but I didn't include them as it's quite a bit messier due to the length of the URI, and they are less frequently used than the other two optional concepts displayed.

Note that white space handling in XML is not a simple concept. Different implementations treat the white space inside an element (the character data, above) differently. That makes "pretty printing" easier with JSON.

Arrays

XML

```
<array>
  <item>data</item>
  <item>data</item>
</array>
```

JSON

```
{
  "array":
  [
    "data",
    "data"
  ]
}
```


XML does not have a concept of an array, natively. Attributes can’t be used easily as there must be only one attribute with a given name on any element. You could delimit values within an attribute, but there is no prescribed methodology. It would be up to the producing and consuming apps to handle and validate that.. However, XML does permit repeating elements at the same nesting level (except for the root level).

JSON has a native concept of an array. It can be the top-level of a JSON document, but the array itself can’t have a name unless it’s part of an object.

Nesting

XML

```
<Level1>
   <Level2>
      <Level3>
         Data
      </Level3>
   </Level2>
</Level1>
```

JSON

```
{
"Level1":
   {
   "Level2":
      {
      "Level3":
         "Data"
      }
   }
}
```


There’s no nesting limit defined in the specs, but implementations vary in their ability to nest.

Msg 6335, Level 16, State 102, Line 5
XML datatype instance has too many levels of nested nodes. Maximum allowed depth is 128 levels.

Msg 13606, Level 16, State 1, Line 7
JSON text that has more than 128 nesting levels cannot be parsed.

Data Types

XML

- Natively, none.
- With Schemas:
  - String
  - Boolean
  - Decimal
  - dateTime
  - anyURI
  - …more…

JSON

- Strings (quotes)
- Numeric (no quotes; scientific notation supported)
- Boolean (true, false)
- null

XML Schemas:

https://www.w3.org/TR/xmlschema-2/#dt-primitive

Special Characters

XML

- Elements should be letters and numbers, with no spaces. Can use:
  - . - _ :
- In data and attributes, must encode:
  - < as <
  - & as &
- Encode chosen quotes in attributes.
- Control characters (except CR LF TAB) are not allowed.

JSON

- Keys and string data must be quote (") encapsulated.
- Quotes ("), "reverse solidus" aka backslash (\), and control characters (up through code 31, even tabs).
- Encode using backslash and unicode code point or shortcut (\r\n).

XML

Element names must start with a letter, underscore, or colon and must continue with letters, digits, periods, hyphens, underscores, or colons.

Most folks recommend encoding "greater than" and both types of quotes always, but that's not necessary.

https://www.xml.com/pub/a/2001/07/25/namingparts.html

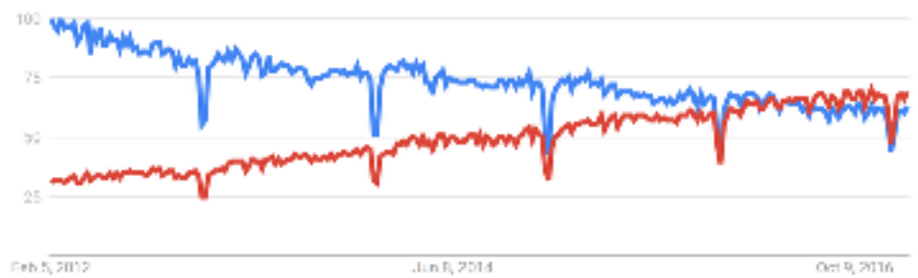
http://stackoverflow.com/questions/1091945/what-characters-do-i-need-to-escape-in-xml-documents

JSON

https://www.ietf.org/rfc/rfc4627.txt

Search Trends

- XML (Blue) vs JSON (Red)

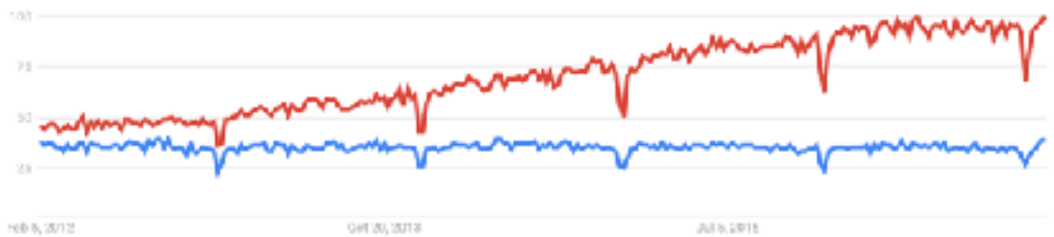


The dips are the year end holidays.

https://trends.google.com/trends/explore?date=2012-02-01%202017-01-31&q=xml,json

Web Ecosystem

- The world wide web loves JSON.
- SOAP (Blue): a complex XML-based API method.
- REST (Red): a simpler API method, usually using JSON.



https://trends.google.com/trends/explore?cat=5&date=2012-02-01%202017-01-31&q=soap,rest

Seriously I've seen people grimace when I mention XML.

Microsoft Ecosystem

XML

- SQL Server Query Plans
- SQL Server Extended Events
- BIML
- SSIS Packages & Configuration
- SSRS Configuration
- SSAS XMLA
- PowerBI Configuration
- Office File Formats
- SQLSaturday.com Data
- XAML

JSON

- TypeScript Configuration
- SSAS Tabular 2016 (TMSL)
- Visual Studio Team Services
- Various REST web services.

SSIS Packages: https://msdn.microsoft.com/en-us/library/hh758694(v=sql.120).aspx
SSRS Configuration: https://docs.microsoft.com/en-us/sql/reporting-services/report-server/reporting-services-configuration-files
SSAS XMLA: https://www.mssqltips.com/sqlservertip/2982/sql-server-2012-analysis-services-xmla/

TMSL: https://docs.microsoft.com/en-us/sql/analysis-services/tabular-model-scripting-language-tmsl-reference
Visual Studio Team Services Extensions: https://www.visualstudio.com/en-us/docs/integrate/extensions/develop/manifest

SQL Server Support

XML

- SQL Server 2000
  - FOR XML
  - OPENXML
- SQL Server 2005
  - XML Data Type
  - XML Indexing
  - XML Schema Processing
  - XML FLWOR Support
  - Functions: query, value, exist, nodes, modify

JSON

- SQL Server 2016
  - FOR JSON
  - OPENJSON
  - Functions: ISJSON, JSON_VALUE, JSON_QUERY, JSON_MODIFY
- Differences
  - No “prepare document” step for OPENJSON
  - No “nodes” function.

http://www.informit.com/articles/article.aspx?p=99813

https://technet.microsoft.com/en-us/library/ms345117%28v=sql.90%29.aspx?f=255&MSPPError=-2147217396

http://www.sqlservercurry.com/2011/05/sql-server-xml-flowr-expression-in-sql.html

XML vs JSON - Sample Data

```
DECLARE @Orders TABLE
(
        OrderID bigint IDENTITY,
        OrderDate datetime
);
```

```
DECLARE @OrderDetails TABLE
(
        OrderDetailsID bigint IDENTITY,
        OrderID bigint,
        ProductID varchar(50),
        Qty int
);
```

OrderID	OrderDate	ProductID	Qty
1	2015-10-10	Bike	2
1	2015-10-10	Helmet	2
1	2015-10-10	Wheels	4
2	2015-10-09	Ball	10

XML vs JSON – Creation (Path)

XML

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         XML PATH;
```

JSON

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         JSON PATH;
```

XML vs JSON – Creation (Path)

XML

```
<row>
    <OrderID>1</OrderID>
    <OrderDate>2015-10-10T00:00:00
        </OrderDate>
    <ProductID>Bike</ProductID>
    <Qty>2</Qty>
</row>
<row>
    <OrderID>1</OrderID>
    <OrderDate>2015-10-10T00:00:00
        </OrderDate>
    <ProductID>Helmet</ProductID>
    <Qty>2</Qty>
</row>…
```

JSON

```
[{
    "OrderID":1,
    "OrderDate":"2015-10-10T00:00:00",
    "ProductID":"Bike",
    "Qty":2
},
{
    "OrderID":1,
    "OrderDate":"2015-10-10T00:00:00",
    "ProductID":"Helmet",
    "Qty":2
}…]
```

XML vs JSON – Creation (Auto)

XML

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         XML AUTO;
```

JSON

```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM  @Orders AS
        Orders
JOIN    @OrderDetails AS
        OrderDetails
ON Orders.OrderID =
        OrderDetails.OrderID
FOR         JSON AUTO;
```

XML vs JSON – Creation (Auto)

XML

```
<Orders
    OrderID="1"
    OrderDate="2015-10-10T00:00:00"
>
    <OrderDetails
        ProductID="Bike" Qty="2" />
    <OrderDetails
        ProductID="Helmet" Qty="2" />
    <OrderDetails
        ProductID="Wheels" Qty="4" />
</Orders>…
```

JSON

```
[{
"OrderID":1,
"OrderDate":"2015-10-10T00:00:00",
"OrderDetails":[
    {"ProductID":"Bike","Qty":2},
    {"ProductID":"Helmet","Qty":2},
    {"ProductID":"Wheels","Qty":4}]
},…
]
```


Notice there is no green on the XML side. There are no more text nodes.

XML vs JSON - Get Values

XML

```
DECLARE
   @x xml = '<x>y</x>';

SELECT
   @x.value
   (
      '(/x/text())[1]',
      'varchar(50)'
   );
```

JSON

```
DECLARE
   @j varchar(50) = '{"x":"y"}';

SELECT
   JSON_VALUE
   (
      @j,
      '$.x'
   );
```


XML

https://docs.microsoft.com/en-us/sql/t-sql/xml/value-method-xml-data-type

JSON

Regardless of the underlying type, it returns nvarchar. It has a size limit. Use OPENJSON to get around that.

https://docs.microsoft.com/en-us/sql/t-sql/functions/json-value-transact-sql

XML vs JSON - Getting Subsets

XML

```
DECLARE
   @x xml =
   '<x><y>z</y></x>';

SELECT
   @x.query ('(/x/y)');

Result:
   <y>z</y>
```

JSON

```
DECLARE
   @j varchar(50) =
   '{"x":{"y":"z"}';

SELECT
   JSON_QUERY (@j,'$.x');

Result:
   {"y":"z"}
```


https://docs.microsoft.com/en-us/sql/t-sql/xml/query-method-xml-data-type

https://docs.microsoft.com/en-us/sql/t-sql/functions/json-query-transact-sql

XML vs JSON – Getting Rows

- XML has OPENXML and nodes function. Both support XQuery.
- OPENXML
  - Requires “prepare document” step.
  - Separate T-SQL Statement– can’t be used in views or inline functions.
  - Might be faster for repeat access.
  - You have to remove the document from memory manually.
- Nodes
  - Can be used as part of T-SQL statement.
- OPENJSON works like nodes, but without the XQuery.

JSON Support in SQL Server 2016 - Jovan Popovic (MSFT) 16 May 2015 7:17 AM
http://blogs.msdn.com/b/jocapc/archive/2015/05/16/json-support-in-sql-server-2016.aspx

MSSQL Server 2016 coming with JSON support (not really)
http://www.itworld.com/article/2925117/enterprise-software/mssql-server-2016-coming-with-json-support-not-really.html

OPENXML - Query

```
DECLARE @i int, @x xml =
'<x>
  <Element attribute="Attribute Value">
      Element Value
  </Element>
  <y><z>Hello</z></y>
</x>';
EXEC sp_xml_preparedocument @i OUTPUT, @x;
SELECT * FROM OPENXML (@i,'/');
```

OPENXML - Results

id	parentid	nodetype	localname	prefix	namespaceuri	datatype	prev	text
0	NULL	1	x	NULL	NULL	NULL	NULL	NULL
2	0	1	Element	NULL	NULL	NULL	NULL	NULL
3	2	2	attribute	NULL	NULL	NULL	NULL	NULL
7	3	3	#text	NULL	NULL	NULL	NULL	Attribute Value
4	2	3	#text	NULL	NULL	NULL	NULL	Element Value
5	0	1	y	NULL	NULL	NULL	2	NULL
6	5	1	z	NULL	NULL	NULL	NULL	NULL
8	6	3	#text	NULL	NULL	NULL	NULL	Hello

OPENJSON

```
DECLARE @j varchar(max) =
'{
        "NULL": null,
        "String": "Hello",
        "Number": 123.4E05,
        "Boolean": true,
        "Array":[1,2,3],
        "JSON": {"a":"b"}
}';
SELECT          *
FROM        OPENJSON(@j);
```

key	value	type
NULL	NULL	0
String	Hello	1
Number	1.2E+07	2
Boolean	TRUE	3
Array	[1,2,3]	4
JSON	{"a":"b"}	5

XML vs JSON – Consuming (OPEN*)

OPENXML

```
DECLARE
    @i int, @x xml =
    '<x><a>1</a><a>2</a></x>';

EXEC sp_xml_preparedocument
@i OUTPUT, @x;

SELECT * FROM
        OPENXML (@i, '/x/a', 2)
WITH (a int '.');
```

OPENJSON

```
DECLARE
  @j varchar(max) =
  '{"x":[{"a":1},{"a":2}]}';

SELECT a.value FROM
OPENJSON (@j) AS x
CROSS APPLY OPENJSON (x.
[value]) AS a_array
CROSS APPLY OPENJSON
(a_array.[value]) AS a;
```


JSON requires no “prepare document” step.

Each OPENJSON peels off one layer. If the layer is an object, each property gets a row. If the layer is an array, each array entry gets a row.

https://docs.microsoft.com/en-us/sql/relational-databases/xml/openxml-sql-server

XML vs JSON – Consuming (Nodes)

XML Nodes()

```
DECLARE
  @x xml =
  '<x><a>1</a><a>2</a></x>';

SELECT
  a.value('.','int')
FROM @x.nodes('/x/a') AS x(a);
```

OPENJSON

```
DECLARE
  @j varchar(max) =
  '{"x":[{"a":1},{"a":2}]}';

SELECT a.value FROM
OPENJSON (@j) AS x
CROSS APPLY OPENJSON (x.
[value]) AS a_array
CROSS APPLY OPENJSON
(a_array.[value]) AS a;
```

XML vs JSON – Consuming (JSON v JSON)

OPENJSON

```
SELECT a.value
FROM
   OPENJSON (@j) AS x
CROSS APPLY
   OPENJSON
   (x.[value]) AS a_array
CROSS APPLY
   OPENJSON
   (a_array.[value]) AS a;
```

Combo

```
SELECT JSON_VALUE
(a_array.value,'$.a') FROM
(
      SELECT
      JSON_QUERY(@j,'$.x')
AS x
) xtable
CROSS APPLY OPENJSON
(xtable.x) AS a_array;
```

XML vs JSON – Data Type

- XML has a native type, but can be stored as nvarchar or varchar.
- JSON does *not* have a native type. Use nvarchar or varchar.
- Why not?
  - Already being stored as text.
    - But so was XML.
    - And so what? Convert over time. Convert on the fly.
  - Don’t have to update other SQL Server tools.
    - Boo hoo. Ok for now, but convert over time.
  - Client apps can handle native XML but not JSON.
    - Wait, what?
    - And so what if it’s text to the outside world; what about in-database performance?

JSON Support in SQL Server 2016 - Jovan Popovic (MSFT) 16 May 2015 7:17 AM
http://blogs.msdn.com/b/jocapc/archive/2015/05/16/json-support-in-sql-server-2016.aspx

MSSQL Server 2016 coming with JSON support (not really)
http://www.itworld.com/article/2925117/enterprise-software/mssql-server-2016-coming-with-json-support-not-really.html

Note that sometimes keeping XML as text can actually be faster.

XML vs JSON – Data Type – Validation

- Without JSON type, can’t use TRY_CONVERT() to validate.
- Use ISJSON() instead.
- Can use in CHECK constraint to ensure text field has valid JSON.
- Can then safely create calculated field based off JSON contents.

XML vs JSON – Data Type – Nesting Issue

XML

```
SELECT
CONVERT(xml,
'<TextXML>I typed this.</TextXML>'
) AS 'OuterTag'
FOR XML PATH('');
```

Results:

```
<OuterTag>
<TextXML>I typed this.</TextXML>
</OuterTag>
```

JSON

```
SELECT
'{"TextJSON":"I typed this."}' AS 'OuterTag'
FOR JSON PATH;
```

Results:

```
{"OuterTag":"{\"TextJSON\":\"I typed this.\"}"}
```

XML vs JSON – Data Type – Nesting Fix

```
SELECT
	(
		SELECT
			'I typed this.' AS TextJSON
		FOR JSON PATH
	) AS 'OuterTag'
FOR JSON PATH;
```

Results:

```
{"OuterTag":{"TextJSON":"I typed this."}}
```


You can do this with XML, too. You just wouldn't need to. However, using subqueries like this is probably a better practice than hard-coding the strings.

Additional Features (in SQL Server)

XML

- XPath
- DTDs
- Entities
- Schema
- Namespaces
- FLWOR
- XHTML (Sort of)
- SQLXML (Deprecated)

JSON

There are JSON-equivalents for some of these, but they are not really standards-based, and they aren’t part of SQL Server. XSLT is also available for XML, and is standards-based, but it’s not in SQL Server.

XML Feature: XQuery

```
DECLARE
  @x xml = '<r><x a="1">y</x><x a="2">z</x></r>';

SELECT
  @x.query('//x[@a>1]'),
  @x.query('//x[text()="z"]');
```

Result:

```
  <x a="2">z</x>
```


Just this simple query had a cost of 209.974.

XML Feature: XQuery - More Complex

```
DECLARE @x xml =
  '<r>
    <x a="1" b="2">
      <y b="2">PickMe!</y>
      <y b="3">No</y>
    </x>
    <x a="1" b="3">
      <y b="2">No</y>
    </x>
    <x a="2" b="2">
      <y b="2">No</y>
    </x>
  </r>';
```

```
SELECT
  @x.value('
  '(/r/x[@a=1 and @b=2]/y)[1]',
  'varchar(50)');
```

Result:
PickMe!

Just this simple query had a cost of 209.974.

XML Feature: DTDs / Entities

- SQL Server has “limited” DTD support.
- Provides Entity substitution.
- Provides default attribute values.
- Consumed by XML conversion. (One way trip.)
- Validation not supported by SQL Server.

XML Feature: DTDs / Entities

T-SQL

```
SELECT CONVERT(xml, N'
<!DOCTYPE Test [
<!ENTITY ReplaceMe
"Replacement">
<!ATTLIST Test Attr CDATA
"Default">]>
<Test>
   &ReplaceMe;
   &ReplaceMe;
</Test>
',2);
```

Result

```
<Test Attr="Default">
   Replacement
   Replacement
</Test>
```


SQL Server won't validate the DTD.

Once consumed, the DTD is not preserved.

https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql

XML Feature: Schema

- Provides data validation.
- Provides structure validation.
- Creates “typed” XML.
  - More efficient storage.
  - Allows XML indexes.
- Does not allow entity creation / substitution.
- Schema collection must be created in advance of use.

https://docs.microsoft.com/en-us/sql/relational-databases/xml/compare-typed-xml-to-untyped-xml

http://stackoverflow.com/questions/20157113/xsd-how-to-use-entity-in-xsd

XML Feature: Schema

T-SQL

```
CREATE XML SCHEMA COLLECTION
TestSchema AS

N'<schema xmlns="http://
www.w3.org/2001/XMLSchema">

<element name="Test"
type="integer" />

</schema>';

GO

SELECT CONVERT(xml
(TestSchema), N'<Test>a</
Test>');

GO

DROP XML SCHEMA COLLECTION
TestSchema;
```

Result

```
Msg 6926, Level 16,
State 1, Line 6

XML Validation: Invalid
simple type value: 'a'.
Location: /*:Test[1]
```


https://docs.microsoft.com/en-us/sql/t-sql/statements/create-xml-schema-collection-transact-sql

Counterpoint - JSON “Validation”

T-SQL

```
SELECT * FROM
OPENJSON('{"a":test}');
```

Result

```
Msg 13609, Level 16,
State 4, Line 1

JSON text is not
properly formatted.
Unexpected character
't' is found at
position 5.
```


Because JSON supports some primitive data types– numbers, strings, Booleans, and null, you can rely on OPENJSON (and related SQL JSON tools) to make sure that anything which isn’t in quotes is either a number, Boolean, or a null. Furthermore, OPENJSON will describe which type each value is.

XML Feature - Namespaces

- Allows disambiguation of element names.
- Makes for very ugly XML.
- Namespace requires “prefix” and “namespace identifier”.
  - “Prefix” is shorthand way to reference in XML elements.
  - “Namespace identifier” must be a URL or URN.
    - URLs were chosen with the idea that you would buy the domain to guarantee you owned that “space”.
    - But these don’t have to be actual, Internet accessible locations.
    - SQL Server does not navigate to the URLs.
- Requires special handling and syntax in T-SQL.

https://docs.microsoft.com/en-us/sql/relational-databases/xml/add-namespaces-to-queries-with-with-xmlnamespaces

XML Feature – Namespaces

T-SQL

```
DECLARE @x xml = N'
<a:x
xmlns:a="example.com">
Test
</a:x>';
SELECT
  @x.value('(/a:x)
[1]','varchar(50)');
```

Results

```
Msg 2229, Level 16,
State 1, Line 3
XQuery [value()]: The
name "a" does not
denote a namespace.
```

XML Feature – Namespaces

T-SQL

```
DECLARE @x xml = N'
<a:x
xmlns:a="example.com">
  Test
</a:x>';
SELECT
  @x.value('declare
namespace
a="example.com"; (/a:x)
[1]','varchar(50)');
```

Alternative T-SQL

```
DECLARE @x xml = N'
<a:x
xmlns:a="example.com">
  Test
</a:x>';
SELECT
  @x.value('(/*:x)
[1]','varchar(50)');
```


https://docs.microsoft.com/en-us/sql/t-sql/xml/value-method-xml-data-type

Could also use WITH NAMESPACES

XML Feature: FLWOR

- FOR, LET, WHERE, ORDER BY, RETURN
- There's a whole programming language inside of XML.
- You can loop, do calculations, and construct XML.
- There are special cases where this makes sense, but there are often better ways.

https://docs.microsoft.com/en-us/sql/xquery/flwor-statement-and-iteration-xquery

XML Feature: FLWOR

T-SQL

```
DECLARE @x xml = N'
   <x>   <a>1</a>
         <b>2</b>
         <c>3</c>   </x>';
SELECT  @x.query('
for $n in x/*
order by ($n/text())[1]*-1
return
   <num>
      {(($n/text())[1])+1}
   </num>     ');
```

Result

```
<num>4</num>
<num>3</num>
<num>2</num>
```


https://www.w3schools.com/xml/xquery_flwor.asp

https://sqljudo.wordpress.com/2013/12/26/xquery-for-the-sql-dba-part-three/

XML Feature: XHTML

- XHTML is not a SQL Server feature per se.
- You can construct XHTML code using T-SQL XML features.
- You could use this to construct entire web pages (laboriously).
- This will perform much more poorly than string concatenation.
- But you don’t have to worry about syntactical mistakes.
- You could use this to construct pretty HTML-style emails to be sent using SQL Server, without any outside toolset.

https://en.wikipedia.org/wiki/XHTML

XML Feature: XHMTL

T-SQL

```
SELECT
    'Hello, world!' AS 'div'
FOR
    XML
    PATH ('body'),
    ROOT('html'),
    TYPE;
```

Result

```
<html>
    <body>
        <div>Hello, world!</div>
    </body>
</html>
```

Gotchas

XML

- Must have root element (but SQL more forgiving).
- No repeated attribute names.
- Funky whitespace handling.
- No colons in element names.
- No low level ASCII (except CR LF TAB).
- Character restrictions for element names.
- Exact text not preserved in SQL Server XML data type.

JSON

- No comments.
- Repeated key names are variably supported. (Use array instead.)
- “Root” can be array or object.

XML:

SQL Server doesn’t care whether there’s an XML declaration or a root element.

JSON:

If you have two name/value pairs at the same level with the same name, the second value might not be easy to extract depending on the tool. E.g. SQL Server can only grab first value using JSON_VALUE (must use OPENJSON otherwise). JavaScript results in last value.

https://docs.microsoft.com/en-us/sql/relational-databases/json/solve-common-issues-with-json-in-sql-server

Note: must be in a database with a compatibility level equal to or greater than 130.

Both:

Both are case sensitive.

Conciseness

- Shorter is not necessarily better.
- Raw binary data is most efficient, but it's not human readable.
- Even human readable code can be impractically terse.

This is a valid program written in the language 05AB1E. It is a "quine", a program which prints itself without reading its source code.

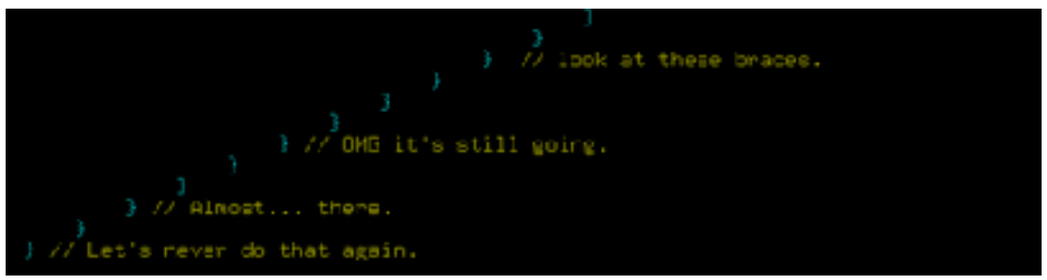
0"D34çý"D34çý

http://codegolf.stackexchange.com/a/97899

http://stackoverflow.com/a/13679245/2266979

Conciseness

- Sometimes, more characters are better.
- XML's extra characters come from labeling the end of a section.
- That can help with navigation in a complex document.



https://twitter.com/icculus/status/376575908577935360/photo/1

Conciseness

- XML stored as optimized binary (MS-BINXML).
- Compression is now in SQL Server Standard edition (SP1).
- HTTPs/HTTP2 makes automatic compression widespread.

https://blogs.msdn.microsoft.com/sqlreleaseservices/sql-server-2016-service-pack-1-sp1-released/

https://en.wikipedia.org/wiki/HTTP/2

Speed

- JSON parsing is significantly faster in SQL Server and elsewhere.
- XML, especially with multiple XQuery expressions, will create very complex query plans. Even if not slower to execute, slower to compile.

http://www.cs.tufts.edu/comp/150IDS/final_papers/tstras01.1/FinalReport/FinalReport.html

https://blogs.msdn.microsoft.com/sqlserverstorageengine/2016/01/14/json-parsing-performance-comparison/

XML vs JSON - Winner?

XML

- Microsoft Ecosystem
- XQuery
- Features
- Close Tags

JSON

- Web Ecosystem
- Simpler
- Faster

Riley Major

- @RileyMajor | PASSMN@RileyMajor.com
- Enterprise Architect
- Manna Freight Systems, Inc.
- Worked with SQL Server since May of 2000
- PASSMN Board – Director of SQL Saturday
- Conference speaker
- Father of three girls

Image Credits

- Page 1
  - 4381948277_66eb46cc2e_o.jpg
  - GRU - Melee 333: Dragoon Jumping and Double Spears Technique
  - Kevin Thai
  - https://www.flickr.com/photos/kthai/4381948277