**Q1: What is XML?**

A1: Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services. Several schema systems exist to aid in the definition of XML-based languages, while programmers have developed many application programming interfaces (APIs) to aid the processing of XML data.

The material in this section is based on the XML Specification. This is not an exhaustive list of all the constructs that appear in XML; it provides an introduction to the key constructs most often encountered in day-to-day use.

**Character**

An XML document is a string of characters. Almost every legal Unicode character may appear in an XML document.

Processor and application. The processor analyses the markup and passes structured information to an application. The specification places requirements on what an XML processor must do and not do, but the application is outside its scope. The processor (as the specification calls it) is often referred to colloquially as an XML parser.

**Markup and content**

The characters making up an XML document are divided into markup and content, which may be distinguished by the application of simple syntactic rules. Generally, strings that constitute markup either begin with the character < and end with a >, or they begin with the character & and end with a ;. Strings of characters that are not markup are content. However, in a CDATA section, the delimiters <! [CDATA[ and ]]> are classified as markup, while the text between them is classified as content. In addition, whitespace before and after the outermost element is classified as markup.

**Tag**

A tag is a markup construct that begins with < and ends with >. Tags come in three flavors:

start-tag, such as <section>;

end-tag, such as </section>;

empty-element tag, such as <line-break />.

**Element**

An element is a logical document component that either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The characters between the start-tag and end-tag, if any, are the element's content, and may contain markup, including other elements, which are called child elements. An example is <greeting>Hello, world!</greeting>. Another is <line-break />.

**Attribute**

An attribute is a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag. An example is <img src="madonna.jpg" alt="Madonna" />, where the names of the attributes are "src" and "alt", and their values are "madonna.jpg" and "Madonna" respectively. Another example is <step number="3">Connect A to B.</step>, where the name of the attribute is "number" and its value is "3". An XML attribute can only have a single value and each attribute can appear at most once on each element. In the common situation where a list of multiple values is desired, this must be done by encoding the list into a well-formed XML attribute[i] with some format beyond what XML defines itself. Usually this is either a comma or semi-colon delimited list or, if the individual values are known not to contain spaces,[ii] a space-delimited list can be used. <div class="inner greeting-box">Welcome!</div>, where the attribute "class" has both the value "inner greeting-box" and also indicates the two CSS class names "inner" and "greeting-box".

**XML declaration**

XML documents may begin with an XML declaration that describes some information about themselves. An example is <?xml version="1.0" encoding="UTF-8"?>.

**Q2: What is XSLT?**

**A2:** XSLT (Extensible Stylesheet Language Transformations) is a language for transforming XML documents into other XML documents, or other formats such as HTML for web pages, plain text or XSL Formatting Objects, which may subsequently be converted to other formats, such as PDF, PostScript and PNG.

The original document is not changed; rather, a new document is created based on the content of an existing one. Typically, input documents are XML files, but anything from which the processor can build an XQuery and XPath Data Model can be used, such as relational database tables or geographical information systems.

XSLT 2.0: after an abortive attempt to create a version 1.1 in 2001, the XSL working group joined forces with the XQuery working group to create XPath 2.0, with a richer data model and type system based on XML Schema. Building on this is XSLT 2.0, developed under the editorship of Michael Kay, which reached recommendation status in January 2007.The most important innovations in XSLT 2.0 include:

* String manipulation using regular expressions
* Functions and operators for manipulating dates, times, and durations
* Multiple output documents
* Grouping (creating hierarchic structure from flat input sequences)
* A richer type system and stronger type checking

Design and processing model

**Basic elements and process flow of Extensible Stylesheet Language Transformations**:

The XSLT processor takes one or more XML source documents, plus one or more XSLT stylesheets, and processes them to produce an output document. In contrast to widely implemented imperative programming languages like C, XSLT is declarative. The basic processing paradigm is pattern matching. Rather than listing an imperative sequence of actions to perform in a stateful environment, template rules only define how to handle a node matching a particular XPath-like pattern, if the processor should happen to encounter one, and the contents of the templates effectively comprise functional expressions that directly represent their evaluated form: the result tree, which is the basis of the processor's output.

**XPath**

XSLT uses XPath to identify subsets of the source document tree and perform calculations. XPath also provides a range of functions, which XSLT itself further augments.

XSLT 1.0 uses XPath 1.0, while XSLT 2.0 uses XPath 2.0. XSLT 3.0 will work with either XPath 3.0 or 3.1. In the case of 1.0 and 2.0, the XSLT and XPath specifications were published on the same date.

**Q3: What is JSON?**

**A3:** JSON is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types. It is a very common data format, with a diverse range of applications, such as serving as a replacement for XML in AJAX systems.JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is application/json. JSON filenames use the extension.

**Syntax**

The following example shows a possible JSON representation describing a person.

{

"firstName": "John",

"lastName": "Smith",

"isAlive": true,

"age": 27,

"address": {

"streetAddress": "21 2nd Street",

"city": "New York",

"state": "NY",

"postalCode": "10021-3100"

},

"phoneNumbers": [

{

"type": "home",

"number": "212 555-1234"

},

{

"type": "office",

"number": "646 555-4567"

}

],

"children": [],

"spouse": null

}

**Data types**

JSON's basic data types are:

* **Number:** a signed decimal number that may contain a fractional part and may use exponential E notation, but cannot include non-numbers such as NaN. The format makes no distinction between integer and floating-point. JavaScript uses a double-precision floating-point format for all its numeric values (until later also supports BigInt[25]), but other languages implementing JSON may encode numbers differently.
* **String:** a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
* **Boolean:** either of the values true or false
* **Array:** an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation with comma-separated elements.
* **Object:** a collection of name–value pairs where the names (also called keys) are strings. Objects are intended to represent associative arrays,[2] where each key is unique within an object. Objects are delimited with curly brackets and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.
* null: an empty value, using the word null

**Q4: Give an example of JSON?**

**A4: Sending Data:** If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

Example

var myObj = {name: "John", age: 31, city: "New York"};

var myJSON = JSON.stringify(myObj);

window.location = "demo\_json.php?x=" + myJSON;

<!DOCTYPE html>

<html>

<body>

<h2>Convert a JavaScript object into a JSON string, and send it to the server.</h2>

<script>

var myObj = { name: "John", age: 31, city: "New York" };

var myJSON = JSON.stringify(myObj);

window.location = "demo\_json.php?x=" + myJSON;

</script>

</body>

</html>

**Receiving Data**

If you receive data in JSON format, you can convert it into a JavaScript object:

Example

var myJSON = '{"name":"John", "age":31, "city":"New York"}';

var myObj = JSON.parse(myJSON);

document.getElementById("demo").innerHTML = myObj.name;

<!DOCTYPE html>

<html>

<body>

<h2>Convert a string written in JSON format, into a JavaScript object.</h2>

<p id="demo"></p>

<script>

var myJSON = '{"name":"John", "age":31, "city":"New York"}';

var myObj = JSON.parse(myJSON);

document.getElementById("demo").innerHTML = myObj.name;

</script>

</body>

</html>

**Storing Data**

When storing data, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

**Example**

**Storing data in local storage**

// Storing data:

myObj = {name: "John", age: 31, city: "New York"};

myJSON = JSON.stringify(myObj);

localStorage.setItem("testJSON", myJSON);

// Retrieving data:

text = localStorage.getItem("testJSON");

obj = JSON.parse(text);

document.getElemen <!DOCTYPE html>

<html>

<body>

<h2>Store and retrieve data from local storage.</h2>

<p id="demo"></p>

<script>

var myObj, myJSON, text, obj;

// Storing data:

myObj = { name: "John", age: 31, city: "New York" };

myJSON = JSON.stringify(myObj);

localStorage.setItem("testJSON", myJSON);

// Retrieving data:

text = localStorage.getItem("testJSON");

obj = JSON.parse(text);

document.getElementById("demo").innerHTML = obj.name;

</script>

</body>

</html> tById("demo").innerHTML = obj.name;

**Q5: What is API?**

**A5:** API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you’re using an API.

An application programming interface (API) is a computing interface that defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc. It can also provide extension mechanisms so that users can extend existing functionality in various ways and to varying degrees. An API can be entirely custom, specific to a component, or designed based on an industry-standard to ensure interoperability. Through information hiding, APIs enable modular programming, allowing users to use the interface independently of the implementation.

In building applications, an API (application programming interface) simplifies programming by abstracting the underlying implementation and only exposing objects or actions the developer needs. While a graphical interface for an email client might provide a user with a button that performs all the steps for fetching and highlighting new emails, an API for file input/output might give the developer a function that copies a file from one location to another without requiring that the developer understand the file system operations occurring behind the scenes.

In building applications, an API (application programming interface) simplifies programming by abstracting the underlying implementation and only exposing objects or actions the developer needs. While a graphical interface for an email client might provide a user with a button that performs all the steps for fetching and highlighting new emails, an API for file input/output might give the developer a function that copies a file from one location to another without requiring that the developer understand the file system operations occurring behind the scenes.

An API can also be related to a software framework: a framework can be based on several libraries implementing several APIs, but unlike the normal use of an API, the access to the behavior built into the framework is mediated by extending its content with new classes plugged into the framework itself.

Moreover, the overall program flow of control can be out of the control of the caller and in the framework's hands by inversion of control or a similar mechanism.

**Q6: Define Browser API?**

**A6:** Web API is an application programming interface for either a web server or a web browser. It is a web development concept, usually limited to a web application's client-side (including any web frameworks being used), and thus usually does not include web server or browser implementation details such as SAPIs or APIs unless publicly accessible by a remote web application.

**Server side**

A server-side web API is a programmatic interface consisting of one or more publicly exposed endpoints to a defined request–response message system, typically expressed in JSON or XML, which is exposed via the web—most commonly by means of an HTTP-based web server. Mashups are web applications which combine the use of multiple server-side web APIs. Webhooks are server-side web APIs that take input as a Uniform Resource Identifier (URI) that is designed to be used like a remote named pipe or a type of callback such that the server acts as a client to dereference the provided URI and trigger an event on another server which handles this event thus providing a type of peer-to-peer IPC.

**Endpoints:**

Endpoints are important aspects of interacting with server-side web APIs, as they specify where resources lie that can be accessed by third party software. Usually the access is via a URI to which HTTP requests are posted, and from which the response is thus expected.

Web 2.0 Web APIs often use machine-based interactions such as REST and SOAP. RESTful web APIs are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data. By contrast, SOAP protocols are standardized by the W3C and mandate the use of XML as the payload format, typically over HTTP. Furthermore, SOAP-based Web APIs use XML validation to ensure structural message integrity, by leveraging the XML schemas provisioned with WSDL documents. A WSDL document accurately defines the XML messages and transport bindings of a Web service.

**Growth and impact**

The number of available web APIs has grown consistently over the past years, as businesses realize the growth opportunities associated with running an open platform, that any developer can interact with. Programmable Web tracks 9000 Web APIs that were available in 2013, up from 105 in 2005.

Web APIs have become ubiquitous. There are few major software applications/services that do not offer some form of web API. One of the most common forms of interacting with these web APIs is via embedding external resources, such as tweets, Facebook comments, YouTube videos, etc. In fact, there are very successful companies, such as Disqus, whose main service is to provide embeddable tools, such as a feature-rich comment system. Any website of the TOP 100 Alexa Internet ranked websites uses APIs and/or provides its own APIs, which is a very distinct indicator for the prodigious scale and impact of web APIs as a whole.

**Q7: Define third-party API?**

**A7:** Third party APIs are APIs provided by third parties — generally companies such as Facebook, Twitter, or Google — to allow you to access their functionality via JavaScript and use it on your site. One of the most obvious examples is using mapping APIs to display custom maps on your pages.

The APIs we've covered so far are built into the browser, but not all APIs are. Many large websites and services such as Google Maps, Twitter, Facebook, PayPal, etc. provide APIs allowing developers to make use of their data (e.g. displaying your twitter stream on your blog) or services (e.g. using Facebook login to log in your users). This article looks at the difference between browser APIs and 3rd party APIs and shows some typical uses of the latter.

Third party APIs are APIs provided by third parties — generally companies such as Facebook, Twitter, or Google — to allow you to access their functionality via JavaScript and use it on your site. One of the most obvious examples is using mapping APIs to display custom maps on your pages.

Let's look at a Simple Mapquest API example, and use it to illustrate how third-party APIs differ from browser APIs.

Browser APIs are built into the browser — you can access them from JavaScript immediately. For example, the Web Audio API we saw in the Introductory article is accessed using the native AudioContext object. For example:

const audioCtx = new AudioContext();

...

const audioElement = document.querySelector('audio');

...

const audioSource = audioCtx.createMediaElementSource(audioElement);

// etc. Third party APIs, on the other hand, are located on third party servers. To access them from JavaScript you first need to connect to the API functionality and make it available on your page. This typically involves first linking to a JavaScript library available on the server via a <script> element, as seen in our Mapquest example:

<script src="https://api.mqcdn.com/sdk/mapquest-js/v1.3.2/mapquest.js"></script>

<link type="text/css" rel="stylesheet" href="https://api.mqcdn.com/sdk/mapquest-js/v1.3.2/mapquest.css"/>

You can then start using the objects available in that library. For example:

let map = L.mapquest.map('map', {

center: [53.480759, -2.242631],

layers: L.mapquest.tileLayer('map'),

zoom: 12

});

**Q8: Give an example of XML?**

**A8**: XML Example

XML documents create a hierarchical structure looks like a tree so it is known as XML Tree that starts at "the root" and branches to "the leaves".

Example of XML Document

XML documents uses a self-describing and simple syntax:

<?xml version="1.0" encoding="ISO-8859-1"?>

<note>

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the root element of the document (like saying: "this document is a note"):

<note>

The next 4 lines describe 4 child elements of the root (to, from, heading, and body).

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

And finally the last line defines the end of the root element.

</note>

XML documents must contain a root element. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements).

<root>

<child>

<subchild>.....</subchild>

</child>

</root>

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters). All elements can have text content and attributes (just like in HTML).

**Q9: Differentiate between XML and JSON?**

**A9:** SON (JavaScript Object Notation) is a lightweight data-interchange format and it completely language independent. It is based on the JavaScript programming language and easy to understand and generate.

Example :

filter\_none

brightness\_4

{"Geeks":[

{ "firstName":"Vivek", "lastName":"Kothari" },

{ "firstName":"Suraj", "lastName":"Kumar" },

{ "firstName":"John", "lastName":"Smith" },

{ "firstName":"Peter", "lastName":"Gregory" }

]}

XML (Extensible markup language) was designed to carry data, not to display data. It is a W3C recommendation. Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML focus on simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

Example :

filter\_none

brightness\_4

<Geeks>

<Geek>

<firstName>Vivek</firstName> <lastName>Kothari</lastName>

</Geek>

<Geek>

<firstName>Suraj</firstName> <lastName>Kumar</lastName>

</Geek>

<Geek>

<firstName>John</firstName> <lastName>Smith</lastName>

</Geek>

<Geek>

<firstName>Peter</firstName> <lastName>Gregory</lastName>

</Geek>

</Geeks>

Both of these are self-describing and can be parsed and used by lots of programming languages.

|  |  |
| --- | --- |
| **JSON** | **XML** |
| It is JavaScript Object Notation | It is Extensible markup language |
| It is based on JavaScript language. | It is derived from SGML. |
| It is a way of representing objects. | It is a markup language and uses tag structure to represent data items. |
| It does not provide any support for namespaces. | It supports namespaces. |
| It supports array. | It doesn’t supports array. |
| Its files are very easy to read as compared to XML. | Its documents are comparatively difficult to read and interpret. |
| It doesn’t use end tag. | It has start and end tags. |
| It is less secured. | It is more secured than JSON. |
| It doesn’t supports comments. | It supports comments. |
| It supports only UTF-8 encoding. | It supports various encoding. |

**Q10: Why XML is outdated?**

**A10:** XML is still widely used by companies whose systems are older and it would cost a fortune to update. XML is still widely used by companies whose systems are older and it would cost a fortune to update. If the idea was to change the format at a low level such as output from the database, then yes, it would be expensive. As both JSON and XML seem to solve a somewhat similar purpose, the comparison is mostly based on the features and benefits of each. The bottom line is that if you have simple data exchanges that need not bother about semantics and validation, go for JSON. But, even with JSON, it is essential to learn XML and related technologies to do more than just data exchange and fast processing! JSON still has a long way to go before it comes at par with the powers of the ‘old is gold’ XML. In contrast to XML, JSON is designed for data and data only. That is why we see its adoption in the space where data is being transferred between applications such as through APIs. JSON’s popularity over XML is partly due to it being pure data that’s formatted in a way that’s also human readable.

Douglas Crockford designed JSON in the early 2000s to be simple and highly predictable in design. While JSON is much younger in comparison to XML, it is more widely used and consumed by APIs, despite only being officially standardized in 2013.

JSON itself is a language-independent data format and was a response to the need for a stateless communication protocol without the need for additional plugins to process.

With the web’s growing support for JavaScript, JSON was created as a subset of JavaScript. This helped form the strong relationship between JSON and JavaScript, pushing XML to the side as a result.

Despite JSON being the staple in modern app development, XML exists in many enterprise systems and has moved on from being used as a data transfer format. It is good to note that despite being used in the data layer, XML wasn’t created specifically for storing and sending data. The idea behind markup languages is that it’s used to structure documents through tags to define elements. XML rise to prominence in the data layer is due to its ability to use custom tags.

XML is now commonly found as part of UI development processes in spaces such as Android, JavaFX, UPF and Xmarin. It is also extensively used in Magneto, in addition to Microsoft based programs such as Word, Excel, and Powerpoint.

Being a markup language, XML makes the perfect platform-agnostic choice when it comes to displaying and styling data. It comes ready and preformatted to the needs of the program and application consuming it. The point of markup languages is that it gives the data structure, with the ability to create hooks and interfaces for other languages to work with.

The pros and cons of XML

XML still lives today, mainly because it is platform agnostic. It supports Unicode and is often used as part of a data presentation workflow.

This is why XML is still useful to learn, especially in spaces where UI development is required. Native Android development, for example, heavily depends on XML to format UI elements. Being a markup language that has no predefined tags, XML is being employed in spaces where the GUI (graphic user interface) is not a browser.

The verbosity of XML can create high data transportation costs, especially if the data is high in volume. It is less human-readable than other data formats available and doesn’t support arrays in a way that is easy to process and understand.

When used in the data layer, XML can end up unnecessarily expensive due to size. Its optimization is dependant on the developer designing the structure, rather than being intrinsic in its normal modes and structures.

As a data layer, XML is prone to redundancies in structure and is much fitter for purpose in the UI rather than data layer space.

**Final thoughts**

* JSON is often the first pick when it comes to web-based integration because it’s easy to read and consume by apps. Despite having its origins in JavaScript, it has grown to be widely supported, with legacy systems depending on XML to work. JSON has also become the standard data format for many tableness databases, turning JSON into the default and puts it in the forefront.
* However, just because JSON is currently the most talked about due to its association with JavaScript and extensive support across different languages and platforms, you shouldn’t completely disregard XML.
* The thing with XML is that it has its place in the app development ecosystem and is not fading away into complete obscurity. There’s more to data than just moving it from one space to another. XML does something that JSON cannot do – it can dress it up and allow languages like Java and C++ to create interfaces for applications that may not be web-based.