XML (eXtensible Markup Language)

***\*Introductory note to better understand these notes****: Although XML is commonly referred to as a "language", it is more accurate to think of it as a* ***format (or syntax)*** *for organising data. This is not just an analogy; it is a precise description of what XML actually is.*

*A real analogy would be to compare it to the language-independent structure found in most (if not all) modern languages, such as sentences, paragraphs, headings, chapters, and lists. Just as these structural elements can be used in various languages to organise content, XML provides a flexible framework for organising data without being tied to specific tags or meanings.*

# What is XML?

**XML (eXtensible Markup Language)** is a markup language (format) that is both human-readable and machine-readable. XML is used to store and transport data in a structured format, making it easier to share and manipulate data across different systems.

## Common Uses

Some common uses of XML include (but are not limited to):

* **Data Interchange:** XML is commonly used for exchanging data between different systems, especially when systems are heterogenous (e.g., different programming languages or platforms). Applications can read XML files, parse the data, and convert it into a format they can work with internally. For example, web services might use XML to transmit data between a client and server.
* **Configuration Files**: Many applications use XML for configuration purposes. XML's hierarchical structure is ideal for representing complex configuration data.
* **Document Storage and Processing**: Office suites like Microsoft Office or LibreOffice often use XML-based formats for their files. For example, `.docx` and `.xlsx` files are ZIP archives containing XML files that define a document's structure, content, and formatting.
* **Data Serialisation**: Applications often use XML for serialising objects or data structures. This serialised XML can be stored, transmitted, or deserialised later back into an object or data structure in the receiving application.

## XML Elements (Tags)

XML elements are discrete (distinct and self-contained, not overlapping with other elements) units of information in an XML document that represent data or content. XML tags serve as containers that collectively define the structure and meaning of the data they hold, organising it in a hierarchical, tree-like format.

### Purpose of XML Elements

* **Data Encapsulation**: XML elements encapsulate data, allowing it to be organised into meaningful, nested structures.
* **Hierarchical Organisation**: Elements create a hierarchy, with parent elements containing child elements, representing complex data relationships.
* **Semantic Meaning**: Elements provide context and meaning to the data, allowing it to be easily understood and processed by humans and machines.
* **Data Interchange**: XML elements are used to structure data in a standardised way, making it easy to exchange information between different systems and applications.

XML elements are defined using tags enclosed in angle brackets (`< >`), using the same syntax as HTML tags.

### Flexibility and Customisable Nature of XML Tags (Contrast to HTML) - XML Vocabularies

In contrast to an actual markup language (not just a format) like HTML, which has a specific, discrete set of predefined tags with fixed meanings, XML (in general) is not confined to a particular set of tags. XML is a flexible markup language (format) that allows users and developers to create their own custom tags tailored to their specific needs.

That being said, in reality, there are finite sets of XML tags that are used for a specific purpose, with specific applications that have been designed to handle these tags in a specific way. Each of the examples in the "Common Uses" list near the beginning of this document has its own specific, finite set of XML tags that the relevant application is programmed to handle in a specific way. While any specific name can be used for a tag, the application that is to handle this tag must be updated with code that should execute in response to this tag.

For example, the source code of Word and Excel would contain methods designed to handle the specific XML tags that are used to structure their data. Although Microsoft Office source code is closed-source, it can be assumed that they would use tags with names like <paragraph>, <page>, and <heading> in Word, and <cell>, <table>, and <sheet> in Excel, among many others.

Different, and potentially completely unrelated applications may use tags with the same name, but for totally different purposes. For example, while a <cell> in Excel is used to represent a spreadsheet cell, a biological modelling software may use a <cell> tag to represent a biological cell (more on this in the XML namespaces section). To use the human language analogy from earlier, a Geology or Earth sciences textbook may have a section with the name "Rock", but this would have a completely different meaning (and use) to a section with the same name in a music textbook. This is why, when such naming collisions occur on a website like Wikipedia, there is a "disambiguation" section, where these naming collisions are clarified (i.e., "Rock (Mineral)" and "Rock (Musical Genre)") - this also relates to XML namespaces. Nonetheless, assuming these books were from the same publisher (with the publisher being analogous to XML in general), the "Rock" title would look exactly the same in both books, having the same font size, colour, and alignment etc.

#### **XML Vocabularies**

The collection of tags, rules, and conventions for creating XML documents that are meaningful within a specific context are known as XML **vocabularies**. For example, in the above examples, all the XML used in Microsoft Word makes up a vocabulary, the XML for Excel is another vocabulary, as is the XML for the biological modelling software.

* FXML is an XML vocabulary specifically designed for JavaFX, mapping XML tags to JavaFX components.
* The XML tags used in Microsoft Office documents, spreadsheets, presentations etc., are part of the Office Open XML (OOXML) vocabulary.

## XML Namespaces

Namespaces in XML are used to avoid element (tag) name conflicts when combining XML documents from different XML vocabularies. They help uniquely identify elements in an XML document by associating them with a unique identifier (usually a URI).

1. **Creating a New Namespace**:

A namespace is declared by specifying an `xmlns` attribute in an element within an XML document. This namespace is identified by a unique URI. The `xmlns` attribute binds a prefix (e.g., `ecommerce`) to the URI (e.g., `https://shopify.com/schema`). This is a local declaration specific to the document and does not modify or create global namespaces across multiple documents or systems.

In this case, `ecommerce` is a new namespace within this document.

1. **Using an Existing Namespace**:

If the namespace (URI) has already been defined in other XML documents, and you use the same namespace (URI) in a new document, you are not creating a new namespace but reusing an existing one by declaring it again in your document. The namespace URI is what matters - not the prefix. Multiple documents can use the same namespace URI with different prefixes, and they would still be considered part of the same logical namespace.

Applications that process XML are programmed to process XML elements having a specific namespace URI. That is, for an application to process elements properly, their URI namespace must match the **URI** **namespace** that the application is designed to handle. The prefix does not matter.

For example, in JavaFX, the `FXMLLoader` specifically looks for the namespace URI `http://javafx.com/fxml` to recognise and properly handle JavaFX scene graph elements and FXML-specific attributes. This URI is predefined by the JavaFX framework, and the `FXMLLoader` is programmed to understand and process elements within this namespace.

***\*An `xmlns` attribute (which declares a namespace URI), only needs to be specified once in a document (or in a particular scope within the document), and all other elements having the same namespace can simply use the same prefix to reference the namespace without needing to redeclare the URI.***

1. **Adding Elements to a Namespace**:

Specifying the `xmlns` attribute with a prefix does not "add" elements to an existing namespace in a global sense, such as when using namespaces in programming languages. It simply associates the elements having that prefix in your XML document with that namespace for the scope of that document.

1. **Child Elements**:

By default, all child elements (those within outer tags) have the same namespace as their parent elements. While this is the case even when a namespace is not explicitly restated within the child element tags, it is recommended to explicitly declare the namespace in child elements when integrating or merging XML documents from different sources or when clarity is crucial.

### Practical Demonstration:

#### Syntax

A namespace is declared using the `xmlns` attribute within an element. This declaration associates a URI with a prefix, which can then be used to distinguish elements and attributes that belong to different namespaces.

*\* You can declare a namespace within any element - even one that isn't part of the namespace you are declaring. The `xmlns` namespace is simply used to declare the namespace and associate it with a prefix. The element where the `xmlns` is declared doesn't have to be part of the declared namespace.*

<spaceResearch xmlns:spacex="http://www.spacex.com/schemas/space" xmlns:nasa="http://www.nasa.gov/schemas/space">

In the above example:

* The `spaceResearch` element does not belong to any specific namespace and exists in the global/default namespace.
* The `spacex` prefix references the `http://www.spacex.com/schemas/space` URI (which identifies the actual namespace).
* The `nasa` prefix references the `http://www.nasa.gov/schemas/space` URI (which identifies this namespace).

The rest of the XML may look like this:

<spaceResearch xmlns:spacex="http://www.spacex.com/schemas/space"

xmlns:nasa="http://www.nasa.gov/schemas/space">

<!-- SpaceX Mission Details -->

<spacex:mission>

<spacex:name>Starship to Mars</spacex:name>

<spacex:launchDate>2025-03-14</spacex:launchDate>

<spacex:rocket>Starship</spacex:rocket>

</spacex:mission>

<!-- NASA Mission Details -->

<nasa:mission>

<nasa:name>Artemis I</nasa:name>

<nasa:launchDate>2024-11-16</nasa:launchDate>

<nasa:rocket>Space Launch System</nasa:rocket>

</nasa:mission>

</spaceResearch>

#### URI Determination

The URIs to be used for XML namespaces are typically chosen based on the following characteristics:

* **Uniqueness**: The primary purpose of a namespace URI is to provide a unique identifier that distinguishes one XML vocabulary from another. To ensure uniqueness, organisations often base the URI on their domain name, which is globally unique.
* **Readability and Organisation**: The URI often reflects the organisation or project to make it clear and understandable. It may also include a path that represents the purpose or version of the XML schema. For example, a URI like `http://www.spacex.com/schemas/space indicates that the namespace is related to SpaceX and pertains to space-related data or schemas.
* **Convention**: It is common practice to use the HTTP or HTTPS protocol for the URI, even though the URI does not have to point to an actual web resource. This practice is mostly for familiarity and consistency. The URI might be structured to include the organisation's domain, followed by relevant paths that indicate the specific schema or XML vocabulary.
* **Versioning**: Sometimes URIs include a version number or date to indicate the version of the schema being used. This is helpful when managing multiple versions of an XML schema.

Example: `http://www.nasa.gov/schemas/space/2024`

* **Non-Resolvability**: The URI used for an XML namespace is not required to be resolvable, meaning it doesn't have to lead to an actual webpage or resource. It is purely an identifier.

#### Using Prefixes to Identify the Namespace an Element Belongs To

To identify that an element belongs to a specific namespace, the name of the tag should be prefixed with the namespace prefix, as in each tag below:

<spacex:mission>

<spacex:name>Starship to Mars</spacex:name>

<spacex:launchDate>2025-03-14</spacex:launchDate>

<spacex:rocket>Starship</spacex:rocket>

</spacex:mission>

While each child element (`spacex:name`, `spacex:launchDate`, and `spacex:rocket`) would still implicitly be resolved using the URI associated with the `spacex` prefix (due to being children of an element belonging to this namespace), it is often considered good practice to explicitly label these as such.

#### **Default Namespaces**

A namespace does not necessarily need to be declared with a prefix. When you declare a namespace with no prefix, it defines the **default namespace** for:

* The element in whose tag it is defined.
* All child elements of that element.

For example, in the individual XML files of SpaceX or NASA, for projects without collaboration between the organisations, they could use the following namespace declaration:

<spaceResearch xmlns="http://www.spacex.com/schemas/space">…</spaceResearch>

Assuming this XML document would not be shared between organisations and all elements within it would share the same namespace, there is no need to assign a prefix for the namespace. Therefore, in this XML document, all non-prefixed elements (tags) will be treated as belonging to the `http://www.spacex.com/schemas/space` namespace.`

*\*The use of a default namespace simplifies the XML by eliminating the need for repeated prefixes.*

## XML Attributes (Arguments)

XML attributes are additional pieces of information that provide extra details about an XML element. They are used to add metadata or qualifiers to elements, allowing for more precise definitions of the data contained within the elements.

<elementName attributeName="attributeValue">content</elementName>

XML attributes have several purposes:

1. **Metadata or Qualifiers**: Attributes provide extra information about an element without changing its content. For example, in an element representing a book, attributes might store the title, author, and year of publication.

<book title="1984" author="George Orwell" year="1949"/>

1. **Configuration or Settings**: Attributes can be used to store configuration data or settings for an element. For instance, a `<button> element might have attributes to specify its colour or size.

<button colour="blue" size="large">Submit</button>

1. **Uniqueness**: Attributes can help distinguish between different instances of the same element. For example, if you have multiple `<student>` elements, you might use an `id` attribute to uniquely identify each one.

<student id="001" name="Alice"/>

<student id="002" name="Bob"/>

1. **Reference IDs**: Attributes can store references to other elements or data, often using IDs or links.

<image src="logo.png" alt="Brand Logo"/>

### Use of XML Namespaces with Attributes

Attributes in XML can also be part of a namespace, which helps distinguish between attributes with the same name but different contexts (vocabularies). Attributes are **not** assumed to be in the same namespace as the element they are applied to. Not only that, but even when declared without a prefix, attributes are not automatically placed even in the default namespace.

The following points elaborate on this in more detail:

1. **Attributes Do Not Inherit the Default Namespace**: Attributes that are declared without a prefix are in no namespace. They are considered to be in a "null" or "empty" namespace, regardless of what namespace the element they apply to belongs to.

<root xmlns="http://example.com/default">

<element attribute="value"/> <!-- 'attribute' is not in any namespace -->

</root>

In this example, `element` is in the `http://example.com/default` namespace, but `attribute` is not. An attribute does not inherit the namespace of its parent element unless it is specifically prefixed to the attribute.

1. **Attributes Must Be Prefixed to Be in a Namespace**: If an attribute is to be associated with a specific namespace, it must be explicitly prefixed with the corresponding namespace prefix.

Example:

<element xmlns:ns=http://example.com/namespace ns:attribute="value"/>

***\*Therefore, if you intend for an attribute to belong to any namespace, then that namespace must have a prefix.***

1. **Unprefixed Attributes are in a "Null" Namespace**: Unprefixed attributes do not belong to any namespace. They are considered to be in a "null" namespace, which is separate from the default or any other explicitly declared namespace.
2. **Namespace Declaration Attributes are not in Any Namespace**: The `xmlns` attribute and any attributes prefixed with `xmlns` (such as `xmlns:prefix`) are used for declaring namespaces and are not considered to be in any namespace themselves. They serve the special function of namespace declaration.
3. **Namespace Consistency**: When using namespaces, it is important to maintain consistency in prefixing attributes that belong to the same namespace across your XML document to avoid confusion and errors.

Using prefixes for attributes helps to avoid naming conflicts, especially when integrating data from different XML sources.