

# Data-centric XML

## The Document Object Model

# DOM Overview

- Developed to support „dynamic HTML“
  - Provide a standard tree interface to document structure across browsers, for use in JavaScript
- Different levels of implementation:
  - Level 1: Flat object model (two features: DOM and HTML)
  - Level 2: API structured into multiple modules
    - Core, XML, HTML, Range, Traversal, ...
    - Focus of this presentation
  - Level 3: New and revised modules
    - new: Load and Save, Validation
    - revised: Core, Events
    - in progress, W3C Notes: XPath, Views and Formatting, ...
- Specified as an API, by means of OMG IDL
  - Programming-language specific mappings for JavaScript, Java as part of the specification
  - Implementations in other languages: C++, Python, C#, ...

# DOM Features

- Core: Represent basic structure of well-formed XML documents
- XML: Access entities, notations, ...
- Events: Communicate user interaction and document changes to the application
  - HTMLEvents, MutationEvents, UIEvents, ...
- Range: Select portions of a document
- Traversal: Process/Filter nodes in sequence
- Views: Access alternative representations of a document
- StyleSheet/CSS: Represent of style sheets
- HTML: Represent HTML documents
- LS, LS-Async: Load and Save

# Object Model

- Distinguish object hierarchy and type hierarchy
  - Node is the root of the type hierarchy
  - Document objects are the root of the object hierarchy
- Polymorphism typically used through introspection:
  - Query Node objects for their type
- Level 3: Mapping to Infoset

# DOM Principles

- Memory management:
  - Creation is always through factory, either on DOMImplementation or Document objects
    - No standard way to obtain a DOMImplementation (in Level 2)
  - Object deletion not specified
- Support both „typed“ and „untyped“ operation:
  - Most interactions available through Node, without need of casting
- Strings are represented as DOMString
  - Implementations must use Unicode type
- Level 3: untyped types DOMUserData (any), DOMObject (Object)

# Node interface: Attributes

```
interface Node {
    readonly attribute DOMString      nodeName;
    attribute DOMString                nodeValue;
    readonly attribute unsigned short  nodeType;
    readonly attribute Node            parentNode;
    readonly attribute NodeList        childNodes;
    readonly attribute Node            firstChild, lastChild;
    readonly attribute Node            previousSibling, nextSibling;
    readonly attribute NamedNodeMap    attributes;
    readonly attribute Document        ownerDocument;
    readonly attribute DOMString       namespaceURI, localName;
    attribute DOMString                prefix;

    // L3
    readonly attribute DOMString       baseURI;
    attribute DOMString                textContent;
```

# Node interface: Operations (1)

```
Node insertBefore(in Node newChild, in Node refChild)
                raises(DOMException);
Node replaceChild(in Node newChild, in Node oldChild)
                raises(DOMException);
Node removeChild(in Node oldChild)
                raises(DOMException);
Node appendChild(in Node newChild)
                raises(DOMException);
void normalize();
```

# Node interface: Operations (2)

Node cloneNode(in boolean deep);  
boolean isSupported(in DOMString feature,  
in DOMString version);  
boolean hasAttributes();



# Node interface: Operations (Level 3)

```
unsigned short compareDocumentPosition(in Node other);  
    // result is a bit mask of DOCUMENT_POSITION_*  
boolean        isSameNode(in Node other);  
DOMString      lookupPrefix(in DOMString namespaceURI);  
boolean        isDefaultNamespace(in DOMString namespaceURI);  
DOMString      lookupNamespaceURI(in DOMString prefix);  
boolean        isEqualNode(in Node arg);  
DOMObject      getFeature(in DOMString feature,  
                           in DOMString version);  
DOMUserData    setData(in DOMString key,  
                       in DOMUserData data,  
                       in UserDataHandler handler);  
DOMUserData    getUserData(in DOMString key);
```

# Node interface: Node Types

const unsigned short	ELEMENT_NODE	= 1;
const unsigned short	ATTRIBUTE_NODE	= 2;
const unsigned short	TEXT_NODE	= 3;
const unsigned short	CDATA_SECTION_NODE	= 4;
const unsigned short	ENTITY_REFERENCE_NODE	= 5;
const unsigned short	ENTITY_NODE	= 6;
const unsigned short	PROCESSING_INSTRUCTION_NODE	= 7;
const unsigned short	COMMENT_NODE	= 8;
const unsigned short	DOCUMENT_NODE	= 9;
const unsigned short	DOCUMENT_TYPE_NODE	= 10;
const unsigned short	DOCUMENT_FRAGMENT_NODE	= 11;
const unsigned short	NOTATION_NODE	= 12;

# Node interface: Document Position

```
const unsigned short DOCUMENT_POSITION_DISCONNECTED = 0x01;  
const unsigned short DOCUMENT_POSITION_PRECEDING   = 0x02;  
const unsigned short DOCUMENT_POSITION_FOLLOWING   = 0x04;  
const unsigned short DOCUMENT_POSITION_CONTAINS    = 0x08;  
const unsigned short DOCUMENT_POSITION_CONTAINED_BY = 0x10;  
const unsigned short DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC = 0x20;
```

# Node Usage: Node Types

- Not all attributes are available for all types
  - E.g. Text nodes do not have children
  - Accessing unsupported attributes raises DOMExceptions
- Node name is always present:
  - „canonical“ name for elements, attributes, entity references, processing instructions
  - „#cdata-section“, „#comment“, „#document“, „#document-fragment“, „#text“ otherwise
- Node value is attribute value, text content, PI content, or comment text; otherwise null

# Node Usage: Namespaces

- namespaceURI is set on creation time; no dynamic lookup is performed
- Changing the prefix also changes the nodeName (and attributeName/name in Attribute/Element nodes)

# Node Usage: Modifications

- Modifications of the child list must follow structural requirements:
  - HIERARCHY\_REQUEST\_ERR if node is not allowed (e.g. inserting parent into child)
  - WRONG\_DOCUMENT\_ERR if node belongs to a different document
  - NO\_MODIFICATION\_ALLOWED if document is read-only
  - NOT\_FOUND\_ERR if refChild is not found (e.g. insertBefore)
- Insertion of DocumentFragment inserts all child nodes
- normalize consolidates subsequent text nodes:
  - Removes empty Text nodes
  - Leaves alone CDATA sections

# Documents

```
interface Document : Node {  
    readonly attribute DocumentType           doctype;  
    readonly attribute DOMImplementation     implementation;  
    readonly attribute Element               documentElement;
```

```
Element               createElement(in DOMString tagName)  
                      raises(DOMException);
```

```
DocumentFragment     createDocumentFragment();
```

```
Text                  createTextNode(in DOMString data);
```

```
Comment              createComment(in DOMString data);
```

```
CDATASection          createCDATASection(in DOMString data)  
                      raises(DOMException);
```

## Documents (2)

ProcessingInstruction	createProcessingInstruction(in DOMString target, in DOMString data) raises(DOMException);
Attr	createAttribute(in DOMString name) raises(DOMException);
EntityReference	createEntityReference(in DOMString name) raises(DOMException);
Element	createElementNS(in DOMString namespaceURI, in DOMString qualifiedName) raises(DOMException);
Attr	createAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName) raises(DOMException);



# Documents (3)

NodeList	<code>getElementsByTagName(in DOMString tagname);</code>
NodeList	<code>getElementsByTagNameNS(in DOMString namespaceURI, in DOMString localName);</code>
Element	<code>getElementById(in DOMString elementId);</code>
Node	<code>importNode(in Node importedNode, in boolean deep) raises(DOMException);</code>

# Documents (Level 3)

```
readonly attribute DOMString          inputEncoding;  
readonly attribute DOMString          xmlEncoding;  
          attribute boolean            xmlStandalone;  
          attribute DOMString         xmlVersion;  
          attribute boolean            strictErrorChecking;  
          attribute DOMString         documentURI;  
Node          adoptNode(in Node source)  
                                   raises(DOMException);  
readonly attribute DOMConfiguration domConfig;  
void              normalizeDocument( );  
Node            renameNode(in Node n,  
                               in DOMString namespaceURI,  
                               in DOMString qualifiedName)  
                                   raises(DOMException);
```

# Document Usage

- doctype: read-only; DOM does not support editing the document type
- documentElement: Convenience attribute
  - Could traverse children for ELEMENT type
- Node creation: Document performs certain consistency checks
  - INVALID\_CHARACTER\_ERR if element/attribute names do not match Name production
  - NAMESPACE\_ERR if names are not QNames, if name is qualified and no namespace provided, if prefix is xml/xmlns, and namespace URI not „http://www.w3.org/XML/1998/namespace“/ „http://www.w3.org/2000/xmlns/“

## Document Usage (2)

- `getElementsByTagName` returns a node list of all elements with a given name
  - „\*“ matches all elements
- `getElementById` looks for an ID attribute
  - Must have processed DTD to find ID attributes
  - Returns null if no element was found
- `importNode` allows migration of nodes from one document to another
  - Level 3: `adoptNode` tries to avoid copying
- `normalizeDocument` emulates save-load-cycle
-

# Elements

- Convenience interface

```
interface Element : Node {
    readonly attribute DOMString      tagName;
    DOMString      getAttribute(in DOMString name);
    void           setAttribute(in DOMString name, in DOMString value)
                    raises(DOMException);

    void          removeAttribute(in DOMString name) raises(DOMException);
    Attr          getAttributeNode(in DOMString name);
    Attr          setAttributeNode(in Attr newAttr) raises(DOMException);
    Attr          removeAttributeNode(in Attr oldAttr) raises(DOMException);
    NodeList      getElementsByTagName(in DOMString name);
    boolean       hasAttribute(in DOMString name);
}
```

## Elements (2)

DOMString    getAttributeNS(in DOMString namespaceURI,  
                  in DOMString localName);

void         setAttributeNS(in DOMString namespaceURI,  
                  in DOMString qualifiedName, in DOMString value)  
                  raises(DOMException);

void         removeAttributeNS(in DOMString namespaceURI,  
                  in DOMString localName) raises(DOMException);

Attr         getAttributeNodeNS(in DOMString namespaceURI,  
                  in DOMString localName);

Attr         setAttributeNodeNS(in Attr newAttr) raises(DOMException);

NodeList    getElementsByTagNameNS(in DOMString namespaceURI,  
                  in DOMString localName);

boolean     hasAttributeNS(in DOMString namespaceURI,  
                  in DOMString localName);

# Elements (Level 3)

```
readonly attribute TypeInfo          schemaTypeInfo;  
void          setIdAttribute(in DOMString name,  
                                in boolean isId)  
                                raises(DOMException);  
void          setIdAttributeNS(in DOMString namespaceURI,  
                                in DOMString localName,  
                                in boolean isId)  
                                raises(DOMException);  
void          setIdAttributeNode(in Attr idAttr,  
                                in boolean isId)  
                                raises(DOMException);
```

# Text

- Several interfaces

- CharacterData : Node

- Abstract interface
    - attribute DOMString data;
    - Several modification operations

- Text : CharacterData

- splitText separates text node into two nodes
    - Level 3: wholeText, isElementContentWhitespace, replaceWholeText

- CDataSection : Text

- Empty interface



# Attributes

```
interface Attr : Node {  
    readonly attribute DOMString    name;  
    readonly attribute boolean      specified;  
    attribute DOMString            value;  
    readonly attribute Element      ownerElement;  
    // Level 3  
    readonly attribute TypeInfo      schemaTypeInfo;  
    readonly attribute boolean      isId;  
};
```

- Attribute nodes are not part of the tree
  - parentNode, previousSibling, nextSibling are all null

# Live Objects

- Collections whose contents changes when underlying tree changes
- NodeList: enumerated collection
  - E.g. of child nodes, getElementByTagName results
  - Indices start with 0
- NamedNodeMap: unordered collections of named things
  - E.g. attributes of a node, entities, notations of a DTD
  - Access either by QName, or namespaceURI/localname
  - Indexed access (starting with 0) also supported

# DOM Implementations

```
interface DOMImplementation {  
    boolean          hasFeature(in DOMString feature,  
                               in DOMString version);  
  
    DocumentType    createDocumentType(in DOMString qualifiedName,  
                                       in DOMString publicId, in DOMString systemId)  
                                       raises(DOMException);  
  
    Document        createDocument(in DOMString namespaceURI,  
                                   in DOMString qualifiedName,  
                                   in DocumentType doctype)  
                                   raises(DOMException);  
  
    DOMObject       getFeature(in DOMString feature, in DOMString version);  
};
```

- Feature names are the DOM module names („Core“, „XML“, ...)

# DOM Parsing in Java

- `javax.xml.parsers.DocumentBuilderFactory`
- Underlying parser is assumed to follow SAX API

```
dbf = DocumentBuilderFactory.newInstance();
dbf.setValidating(true);
dbf.setNamespaceAware(true);
dbf.setIgnoringComments(true);
...
db = dbf.newDocumentBuilder();
db.setEntityResolver(myResolver)
document = db.parse(File/URL/InputStream/InputStream)
```

# Bootstrapping with DOM Level 3

- predefined object `DOMImplementationRegistry`
  - `getDOMImplementation`(in `DOMString` features) searches for an appropriate implementation
  - maintains list of objects implementing `DOMImplementationSource`
  - Java: `org.w3c.dom.bootstrap.DOMImplementationRegistry`

# DOM Parsing with Level 3

```
interface DOMImplementationLS {  
    const unsigned short    MODE_SYNCHRONOUS        = 1;  
    const unsigned short    MODE_ASYNCHRONOUS      = 2;  
  
    LSParser                createLSParser(in unsigned short mode,  
                                           in DOMString schemaType)  
                            raises(dom::DOMException);  
  
    LSSerializer            createLSSerializer();  
    LSInput                 createLSInput();  
    LSOutput                createLSOutput();  
};
```

# DOM Parsing with Level 3 (2)

```
interface LSParser {  
    readonly attribute DOMConfiguration config;  
    attribute LSParserFilter filter;  
  
    readonly attribute boolean async;  
    readonly attribute boolean busy;  
  
    Document parse(in LSInput input) raises(dom::DOMException);  
    Document parseURI(in DOMString uri)raises(dom::DOMException);  
    Node parseWithContext(in LSInput input,  
                          in Node contextArg,  
                          in unsigned short action)  
                      raises(dom::DOMException);  
  
    void abort();  
};
```

# DOM Parsing with Level 3 (3)

```
interface DOMConfiguration {  
    void                setParameter(in DOMString name,  
                                     in DOMUserData value)  
                                     raises(DOMException);  
    DOMUserData         getParameter(in DOMString name)  
                                     raises(DOMException);  
    boolean              canSetParameter(in DOMString name,  
                                         in DOMUserData value);  
    readonly attribute DOMStringList  parameterNames;  
};
```



# DOM Parsing with Level 3 (3)

- Standard parser parameters
  - canonical-form
  - cdata-sections
  - check-character-normalization
  - comments
  - datatype-normalization
  - entities
  - namespaces
  - namespace-declarations
  - normalize-characters
  - validate
  - validate-if-schema
  - element-content-whitespace
  - ...

# DOM Parsing with Level 3

```
System.setProperty(DOMImplementationRegistry.PROPERTY,  
    "org.apache.xerces.dom.DOMXSImplementationSourceImpl");
```

```
DOMImplementationRegistry reg = DOMImplementationRegistry.newInstance();
```

```
DOMImplementation di = reg.getDOMImplementation("LS");
```

```
DOMImplementationLS dils = (DOMImplementationLS) di.getFeature("LS", null);
```

```
LSParser parser =
```

```
dils.createLSParser(DOMImplementationLS.MODE_SYNCHRONOUS, null);
```

```
Document d = parser.parseURI("file:///c:/temp/foo.xml");
```

```
System.out.println(d.getDocumentElement().getNodeName());
```

# DOM Saving Trees in Java

- Based on the notion of transformations

- Input is a DOM tree, output is a stream

```
TransformerFactory tFactory = TransformerFactory.newInstance();
```

```
Transformer transformer = tFactory.newTransformer();
```

```
DOMSource source = new DOMSource(document);
```

```
StreamResult result = new StreamResult(System.out);
```

```
transformer.transform(source, result);
```

# Saving trees with Level 3

- interface LSSerializer: inverse of SAX InputSource
  - attributes characterStream, byteStream, systemId, encoding
- interface LSSerializer
  - operations write, writeToUri, writeToStream
    - each expects a node
    - target is either LSSerializer, DOMString (Uri), or result string
  - various configuration parameters
    - canonical-form
    - discard-default-content
    - format-pretty-print
    - ignore-unknown-character-denormalizations
    - normalize-characters
    - xml-declaration