VTD-XML Introduction and API Overview

XimpleWare

info@ximpleware.com

2/2008

Agenda

- Motivations Behind VTD-XML
- Why VTD-XML?
- When to Use VTD-XML?
- Basic Concept
- Essential Classes and Methods
- VTD-XML in C and C#
- Summary

Motivations Behind VTD-XML

- *Numerous*, well-known issues of old XML processing models, below summarizes a few:
  - DOM: Too slow and resource intensive
  - SAX: Forward only; treat XML as CSV; performance/memory benefits insufficient to justify its difficulty
  - Pull: Only programming style change; inherit most of the problems from SAX
- Enterprise developers have no other via options

Why VTD-XML?

- The next generation XML processing model that is simultaneously:
  - The world's ***fastest*** XML parser (1.5x~3x of SAX with null content handler)
  - The world's ***most memory efficient***, ***random-access-capable*** XML parser (1.3x~1.5x size of the XML document)
  - The world's first XML parser supporting ***incremental update***
  - The world's first XML parser with ***built-in indexing feature*** **(**aka. **VTD+XML)**
  - The world's first XML parser that is portable to ***ASIC***
  - The world's first XML parser with ***built-in buffer reuse*** feature

When to Use VTD-XML?

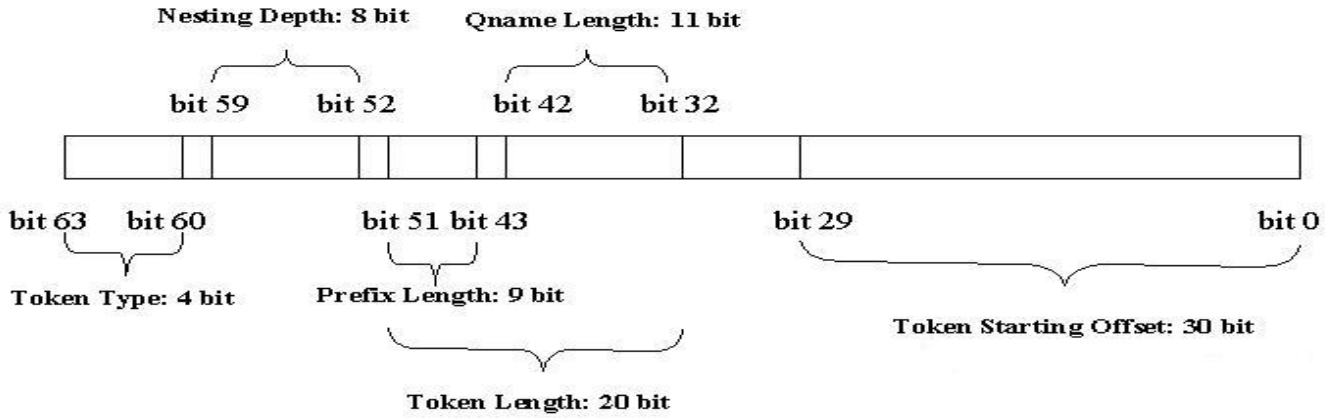
- The scenarios that you may consider using VTD-XML
  - Large XML files that DOM can't handle
  - Performance-critical transactional Web-Services/SOA applications
  - Native XML database applications
  - Network-based XML content switching/routing/security applications

Known Limitations

- Not yet support external entities (those declared within DTD)
- Not yet process DTD (return as a single VTD record)
- Schema validation feature is planned for a future release.
- Extreme long (>=512 chars) element/attribute names or ultra deep document (>= 255 levels) will cause parse exception

Basic Concept

- Non-extractive tokenization based on Virtual Token Descriptor (VTD): use 64-bit integers to encode offsets, lengths, token types, depths



- The XML document is kept intact and un-decoded.

Basic Concept

- In other words, in vast majority of the cases string allocation is *unnecessary*, and nothing but a waste of CPU and memory
- VTD-XML performs many string operations directly on VTD records
  - String to VTD record comparison (both boolean and lexicographically)
  - Direct conversions from VTD records to ints, longs, floats and doubles
  - VTD record to String conversion also provided, but avoid them whenever possible for performance reasons

Basic Concept

- VTD-XML's document hierarchy consists *exclusively* of elements
- Move a single, global cursor to different locations in the document tree
- Many VTDNav's methods identify a VTD record with its index value
- -1 corresponds to "no such record"

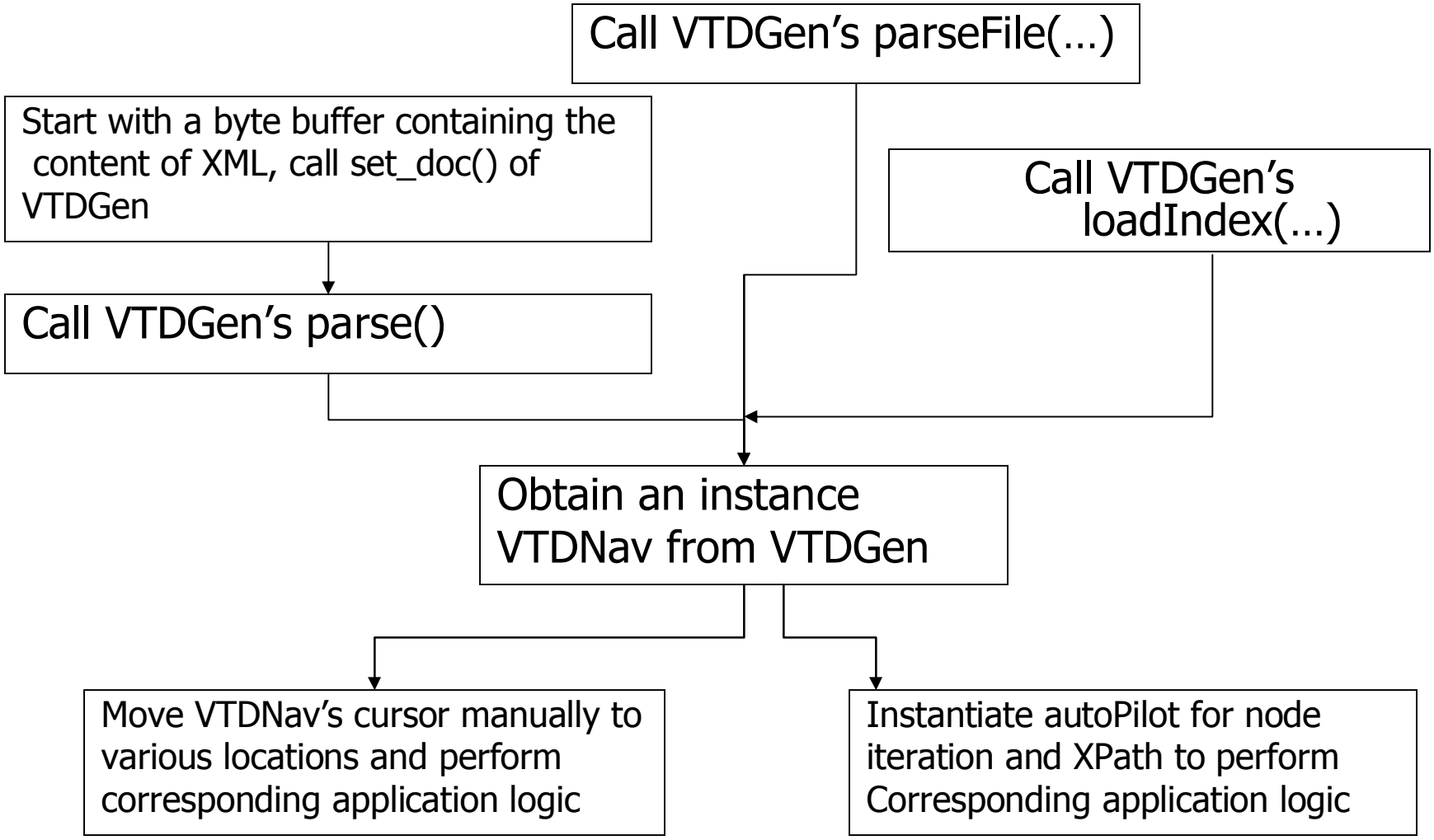
Essential Classes

- **VTDGen**: Encapsulates the parsing, indexing routines
- **VTDNav**: VTD navigator allows cursor-based random access and various functions operating on VTD records
- **AutoPilot**: Contains XPath and Node iteration functions
- **XMLModifier**: Incrementally update XML

Essential Classes

- Exceptions
  - **ParseException**: Thrown during parsing when XML is not well-formed
  - **IndexingReadException**: Thrown by VTDGen when there is error in loading index
  - **IndexingWriteException**: Thrown by VTDGen when there is error writing index
  - **NavException**: Thrown when there is an exception condition when navigating VTD records
  - **PilotException**: Child class of NavException; thrown when using autoPilot to perform node iteration.
  - **XPathParseException**: Thrown by autoPilot when compiling an XPath expression
  - **XPathEvalException**: Thrown by autoPilot when evaluating an XPath expression
  - **ModifyException**: Thrown by XMLModifier when updating XML file

Typical Programming Flows



Methods of VTDGen

- *void* **setDoc** (***byte[]*** *ba*)**:** Pass the byte buffer containing the XML document
- *void* **setDoc_BR** (***byte[]*** *ba*)**:** Pass the byte buffer containing the XML document, with Buffer Reuse feature turned on.
- *void* **setDoc** (***byte[]*** *ba,* ***int*** *offset,* ***int*** *length*)**:** Pass the byte buffer containing the XML document, offset and length further specify the start and end of the XML document in the buffer
- *void* **setDoc_BR** (***byte[]*** *ba,* ***int*** *offset,* ***int*** *length*)**:** Pass the byte buffer containing the XML document, offset and length further specify the start and end of the XML document in the buffer, with Buffer Reuse feature turned on

Methods of VTDGen

- *void* **parse():** The main parsing function, internally generates VTD records, etc.
- *boolean* **parseFile(*String* fileName, *boolean* ns):** Directly parse an XML file of the given name
- *boolean* **parseHttpUrl(*String* fileName, *boolean* ns):** Directly parse an XML file of the given name
- *VTDNav* **getNav():** If **parse**() or **parseFile**(…) succeed, this method returns an instance of VTDNav
- *void* **clear():** Clear the internal state of VTDGen . This method is called internally by **getNav**(); call this method explicitly between successive **parse**()

Methods of VTDGen

- *VTDNav* **loadIndex(InputStream is):** Load index from input stream
- *VTDNav* **loadIndex(*String* fileName):** Load index from a file (recommended extension vxl)
- *VTDNav* **loadIndex(byte[] ba):** If parse() or parseFile(...) succeed, this method returns an instance of VTDNav
- *void* **writeIndex(OutputStream os):** Write the index into output stream
- *void* **writeIndex**(**String fileName**): Write index into a file
- *long* **getIndexSize**(): Pre-compute the size of VTD+XML index

Methods of VTDNav

- **The main navigation functions that moves the global cursor:**
  - boolean **toElement** (***int*** *direction*)
  - boolean **toElement** (***int*** *direction,* ***String*** *elementName*)
  - boolean **toElementNS** (***int*** *direction,* ***String*** *URL,* ***String*** *localName*)
  - "Direction" takes one of the following constants (self-explanatory): PARENT, ROOT, FIRST_CHILD, LAST_CHILD, FIRST_SIBLING, LAST_SIBLING

Methods of VTDNav

- **Attribute lookup methods for the element at the cursor position**
  - *int* **getAttrVal** (***String*** *attrName*)
  - *int* **getAttrValNS** (***String*** *URL,* ***String*** *localName*)
  - *int* **getAttrCount()**: Return the attribute count of the element at the cursor position.
- **Attribute Existence Test for the element at the cursor position**
  - *boolean* **hasAttr** (***String*** *attrName*)
  - *boolean* **hasAttrNS** (***String*** *URL,* ***String*** *localName*)

Methods of VTDNav

- **Retrieve Text Node**
  - *int* **getText(): ** Returns the index value of the VTD record corresponding to character data or CDATA
  - More sophisticated retrieval, such as mixed content, available in **TextIter** class

Methods of VTDNav

- **VTD to String boolean comparison functions**
  - *boolean* **matchElement** (***String*** *en*)**:** Test if the current element matches the given name.
  - *boolean* **matchElementNS** (***String*** *URL,* ***String*** *localName*): Test whether the current element matches the given namespace URL and localName.
  - *boolean* **matchRawTokenString** (***int*** *index,* ***String*** *s*): Match the string against the token at the given index value.
  - *boolean* **matchTokens** (***int*** *i1,* ***VTDNav*** ***v****n2,* ***int*** *i2*): This method compares two VTD records of VTDNav objects
  - *boolean* **matchTokenString** ***(int*** *index,* ***String*** *s****):*** Match the string against the token at the given index value.

Methods of VTDNav

- **VTD to String lexical comparison functions**
  - *int* **compareRawTokenString** (***int*** *index,* ***String*** *s*): Compare the token at the given index value against a string (returns 1,0, or -1).
  - *int* **compareTokens** (***int*** *i1,* ***VTDNav*** ***v****n2,* ***int*** *i2*): This method compares two VTD records of VTDNav objects (returns 1, 0, or -1).
  - *boolean* **compareTokenString** ***(int*** *index,* ***String*** *s):* Compare the token at the given index value against a string.

Methods of VTDNav

- **Query cursor attributes**
  - *int* **getCurrentDepth(): ** Get the depth (>=0) of the element at the cursor position
  - *int* **getCurrentIndex(): ** Get the index value of the element at the cursor position.
  - *long* **getElementFragment(): ** Get the starting offset and length of an element encoded in a long, upper 32 bit is length; lower 32 bit is offset; Unit is in bytes.

Methods of VTDNav

- **VTD to other data types conversions**
  - *double* **parseDouble** (***int*** *index*): Convert a VTD record into a double.
  - *float* **parseFloat** (***int*** *index*): Convert a VTD record into a float.
  - *int* **parseInt** (***int*** *index*): Convert a VTD record into an int.
  - *long* **parseLong** ***(int*** *index)**:*** Convert a VTD record into a long.

Methods of VTDNav

- **Convert VTD records into Strings**
  - *String* **toNormalizedString** ***(int*** *index)**:*** This method normalizes a token into a string in a way that resembles DOM: starting and ending white spaces are stripped, and successive white spaces in the middleware are collapsed into a single space char
  - *String* **toRawString** ***(int*** *index)**:*** Convert a token at the given index to a String, (built-in entity and char references not resolved) (entities and char references not expanded).
  - *String* **toString** ***(int*** *index)**:*** Convert a token at the given index to a String, (entities and char references resolved).

Methods of VTDNav

- **Querying attributes of an VTD record**
  - *int* **getTokenDepth** ***(int** index**):*** Get the depth value of a token (>=0).
  - *int* **getTokenLength** ***(int** index**):*** Get the token length at the given index value please refer to VTD spec for more details. Length is in terms of the UTF char unit. For prefixed tokens, it is the qualified name length.
  - *int* **getTokenOffset** ***(int** index**):*** Get the starting offset of the token at the given index.
  - *int* **getTokenType** ***(int** index**):*** Get the token type of the token at the given index value.

Methods of VTDNav

- **Access the global stack**
  - *void* **push**(): push the cursor position into the global
  - *boolean* **pop**(): Load the saved cursor position
- **To cache/save cursor positions for later sequential access, use NodeRecorder class**

Methods of VTDNav

- Query the attributes of parsed XML
  - *int* **getEncoding():** Get the encoding of the XML document.
  - *int* **getNestingLevel():** Get the maximum nesting depth of the XML document (>0).
  - *int* **getRootIndex():** Get root index value , which is the index value of document element
  - *int* **getTokenCount():** Get total number of VTD tokens for the current XML document.
  - *IByteBuffer* **getXML():** Get the XML document

Methods of VTDNav

- Writing VTD+XML Index
  - *void* **writeIndex(OutputStream os):** Write the index into output stream
  - *void* **writeIndex**(**String fileName**): Write index into a file
  - *long* **getIndexSize**(): Pre-compute the size of VTD+XML index

Methods of AutoPilot

- **Constructors**
  - **AutoPilot** ***(VTDNav v):*** AutoPilot constructor comment.
  - **AutoPilot** **():** Use this constructor for delayed binding to VTDNav which allows the reuse of XPath expression
- **Bind VTDNav object to AutoPilot**
  - void **bind**(VTDNav vn): It resets the internal state of AutoPilot so one can attach a VTDNav object to the autoPilot

Methods of AutoPilot

- **XPath Related**
  - *void* **declareXPathNameSpace** **(*String* *prefix*, *String* *URL*):** This function creates URL ns prefix and is intended to be called prior to selectXPath
  - *void* **selectXPath** **(*String* *s*):** This method selects the string representing XPath expression Usually evalXPath is called afterwards
  - *String* **getExprString** **():** Convert the expression to a string For debugging purpose
  - *void* **resetXPath** **():** Reset the XPath so the XPath Expression can be reused and revaluated in anther context position

Methods of AutoPilot

- **XPath Related**
  - *int* **evalXPath ():** This method moves to the next node in the nodeset and returns corresponding VTD index value. It returns -1 if there is no more node After finishing evaluating, don't forget to *reset the xpath*
  - *double* **evalXPathToNumber ():** This function evaluates an XPath expression to a double
  - *String* **evalXPathToString ():** This method returns XPath expression to a String
  - *String* **evalXPathToBoolean ():** This method evaluates an XPath expression to a boolean

Methods of AutoPilot

- **Emulate DOM's Node Iterator**
  - void **selectElement** (String en): Select the element name before iterating.
  - void **selectElementNS** (String URL, String localName): Select the element name (name space version) before iterating.
  - boolean **iterate** (): Iterate over all the selected element nodes in document order.

Methods of XMLModifier

- **Constructors**
  - **XMLModifier(*VTDNav v*):** XMLModifier constructor that binds VTDNav directly.
  - **XMLModifier():** Use this constructor for delayed binding to VTDNav
- **Bind VTDNav object to XMLModifier**
  - void **bind**(VTDNav vn): It resets the internal state of AutoPilot so one can attach a VTDNav object to the XMLModifier

Methods of XMLModifier

- **Remove from the XML document**
  - void **remove** (): Remove whatever that is pointed to by the cursor
  - void **removeAttribute**(int attrNameIndex): Remove an attribute name/value pair as referenced by the attrNameIndex.
  - boolean **removeToken**(int i): Remove the token at the index position
  - boolean **removeContent**(int offset, int len): Remove a segment of byte content from master XML doc.

Methods of XMLModifier

- **Insert into an XML document**
  - void **insertAfterElement**(byte[] b)— This method inserts the byte array b after the cursor element
  - void **insertAfterElement**(String s)— This method inserts the byte value of s after the element
  - void **insertBeforeElement**(byte[] b)— Insert a byte array before the cursor element
  - void **insertBeforeElement**(String attr)— Insert a String before the cursor element

Methods of XMLModifier

- **Insert into an XML document**
  - **void insertAfterElement**(int src_encoding, byte[] b) Insert a byte array of given encoding into the master document.
  - **void insertAfterElement**(int src_encoding, byte[] b, int contentOffset, int contentLen) Insert the transcoded array of bytes of a segment of the byte array b after the element
  - void **insertBeforeElement**(int src_encoding, byte[] b)
    Insert insert the transcoded representatin of the byte array b before the cursor element
  - **void insertBeforeElement**(int src_encoding, byte[] b, int contentOffset, int contentLen) Insert the transcoded representation of a segment of the byte array b before the cursor element.

Methods of XMLModifier

- **Insert into an XML document**
  - void **insertAfterElement**(byte[] b, int contentOffset, int contentLen)— This method inserts a segment of the byte array b after the cursor element
  - void **insertBeforeElement**(byte[] b, int contentOffset, int contentLen)— Insert the segment of a byte array before the cursor element
  - void **insertAfterElement**(ElementFragmentNs ef)— Insert a namespace compensated element after the cursor element
  - void **insertBeforeElement**(ElementFragmentNs ef)— Insert a namespace compensated element before the cursor element

Methods of XMLModifier

- **Insert into XML document**
  - void **insertAttribute**(byte[] b): Insert the byte array representation of attribute name/value pair after the starting tag of the cursor element
  - void **insertAttribute**(String attr): Insert the String representation of attribute name/value pair after the starting tag of the cursor element
  - void **insertBytesAt**(int offset, byte[] content)
    insert the byte content into XML
  - void **insertBytesAt**(int offset, byte[] content, int contentOffset, int contentLen)
    Insert a segment of the byte content into XML

Methods of XMLModifier

- **Update a token in XML**
  - void **updateToken**(int i, byte[] b): Replace the token (of index i) with the byte content of b
  - void **updateToken**(int i, String newContent): Replace the token (of index i) with the byte content of String value
  - void **updateToken**(int index, byte[] newContentBytes, int src_encoding) Update the token with the transcoded representation of given byte array content
  - void **updateToken**(int index, byte[] newContentBytes, int contentOffset, int contentLen, int src_encoding) Update token with the transcoded representation of a segment of byte array (in terms of offset and length)

Methods of XMLModifier

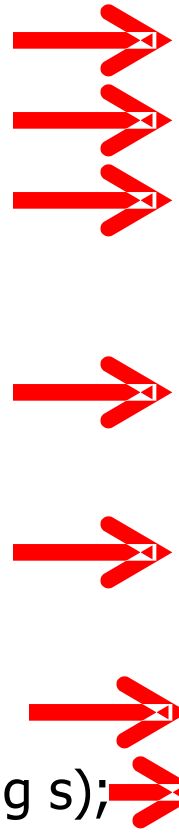
- **Generate Output**
  - void **output**(OutputStream os): Replace the token (of index i) with the byte content of b
  - Void **output**(java.lang.String fileName)
    Generate the updated output XML document and write it into a file of given name
- **Reset XMLModifier for reuse**
  - void **reset**(): Replace the token (of index i) with the byte content of String value
- **Other methods**
  - int **getUpdatedDocumentSize**(): Compute the size of the updated XML document without composing it

VTD-XML in C

- Compared to Java, C is different in the following aspects:
  - No notion of class
  - No notion of constructor
  - No automatic garbage collection
  - No method/constructor overloading
  - No exception handling
- VTD-XML's C version uses the following tactics:
  - Use struct pointer
  - Explicit call "create…" functions
  - Explicit call "free…" functions
  - Pre-pending integer to functions name to differentiate
  - Use <cexcept.h> to provide basic try catch in C

Java Methods vs. C Functions

- VTDGen vg = VTDGen();
- Auto garbage collector
- void setDoc(byte[] ba)
- void setDoc(byte[] ba, int docOffset, int docLen);
- void parse (boolean ns)
- int getTokenCount()
- boolean matchElement(String s);



- VTDGen *vg= createVTDGen();
- void freeVTDGen (vg);
- void setDoc(VTDGen *vg, UByte* ba, int arrayLength);
- void setDoc2(VTDGen *vg, UByte *ba, int arrayLen, int docOffset, int docLen);
- parse(VTDGen *vg, boolean ns)
- int getTokenCount(VTDNav *vn)
- Boolean matchElement(VTDNav *vn, UCSChar *s);

Exception Handling: Java vs. C

```
public static void main(String argv[]){
  try {
     // put the code throwing
      //exceptions  here
  } catch (Exception e){
     // handle exception in here
  }
}
```

```
// set up global exception context
struct exception_context
    the_exception_context[1];
int main(){
   // declare exception
   exception e;
   Try {
    // put the code throwing
    // exceptions here
   } Catch (e) {
      // handle exception in here
   }
}
```

VTD-XML in C#

- Compared to Java, C# is very similar, so the Java code looks and feels the same as the C# code.

Summary

- This presentation provides the basic introduction and API overview for VTD-XML
- Any questions or suggestions? Join our discussion group
- Want to get involved? Having a good idea extending VTD-XML? Write to us: info@ximpleware.com