

JDF Open Source Application Programming Interface



HEIDELBERG

Dr. Rainer Prosi

Reasons for an open source JDF API



- JDF uses XML as a Database, not as a Markup Language
 - lots of ID-IDREF linking
 - Inheritance by nesting
 - Enhanced Validation
- Use of the same code base reduces incompatibility between vendors
 - Compile-time code validation through type safe classes instead of generic string based calls
 - Same spec interpretation through a common high level code base
- Providing an open source API reduces barriers to adapting a JDF workflow

Technical Features I

- Platform independent
 - Mapping of new interfaces to JAVA interface
 - e.g. HTTP handler
 - Platform abstraction layer
 - Reliance on STL
 - 1. developers pre-release Windows VC 6
 - No MFC or windows OS calls

Technical Features II

- JDF requires an object tree in memory in order to navigate links / references
 - ==> Use the **DOM** implementation
- (C++) Standard Template Library - STL is heavily used
 - Strings
 - Vectors
 - Maps

Components of the JDF API - Parser



- Xerces based XML Parser Wrapper
 - Type-safe Data access
 - High-level JDF Feature Support
 - Partitioned Resource abstraction (nesting)
 - Spawning and Merging
 - Reference Inlining
 - Enhanced Validation
 - Utilities
 - I/O handling
 - NodeList -> STL::Vector conversion

Components of the JDF API - Utilities



- Non-XML Utility Classes
 - I/O abstraction
 - Mime Messaging Class
 - PNG Read / Write support
 - URL Support
 - Protocol Handlers
 - HTTP
 - HTTPS
 - FILE
 - CID

Excluded from the Open Source JDF API

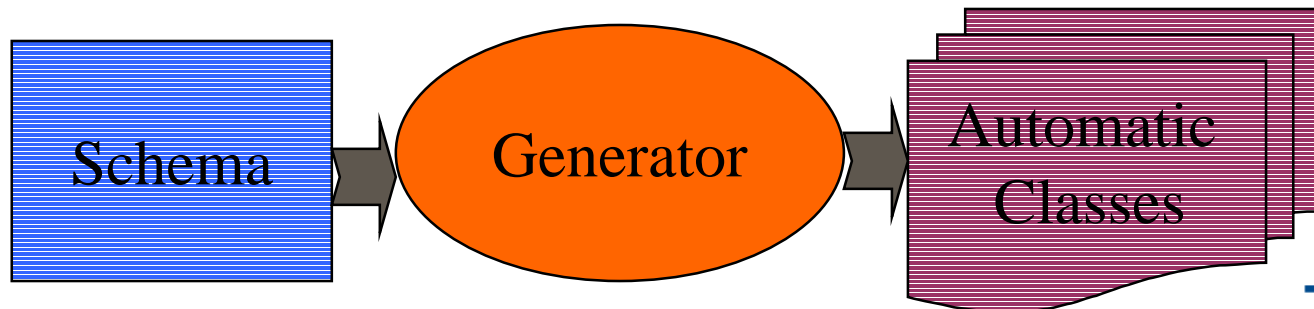


- Excluded because of additional proprietary code that is required.
 - PJTF / PPF to JDF translation
 - JDF to PJTF / PPF
 - Other translators
 - Other Low-level application utilities
 - Database mapping
 - Application infrastructure
- Proprietary Value Add

Type-Safe Code Generator



- Schema still has limits:
 - requires element ordering
 - nesting support not available
 - API validation is always stronger than Schema validation
- Generator written in Java
- Based on standard schema with additional meta-data



JDF Parser Class Layers

	DOM_Element: Xerces
	KElement: STL String wrapper
Auto Manual	JDFElement: First JDF specific layer
	YourElement: empty class, provided for private extensions
Auto Manual	JDFResource: Resource Abstraction layer
	YourResource: empty class, provided for private extensions
Auto Manual	JDFPart: Partition Abstraction layer
	YourPart: empty class, provided for private extensions
Auto Manual	JDFRunList: RunList Abstraction layer

Example Abstraction

```
<RunList ID="1">
  <Runlist Npage="1"
    FirstPage="0"
    Separation="Cyan">
    <LayoutElement>
      <FileSpec
        URL="File1.pdf"/>
    </LayoutElement>
  </Runlist>
  <Runlist Npage="1"
    FirstPage="1"
    Separation="Magenta">
    <LayoutElement>
      <FileSpec
        URL="File1.pdf"/>
    </LayoutElement>
  </Runlist>
</RunList>
```

```
<RunList ID="1" Npage="1">
  <LayoutElement>
    <FileSpecRef rRef="2"/>
  </LayoutElement>
  <Runlist FirstPage="0"
    Separation="Cyan">
  </RunList>
  <Runlist FirstPage="1"
    Separation="Magenta">
  </RunList>
</RunList>
<FileSpec ID="2"
  URL="File1.pdf"/>
```

Enhanced Validation

■ Schema

- ID - IDRef
- nested elements must be optional
- Valid / Invalid
- Automation defines limits

■ Generated code

- Reference Type Check
- required elements must exist in the nested structure
- Valid / Invalid / Incomplete
- Optional additional hand-coding allows for further sanity checks



Status of the API

- Submitted to the CIP4 Open Source working group for review
 - Heidelberg: JDF Class Wrapper
 - AGFA: Utility libraries
- After review and some rounds of bug fixes:
 - Open Source
 - Free Commercial Use License for CIP4 full + partner members
- Java Version is under consideration