NIEM Implementation Cookbook

Contents

Overview

A catalog of techniques and methods that help to implement NIEM. Find what you need. Share what you can.

What's this all about?

Well, when you're creating an information exchange, you rarely ever use NIEM by itself. Depending on business requirements, there are other aspects of information exchange implementations that are required, including access controls, policy automation, transmission protocol, and others.

Based on our NIEM implementation experience, we've identified some common patterns (or "recipes," so to speak) to implement NIEM with other pieces of the information exchange puzzle.

The implementation cookbook serves as a single location for the community to learn and share reusable solutions for implementing NIEM.

With your help, our goal is for the cookbook to define the many ways NIEM-based exchanges can be used within specific environments, with various technologies and standards. This includes programming languages, data formats, query languages, and messaging frameworks.

Remember, NIEM and other data standards are independent of any particular software. However, ensuring NIEM is expressed, defined, and used in various software platforms better enables reusability, portability, and interoperability—which is what this effort is working toward.

At the end of the day, each implementation recipe produces consumable, access controlled data. We encourage you to contribute to the open data conversation through groups like **18F**.

Topics include:

- Size of NIEM XML documents
- Enterprise Integration Patterns
- Browsing the content of NIEM

Avoid replicating data in a NIEM XML document

There are several NIEM-specific strategies to avoid replicating data within a NIEM XML document.

Use structures:id and structures:ref

An XML document may avoid replicating data by using references within the document. Instead of replicating the data for a vehicle, for example, the vehicle may be expressed once within an instance document, and other uses of the vehicle may be expressed by a reference to the vehicle. The use of references is described and specified by the NIEM Naming and Design Rules section on reference elements.

Take, for example, this example from the NDR:

```
<nc:Item>
  <nc:ItemOwner structures:ref="m82"/>
</nc:Item>
<nc:Entity structures:id="m82">
  <nc:EntityPerson>
    <nc:PersonName>
      <nc:PersonFullName>John Doe</nc:PersonFullName>
    </nc:PersonName>
  </nc:EntityPerson>
</nc:Entity>
```

By default, a NIEM subset, any element may either have immediate content, or may identify that it is a reference to the content. A reference must be to another element that has the same type as the referring type, or to a type derived from the referring type.

Note that there is no difference in meaning between XML data that uses a reference and the same data that uses content.

Use NIEM metadata

NIEM defines a mechanism for expressing data about data: NIEM metadata. NIEM metadata is defined by the NDR section "Instance metadata". NIEM defines two kinds of metadata:

1. Metadata on objects and associations: this enables a NIEM message to say information about where information about an object comes from: its source, validity, privacy, etc. Metadata on an object or an association is defined via the `structures:metadata` attribute.
2. Metadata on relationships: this enables a NIEM message to provide information about relationships established by elements in the message. Metadata on a relationship is established via the `structures:relationshipMetdata` attribute.

The NIEM section on instance metadata provides examples of metadata in use. These mechanisms may enable duplicated data (e.g., source of a statement) to be put into one place, and to be referenced from throughout the XML document.

Use NIEM augmentations

A NIEM augmentation is an element that defines additional relationships or characteristics of an object through a specific kind of extension. An augmentation definition enables a domain to define an extension to an object without having to define a new derived type. It can also be used within an IEPD to define new properties on an already-defined type.

A NIEM augmentation type acts as a bucket of relationships and characteristics that may be applied to a variety of objects. If an instance has a set of objects that share a set of characteristics, it may be briefer to express those characteristics via an augmentation type, and to reference a single instance of the augmentation type from all of the objects.

Browsing the content of NIEM

NIEM is built in XML Schema, which means that almost any tool that may be used to browse XML Schema may be used with NIEM. Schemas may be obtained from release.niem.gov.

There are several online resources that may help you find out what is in NIEM:

- The NIEM Schema Subset Generation Tool (SSGT), which you may use to browse the NIEM releases, as well as to build subset schemas suitable for use in IEPDs.
- The NIEM Wayfarer.
- Schema Central.
- The NIEM model content overview.

CONTRIBUTING

The National Information Exchange Model (NIEM) Program GitHub page welcomes your comments and postings. Comments posted to the NIEM GitHub page are subject to GitHub's usage, abuse, and comment policies at https://help.github.com/articles/github-terms-of-service/. Your comments are public and available to anyone visiting the NIEM GitHub page.

To protect your privacy and the privacy of others, do not include your full name, phone numbers, email addresses, social security number, case numbers or any other sensitive personally identifiable information (PII) in your comments or responses.

NIEM does not moderate comments on the NIEM GitHub page prior to posting, but reserves the right to remove any materials that pose a security or privacy risk.

Efficient XML Interchange

Efficient XML Interchange (EXI) uses information in the XML Schema for an XML document to optimize compression of XML documents. Systems using EXI may program against a SAX interface, and character information items will not be decompressed until needed, which can greatly speed processing of XML document. Together, EXI can be very effective at speeding up XML transmission and processing.

The EXI specification is defined by W3C's Efficient XML Interchange Working Group. The [[EXI specification (Efficient XML Interchange (EXI) Format 1.0 (Second Edition))|http://www.w3.org/TR/exi/]] is a W3C recommendation.

Implementations

W3C maintains a list of implementations of EXI, which includes:

- EXIficient, an open source Java implementation. This is provided under the GPL v2 license.
- AgileDelta's Efficient XML™, a commercial implementation.

Enterprise Integration Patterns

Enterprise Integration Patterns (EIP) provide a comprehensive architecture for manipulating NIEM messages. It provides a way of thinking about and expressing integration between and across systems, languages, APIs, and databases. These tools can help NIEM developers to accept data from various sources, transform the data to and from NIEM-conformant formats, and perform operations across data.

Enterprise Integration Patterns (EIP) is a book, written by Gregor Hohpe and Bobby Woolf, and first published in 2003. It defines 65 patterns for integration of applications and interfaces. These patterns are implemented by a large number of middleware systems and software.

Enterprise Integration Patterns (EIP) provides a language that developers can use to conceive and implement complex asynchronous messaging systems. It enables developers to break down a complex problem into small, simple steps using well-defined building blocks. Each building block is an EIP, and provides functionality that interacts with a message, database, or endpoint in some fashion. The base patterns are:

- Messaging: enables communication between components via asynchronous messages, rather than round-trip APIs or shared databases.

- Message Channel: connects components to each other, and provides services such as guaranteed delivery, and publish-subscribe mechanisms.
- Message: the basic unit of information that is communicated between components.
- Pipes and Filters: perform operations on messages, such as deleting unwanted content.
- Message Router: determine where a message should go, and help get it there.
- Message Translator: change the format of message, or integrate data from multiple sources.
- Message Endpoint: pull data from or push data to applications or systems.

The EIP website provides a summary of patterns. These patterns are implemented by a broad array of middleware systems and software. Implementations include:

- Apache Camel
    - Implemented patterns
- Microsoft BizTalk Server
    - Implemented patterns
- IBM WebSphere Application Server

Fast Infoset

Fast Infoset (FI) is a specification established by ITU, which provides for compression of XML documents. It is an application of ITU's ASN.1 standard, but knowledge of ASN.1 is not necessary to use Fast Infoset.

- Fast Infoset on Wikipedia
- Fast Infoset landing page at ITU
- Fast Infoset specification page at ITU
- Fast Infoset implemented as part of GlassFish

Server output filter

A server output filter is a software component of a web server that modifies data being transmitted to a client. A server output filter can compress XML data being transmitted.

The Apache site describes how server filters operate.

Some examples:

- Apache 2.0 uses mod_deflate.

- Microsoft IIS includes HTTP compression.

Size of NIEM XML documents

A common concern regarding NIEM is the size of NIEM documents. Large documents may take longer to send, or may take more disk space or other resources to process.

Reducing the size of XML documents

One strategy is to design XML documents so that they contain less data, or to reduce the amount of data sent. Strategies include:

1. Break up messages into small, independent parts.
2. Use caching to reduce retransmission.
3. Use lazy strategies: Only send what is needed, when it is needed.
4. Use NIEM constructs to avoid replicating data in a NIEM XML document.

Compression

If you are concerned with the size of an on-disk XML document, standard compression applications, such as gzip, compress XML documents, including NIEM documents, very well.

An XML document sent as a network message may be compressed using numerous strategies, including:

1. Use of a Server output filter
2. Fast Infoset
3. Efficient XML Interchange (EXI)

Each of these options has advantages and disadvantages.

A server output filter is generally the simplest to implement; they are available as standard parts of web servers, and may be automatically descripted by client libraries and browsers. Their compression ratios are very good, but since they are not XML-oriented, parsing of XML documents may be slower than the other strategies.

Fast Infoset operates entirely at the XML document level, so it does not require any NIEM-specific work.

Efficient XML Interchange (EXI) works at the schema level, compressing XML documents using the XML Schema for the document to guide compression. This means that the sender and receiver must use the same XML Schema for the document. There are some strategies that will improve EXI's performance in compressing XML documents.