2009-04-13

Demonstration of the eNotarization Markup Language (ENML) Version 1.0

Proposal Prepared for
the eNotarization Technical Committee
Legal XML Member Section
Organization for the
Advancement of Structured Information Standards

Western Illinois University
Laurence L. Leff, Ph.D.

This demonstration will be two command-line programs. The first is for the notary to prepare an eNotarized document given the binary for the document in some format such as XML, Word, PDF, etc. The second will allow a person having this eNML document to verify that it was signed by the notary. We will assume the notary has a public/private key. We will simulate the situation described in the Section 3.5 example (page 31), **except** that we support only a single signer. The person will type in a String that will serve as a "NULL" signature. It will be signed with the notary's private key. That is, the notary signature will use the `ds:signature` element as described in Section 4.7.

A demo web site will contain each notaries public key. The Element URL pointer will point to this entry. Hypothetically, each State's Secretary of State Office could list the public keys for its notaries that choose to participate in electronic notarization.

The first program is used by the Notary Public to notarize the document and create the ENML file.
It will take as input a file `enotary.in` that contains the following:
a) the Notary's private key
b) the Notary's name
c) the notary Commission's number
d) when the Commission Expires
e) the Notary Jurisdiction--we will use the United States form for this prototype and will read the County, State and Country.

f) The Notary Bond Number (if applicable)

The command-line program will take a single command-line argument, the name of the document to be signed.
(We will not attempt to "look at" the document or display it for the notary to inspect.)
The system will allow the notary to enter:
1) the `SignerIdentificationMethod`
2) the date (which will override the current date if entered)
3) the location (which will override the default of the Notary Commisson Information in `enotary.in`)
4) The Notary Identification Type (`Acknowledgement`, `Jurat`, etc. as defined in the

`NotaryIdentificationType` element)
5) the Notarization address (where this notarization occurred, 4.46)

It will then ask the signer to enter in the:
a) statutory text for the signature if any
b) the signer's name
c) the signer's title (optional)
d) the signer's prefix (such as Judge or Mr.)
e) the signer's address (we will use the United States form for the prototype, `SignerUSAAdress`)
f) `SignerIdentificationMethod`
g) the signer's address
h) `SignerSignature`, e. g. `/s= Mr. Ying Liu`

(For our prototype, we will simply represent these by passing the keyboard from the notary to the signer.)

It will produce the ENML, electronically signing the document using the private key.

The second program will validate an ENML document. We will test it and prepare the demo by inputting.

It has a single command-line argument, the received ENML document.

It will go the web site indicated in the `ds:RetrievalMethod`, verify that it was validly signed. Assuming it was, it would display information
about the document such as the notary's commission. It would also reverse the BASE64 encoding of the document and drop the file in the current directory.

Note, for this demonstration, I see no reason to record an X509 Digital Certificate. We are assuming that the web site referenced will have sufficient security and be sufficiently well-known. All that is needed is a pointer to that site and the public key to use can be looked up based upon the Commission Number.

For the purposes of the demonstration, we are not not exercising witnessed documents, ENML documents where the notary electronically notarized several documetns at once or documents with multiple signatures. Nor are we demonstrating apostille's.

As we discussed, this set of programs although functional is not commercial-grade or complete. This is for two reasons. It saves the Legal XML group funds. But more importantly, it is not the role of OASIS technical committees to compete with the software vendors. (Note, the committee would need to decide the disposition of the software; developers would find it helpful to extract the functionality and then build their business interface around it.)

That being said, for modest additional funds, we would complete the following:

1) Provide a GUI interface using Java Swing.

2) Provide a web based interface for one or both command-line programs. This would allow a notary

public to go to a web page. Theoretically, this could store the notary public's private key. They would enter a login and password, upload the document to be signed, and it would produce the ENML document for them to download and give to their customer.

3) We could demonstrate a web service that would validate an ENML document. One's program would submit the ENML document as an attachment and return a validity indicator. (Note I have been teaching web services for about two years) Verification of notarization would thus be an automatic component of work flows in large organizations.

4) We could also support some or all of the other features of ENML documents including: witnessed documents, ENML documents where the notary electronically notarized several documents at once or documents with multiple signatures and apostilles.

5) Our program for the notary could keep a notary journal.

6) We could provide an option in the GUI or command-line menu for the notary public could select the statutory text from the list appropriate for their state. Thus, if the notary is commissioned by California, they could select one of the standard statutory texts by clicking on a radio button.