NIEM-Conformant Encoding

General

The encoding rules for the NIEM[1]-conformant encoding are in alignment with the following NIEM documents:

- National Information Exchange Model (NIEM) Information Exchange Package Documentation (IEPD) Specification version 1.2
- NIEM Naming and Design Rules (NDR), Version 1.3

The 2011 version of the standard is based upon the NIEM-conformant encoding that was used in ANSI/NIST-ITL-2-2008. A substantial difference is the introduction of the Biometrics domain into the NIEM architecture. This allows updates to occur to the standard on a different schedule from that of NIEM itself. There are still elements in NIEM core that are used in the ANSI/NIST-ITL 1-2011 NIEM-conformant encoding, but many of the elements previously listed as either in the ansi-nist or itl namespaces are now specified in the biom namespace. New fields, subfields and information items occur in biom.

An instance document containing all record types and illustrating the use of every data element is available at www.biometrics.nist.gov/standard/instance-NIEM-1-2011.xml

The processing instruction with the UTF-8 character set is specified as:

<?xml version=”1.0” encoding=”UTF-8”?>

The root is specified as the following:

Itl:NISTBiometricInformationExchangePackage
xmls:xsi=”http://www.w3.org/2001/XMLSchema-instance”

xsi:schemaLocation=”http://biometrics.nist.gov/standard/1-2011/ITL-Package.xsd”

The namespaces are:

xmlns:biom=”http://www.biometrics.nist.gov/standard/biometrics-domain-1-2011”

xmlns:itl=”http://www.biometrics.nist.gov/standard/itl-1-2011”

xmlns:s=”http://niem.gov/niem/structures/2.0”

xmlns:nc=”http://niem.gov/niem/niem-core/2.0”

xmlns:fbi=http://niem.gov/niem/fbi/2.0

1. For further information see www.niem.gov

xmlns:i=http://niem.gov/niem/appinfo/2.0

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

The prefixes used in the 2011 version are: biom, itl, s, fbi, i and nc. Note that the prefix ansi-nist, which appeared in the 2008 version, is not used. Elements with this prefix in the 2007 version now have the prefix biom. Caution should be used when translating the 2008 version to the 2011 version, since there are some cases where the itl prefix has been changed to biom. This is since the biometrics domain (biom) contains the elements / constraints related to biometric data in the 2011 version. In the 2008 version, there were some elements that were originally placed in NIEM2.0 (due to the NIEM timeframe) but were revised in the ITL Package Schema (Annex B of the 2008 version of the standard). These elements were:

MinutiaNISTStandard

MinutiaDetail

MinutiaFingerPatternDetail

FingerPatternCodeSourceCode

Additional Data in the NIEM-conformant encoding

Note that in the encoding used in the 2008 version of the standard, there was not a direct mapping from the content of the 2007 version to the 2008 version. The principal differences were in listing organization names. The 2008 version added a field to state the name of the organization in text. This convention is retained in the 2011 NIEM-conformant encoding rules. The resulting encoding for ORI in NIEM-conformant XML (using an ID of WI013415Y) is:

```
<biom:TransactionOriginatingOrganization>
        <nc:OrganizationIdentification>
                <nc:IdentificationID>WI013415Y</nc:IdentificationID>
        </nc:OrganizationIdentification>
        <nc:OrganizationName>Text</nc:OrganizationName>
</biom:TransactionOriginatingOrganization>
```

Guidance on Missing Data

The content of certain elements (such as agency identifiers or types of transaction) is left to users to define. In the case of the element <biom:CaptureOrganization> (or Source agency / ORI), the sender and receiver may choose to establish an identifier for mission information. In this case, the preferred representation for missing data is to use the value “UNKNOWN”

```
    <biom:CaptureOrganization>UNKNOWN</biom:CaptureOrganization>
```

The standard format for the date (e.g. DAT in Record Type 1) IS YYYY-MM-DD with YYYY standing for Year, MM for Month and DD for Day. Partially missing date data may be represented as follows:

```
<biom:CaptureDate>

    <nc:Year>1995</nc:Year>

 <biom:CaptureDate>
```

OR

```
<biom:CaptureDate>

    <nc:YearMonth>1995-05</nc:YearMonth>

 <biom:CaptureDate>
```

For totally missing date data, the preferred representation is to “nil” the parent element as shown in this example:

```
<biom:CaptureDate xsi:nil=”true”/>
```

For the Greenwich Mean Time (GMT) the format is YYYY-MM-DDThh:mm:ssZ   Note that the date is separated from the time by the character “T” and the entire string concludes with the character “Z”.

For all Record Types except 1:

IDC: This mandatory complex element shall match the **ImageReferenceIdentification** found in the **TransactionContentSummary** element of the Type-1 record. Complex element **ImageReferenceIdentification** contains the simple element **nc:IdentificationID** that shall contain the image reference identification datum.

For the NIEM conformant encoding version of this standard, the Type 1 record is <itl:PackageInformationRecord> with its child element <itl:RecordCategoryCode> having a value of “01” Each record type has a different element name.   For subsequent records, the value “01” in RecordCategoryCode is replaced with the record type number.

Type-xx records shall be contained within this complex element:

```
<itl:Element name from list below>
        <itl:RecordCategoryCode>xx</itl:RecordCategoryCode>
                [. . . Type xx Record Content . .  .]
</itl:Element name from list below>
```

The record element names are:

1.	Type 1:	PackageInformationRecord
2.	Type 2:	PackageDescriptiveTextRecord
3.	Type 3:	PackageLowResolutionGrayscaleImageRecord
4.	Type 4:	PackageHighResolutionGrayscaleImageRecord
5.	Type 5:	PackageLowResolutionBinaryImageRecord
6.	Type 6:	PackageHighResolutionBinaryImageRecord
7.	Type 7:	PackageUserDefinedImageRecord
8.	Type 8:	PackageSignatureImageRecord
9.	Type 9:	PackageMinutiaeRecord
10.	Type 10:	PackageFacialAndSMTImageRecord
11.	Type 11:	PackageFacialImageRecord
12.	Type 12:	PackageSMTImageRecord
13.	Type 13:	PackageLatentImageRecord
14.	Type 14:	PackageFingerprintImageRecord
15.	Type 15:	PackagePalmprintImageRecord
16.	Type 16:	PackageUserDefinedTestingImageRecord
17.	Type 17:	PackageIrisImageRecord
18.	Type 18:	PackageDNARecord
19.	Type 19:	PackagePlantarImageRecord
20.	Type-20:	PackageOriginalImageRecord
21.	Type 98:	PackageInformationAssuranceRecord
22.	Type 99:	PackageCBEFFBiometricDataRecord

In Record Type-2:
This Record Type Is unique within the standard. It is used to convey information specific to sending and receiving organizations. The format is user-defined. This section describes how to encode this text in XML and remain consistent with the standard.

In the 2008 version of the standard, the element DomainDefinedDescriptiveText appeared as did OtherDescriptiveText. NIEM naming conventions state that a complex element (with child elements) cannot have a name ending in “Text” Therefore, in this version, these are changed to DomainDefinedDescription and OtherDescription.

This element shall only contain content defined by the Domain owner specified in the Type-1 record element <itl:TransactionDomain>. Individual XML elements, required for given transaction types, including tag names and content, shall conform to the specifications set forth by the agency to whom the exchange package is being sent. Each user-defined XML element used in the Type-2 record.

Complex element <itl:DomainDefinedDescription> is abstract, and as such is unusable by itself. Implementers should define, in an extension schema, a substitution element containing user-defined child elements from the user's domain.

A substitution element should be defined in a user's extension schema similar to this:

```
<xsd:element name=”DomainDefinedDescription”
substitutionGroup=”itl:DomainDefinedDescription”
        type=”user-domain:DomainDefinedDescriptionType”/>
<xsd:complexType name=”DomainDefinedDescriptionType”>
        <xsd:complexContent>
                <xsd:extension base=”s:ComplexObjectType”>
                        <xsd:sequence>
                        <xsd:element ref=”user-domain:OneField”/>
                        <xsd:element ref=”user-domain:TwoField”/>
                        </xsd:sequence>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

The element would then appear in an instance document like this:

```
<user-domain:DomainDefinedDescription>
        <user-domain:OneField>Text</user-domain:OneField>
        <user-domain:TwoField>Text</user-domain:TwoField>
</user-domain:DomainDefinedDescriptionveText>
```

Element <itl:OtherDescription>
This element shall contain additional content not defined by a Domain owner, but necessary for information exchange between certain parties. Individual XML elements, required for given transaction types, including tag names and content, shall conform to the specifications set forth by the agency to whom the exchange package is being sent.

Complex element <itl:OtherDescription> is abstract, and as such is unusable by itself. Implementers should define, in an extension schema, a substitution element containing user-defined child elements from the user's domain.

A substitution element should be defined in a user's extension schema similar to this:

```
<xsd:element name=”OtherDescription”
        substitutionGroup=”itl:OtherDescription”
        type=”user-domain:OtherDescriptiionType”/>
<xsd:complexType name=”OtherDescriptionType”>
        <xsd:complexContent>
                <xsd:extension base=”s:ComplexObjectType”>
                        <xsd:sequence>
                                <xsd:element ref=”user-domain:OneField”/>
                                <xsd:element ref=”user-domain:TwoField””/>
                        </xsd:sequence>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

The element would then appear in an instance document like this:

```
<user-domain:OtherDescription>
        <user-domain:OneField>Text</user-domain:OneField>
        <user-domain:TwoField>Text</user-domain:TwoField>
</user-domain:OtherDescription>
```

In Record Type 7:
Complex element <biom:RecordImage> is abstract, and as such is unusable by itself. Implementers should define, in an extension schema, a substitution element containing user-defined child elements from the user’s domain.

A substitution element should be defined in a user’s extension schema similar to this:

```
<xsd:element name=”UserDefinedImageRecord”
        substitutionGroup=”biom:RecordImage”
        type=”user-domain:UserDefinedImageRecordType”/>
<xsd:complexType name=”UserDefinedImageRecordType”>
        <xsd:complexContent>
                <xsd:extension base=”s:ComplexObjectType”>
                        <xsd:sequence>
                                <xsd:element ref=”user-domain:OneField”/>
                                <xsd:element ref=”user-domain:TwoField””/>
                        </xsd:sequence>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType >
```

In Record Type 9:
At least one occurrence of a concrete substitute for the abstract element is required.

Complex element <biom:RecordMinutiae> is abstract, and as such is unusable by itself. Implementers may use the NIST “Standard format” substitution element <biom:Minutiae>. Implementers alternatively may define, in an extension schema, a substitution element containing user-defined child elements from the user’s domain. Vendor-specific substitution elements may be registered with the domain owner specified in the Type-1 record, element <biom:TransactionDomain>.

A substitution element should be defined in a user’s extension schema similar to this:

```
        <xsd:element name="VendorDefinedMinutiae"
                substitutionGroup="biom:RecordMinutiae"
          type="user-domain:VendorDefinedMinutiaeType"/>

        <xsd:complexType name="VendorDefinedMinutiaeType">
                <xsd:complexContent>
                        <xsd:extension base="s:ComplexObjectType">
                                <xsd:sequence>
                                <xsd:element ref="user-domain:OneField"/>
                                <xsd:element ref="user-domain:TwoField”"/>
                                </xsd:sequence>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>
```

The element would then appear in an instance document like this:

```
        <user-domain:VendorDefinedMinutiae>
                <user-domain:OneField>Text</user-domain:OneField>
                <user-domain:TwoField>Text</user-domain:TwoField>
        </user-domain:VendorDefinedMinutiae>
```