FSML™ - Financial Services Markup Language

Version 1.50

Written by the members of the FSTC Electronic Check project team
Edited by Jeff Kravitz, IBM Research

First Written February 15 1996
Latest Revision July 14 1999

FSTC publishes FSML for your review and comment only at this time. You may reproduce and distribute this to others to facilitate their review and commentary only. Please note that the FSML specification is not final.

FSTC DOES NOT MAKE AND WILL NOT BE DEEMED TO HAVE MADE ANY REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, AS TO THE CONDITION, MERCHANTABILITY, DESIGN, OPERATION, OR FITNESS FOR A PARTICULAR PURPOSE OF THE FSML OR ANY OTHER WARRANTY OR REPRESENTATION WHATSOEVER, EXPRESS OR IMPLIED, WITH RESPECT TO THE FSML.

"FSML" is the trademark of the Financial Services Technology Consortium and may be used only with the prior express written permission of the Financial Services Technology Consortium subject to appropriate license terms.

Please direct all questions or comments to:

```
Frank Jaffe
FSTC Vice President
c/o BankBoston
100 Federal Street.  Mailstop: 01-24-03
Boston, MA 02110
(Voice) 617-434-1838 (Fax) 617-434-9889 (E-mail) fjaffe@netcom.com
```

Contents

Introduction

A child of five would understand this. Send someone to fetch a child of five.

– Groucho Marx

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

– John von Neumann

Make everything as simple as possible, but not simpler.

– Albert Einstein

Research is what I'm doing when I don't know what I'm doing.

– Wernher Von Braun

PROGRAM: n. A magic spell cast over a computer allowing it to turn one's input into error messages. tr.v. To engage in a pastime similar to banging one's head against a wall, but with fewer opportunities for reward.

Then anyone who leaves behind him a written manual, and likewise anyone who receives it, in the belief that such writing will be clear and certain, must be exceedingly simple-minded.

– Plato, Phaedrus 275d

Read over your compositions, and where ever you meet with a passage which you think is particularly fine, strike it out.

– Samuel Johnson, quoting a college tutor, 1773

The knowledge of Cyphering, hath drawn on with it a knowledge relative unto it, which is the knowledge of Discyphering, or of Discreting Cyphers ... Certainly it is an Art which requires great pains and a good wit, and is (as the other wits) consecrate to the Counsels of Princes.

– Sir Francis Bacon, 1623

FSML, the Financial Services Markup Language, is an SGML[1] like mark-up language designed to allow the creation of electronic financial documents.

FSML can be used for several applications. One is to create, and sign generic financial (or even non-financial) documents. Another, more specific application, is to create, and process Electronic Checks, and their associated documents. One chapter will describe the use of FSML for generic documents. Later chapters will describe FSML when used to create and process Electronic Checks and associated documents.

Some of the goals and decisions used in the design of FSML were...

- tag the individual text items making up a document,
- group the text items into document parts which can have business meaning and can be signed individually or together,
- allow document parts to be added and deleted without invalidating previous signatures, and
- allow signing, co-signing, endorsing, co-endorsing, and witnessing operations on documents and document parts.

The signatures and associated certificates become part of the FSML document and can be verified by subsequent recipients as the document travels through the business process. FSML does not define privacy encryption, since privacy encryption is between each sender and receiver in the business process and can differ for each link depending on the transport used.

When development of FSML began in 1995, HTML was in its early stages of widespread deployment. SGML had been standardized some years earlier, software tools were readily available, and the use of tagged, readable text was attractive for its simplicity, ease of understanding, operational support, and ease of development and use. FSML was designed so that it could be defined using an SGML Document Type Definitions (see Appendix A). FSML also defined document formatting rules so that readable text electronic checks could be sent via electronic mail systems without the risk that the mail systems would modify the electronic check in ways that invalidate the signatures.

FSML has been implemented by the Electronic Check Project. Cryptographic hardware, in the form of smart cards, has been developed to contain the private signing keys, perform the hashing and signing operations and to perform other ”electronic checkbook” functions, such as automatically numbering and logging checks written or deposited. Advice of payments are attached cryptographically to the checks when sent between payer and payee, and they are removed by the payee. Similarly, checks are attached cryptographically to deposit slips when they are sent to the payee’s bank. Bank server systems have been developed to process the electronic checks, to interface with existing check processing systems in the banks, and to clear and settle electronic checks between banks. A Certificate Authority hierarchy has been established, and certificates have been issued to banks and checking account holders in the US.

The Electronic Check project, from its inception, has sought to develop a general solution to the issues of authentication and integrity associated with creating electronic financial instruments. The technical and business problems of implementing electronic check payments between payers and payees over the Internet provided a practical context for developing the solution.

Paper checks have a rich tradition, and support numerous options, check types, attached information, and sophisticated processing. Paper checks are fundamentally a signed writing directing a bank to pay money, after a date, from the payer’s account. The Electronic Check project determined that the essence of the problem to be solved was to develop a generalized structure for creating, processing, and displaying electronic ”signed writings”, where cryptographic signatures would substitute for manual signatures and where an electronic message would take the place of the paper medium. The structure would need to support the same business operations as signed paper checks, such as signing, co-signing, and witnessing of signatures, and attaching and removing associated documents such as remittance slips, invoices, advice of payment, and deposit slips.

Since checks are a form of negotiable instruments, and negotiable instruments are a form of contracts, it is believed that FSML may be used to create signed documents suitable for a wide variety of purposes. For example, they may be used as messages to initiate electronic funds transfers, as orders and invoices needed for electronic commerce, or for other forms of signed contracts or agreements.

FSML documents, which are hashed and cryptographically signed using public key signature algorithms, can have the following security attributes:

Authentication
A document recipient can authenticate that the document was created by a specific person, or institution, and was not forged or created by an imposter.

Integrity
A document recipient can determine that the document was not changed or corrupted in any way since it was created by the originator.

Non-repudiation
A document recipient can provide evidence that a document was definitely created by the originator even if the originator "repudiates" the document, claiming that someone else created it using their name.

The FSML signature mechanism also allows documents to be combined, or added to, without loss of these security features.

Some of the business objectives that were instrumental in the design of FSML were...

- To develop a general method for creating and verifying business documents at the application level with integrated digital signatures
  - To provide assurance to the application and the customer
  - To eliminate the need for paper source documents
  - To allow incorporation into a number of different business applications
  - To support peer to peer exchanges
  - To keep, indeed to require, signatures integrated with the documents
  - To identify the individual or organization which created the documents
  - To identify the individuals or organizations which process the documents
  - To support signing, cosigning, counter signing, endorsing, and witnessing
  - To be vendor neutral
  - To be cryptographic algorithm neutral
- To provide a high degree of flexibility in the information content and structure of the signed documents
  - To support a wide range of different documents
  - To allow for efficient processing
  - To clearly define the scope of a signature such that different signers could sign different parts of the same document, or indeed, where needed, could sign someone else's signature

- To provide for the attachment and removal of auxiliary documents while still maintaining the digital signature integrity
  - To allow binding information together
  - To allow removal of information when it reaches its intended recipient to protect privacy and improve efficiency of later handling
- To be usable through any electronic transport media
  - To enable end recipients who do not expect a signed document to understand what they received
  - To be independent of the network or electronic communication system used
  - To work through online connections, such as provided by the WWW or secure sockets
  - To work through any electronic mail system, including those only capable of text transmission.
  - To work through legacy communications and computer systems
- To allow documents to be as self contained as possible
  - To recognize and support different documents which have different security and time duration and immediacy requirements
  - To enable critical processing to occur in simple devices, such as smart cards, where appropriate to the application
  - To provide reasonable assurance to off-line or off-network processing
  - To minimize the reliance on third parties
  - To minimize the need for third party directories
  - To minimize the need for security databases at the end users
  - To simplify auditing and research
- To provide control to the transacting parties
  - By providing flexibility in managing transaction risk as needed
  - By allowing direct peer to peer exchanges
  - By minimizing systemic risk to transacting parties by supporting inexpensive secure processing devices such as smart cards
  - By enabling document semantics to control issues such as transaction effective dating and validity intervals
  - By enabling third party services, such as time stamping, electronic postage, third party processing, or archival storage as needed by the transacting parties
  - By allowing separate control of the privacy requirements

The above business objectives led to a number of technical decisions, based on much thought and discussion amongst the team members. Some of these technical decisions were...

- An SGML derivative language was chosen because it was an ISO standard, and had the additional benefit of using standard printable ASCII text. Using a binary encoding method (e.g. ASN.1) would have meant that all FSML documents required special decoding software just to be viewed or even detected. This seemed too much of a burden since the uses and platforms envisioned would have been too many and diverse for such viewing software to become easily available. Using an ASCII text based language meant that a casual user could still look at and identify an FSML document, with only normal E-mail, Web, or other ASCII-text viewing capabilities.

- It was decided to subset many of the SGML features (e.g. limited use of SGML end-tags) to allow FSML documents to be more easily processed by devices and systems of lower complexity. It was envisioned that some of the processing may be done on electronic tokens meaning PDA's, PCMCIA cards or even smart cards, with limited memory and processing capabilities.

- Formatting and encoding rules that are not part of the ISO SGML standard were added to allow FSML documents to pass, uncorrupted, through a variety of transport mechanisms, especially E-mail, and still allow the signatures to be valid. This decision has been very useful, and appears to be one of the key items in FSML.

- It was decided to keep FSML documents independent of the transport mechanism used to transfer them from party to party. Thus, they can be transported by a large variety of mechanisms, e.g. E-mail, Web (HTTP), FTP, or even as files on a diskette.

- Although privacy is a very important requirement, it was decided not to incorporate a privacy encoding scheme into the FSML definition, as a number of privacy encoding methods and systems were being defined by others, and it was felt that this decision could best be left in the hands of the other groups who were doing that for the various transport mechanisms, e.g S-MIME, PGP, etc.

- FSML is designed to be extensible. Additional document types and blocks (sub-elements) may be added for specific applications. In addition, FSML, as defined in this document, allows attachment and signing of virtually any kind of attached document or file, with no additional FSML changes or extensions required.

- The basic document signature mechanism was designed to allow attachments to be added to and detached from an FSML document, and for additional signatures to be added after original document was created. Thus, the signature mechanism was thought of as a kind of ”electronic staple”.

- All of the signature verification information (except for the public key of the root of the certificate hierarchy) is in the FSML document itself. Thus, any document recipient can perform a full cryptographic signature verification on the document without the need for access to external networks, directories, servers or databases. The only external information needed is the root public key, which can be easily distributed widely.

- It was decided to use X.509 certificates, even though they do not fit in well with other FSML design goals (e.g. X.509 certificates use binary ASN.1 instead of human-readable ASCII) because they have become a widely adopted standard for public-key certificates. Note, however, that the FSML language allows for specification of other certificate types.

- The signature algorithms chosen are the most popular at the time this specification was designed (e.g. RSA/MD5, DSA/SHA-1 EDSA/SHA-1), however the FSML language allows specification of other algorithms which may become available in the future.

An FSML document is structured using more specific syntax rules than normal SGML. SGML defines *elements*, which are delimited by *start tags* and *end tags*, and which may contain textual content or other elements. FSML defines several different varieties of SGML elements. FSML defines a *field* to be an element with a start tag and with content and with no end tag. (Unlike in SGML, where an end tag may defined to be optional, in FSML fields must have no end tag.) The content of a field must be purely textual and must not contain other elements — i.e., the content of a field must not contain SGML markup.

The next level of syntactic definition of FSML is the *block*. A block consists of a start tag, one or more fields, and an end tag. A block must not contain textual content that is not part of a field.

Finally, a *document* consists of a set of blocks and can contain nested documents. Documents must also have start and end tags. As with blocks, documents must not contain textual content not part of a field, and they must not contain fields not part of a block.

FSML is extensible, but this extensibility is defined only in terms of documents, blocks, and fields. Extensions using arbitrary SGML constructs are not permitted; they must consist of documents, blocks, or fields.

SGML comments are not allowed in FSML documents.

All blocks that must be protected from tampering, and all blocks that must be authenticated are signed using a digital signature, which is contained in a signature block. The digital signature uses one of the standard digital signature algorithms, such as MD5/RSA[2][3], SHA/DSS[4][5], or SHA/ECDSA[6], although the use of MD5 is deprecated. Each signature requires a public key, which also requires a certificate. Certificates are distributed as X.509 Version 1 or Version 3 certificates[7].

Blocks may also be “bound” together by the signature block, which contains the block names of the blocks being bound, the digital hashes of these blocks, and a digital signature on these hashes along with the other contents of the signature block. This binding allows the receiving software to verify that all the blocks that were bound are present and have not been tampered with.

The concept of the FSML Electronic Document is that it is a flexible structure. Separating signatures, certificates, actual data, etc. into separate blocks allows a rich, complex document to be built from these “primitives,” while retaining a standard format which can be parsed and verified according to a standard syntax definition.

A number of features or functions described in this document are said to be deprecated. The use of the word deprecated in this context means that use of the feature or function is *strongly discouraged, but not forbidden*. Programs which create FSML documents should avoid use of deprecated features, however programs which receive and process FSML documents should not reject them solely because they use deprecated features.

Notation and Syntax

2.1 Notation

In the definitions below, an attempt is being made to show examples of the format for FSML electronic documents, rather than to use formal meta-linguistic notations to define them. A more accurate definition is also in the document using more formal notations (Extended BNF[8][9] and SGML DTD[1]).

In the later definitions, the following simple notations are used to indicate the type of value being used for a particular field.

namestring a sequence of characters taken from the ASCII subset

`abcdefghijlkmnopqrstuvwxyz0123456789`

where the first character must be a letter.

dnamestring a sequence of characters taken from the ASCII subset

`abcdefghijlkmnopqrstuvwxyz`
`ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-.`

where the first character must be a letter.

valuestring a sequence of any of the allowed set of ASCII characters (see the Character Encoding section) except for the tag delimiters '<' or '>'. Line end characters are allowed but are not considered to be part of the value. Tag delimiters and ASCII characters outside the allowed set may be included in a value string by using the standard SGML[1] character entity mechanism. Quote symbols have no special significance, and if contained in a value string, will be considered part of the value — e.g., `"John Smith"` is not the same value as `John Smith`. In all value strings, mixed-case significance is honored — i.e., the string `John Smith` is not equal to the string `john smith`.

numstring An ASCII character string used to denote an integer numeric string, containing only the digits 0 - 9. This string may include leading zeros, which are considered significant and are part of the string and must be used for purposes of comparison.

number An ASCII character string used to denote an integer numeric string, containing only the digits 0 - 9. This string may include leading zeros, which are *not* considered significant and may be ignored for purposes of comparison or calculation using the number.

amountstring An ASCII character string used to denote a decimal (real) number, containing only the digits 0 - 9 and a single, optional decimal point. Zero, one, or two digits are allowed after the decimal point, depending on the specific currency conventions. Leading zeros are not considered significant.

vstring An ASCII character string used to denote a version number containing only the digits 0 - 9 and a single decimal point. There can be one or two digits before the decimal point, and one digit after the decimal point. Leading zeros are not permitted.

hexstring An ASCII character string used to denote the hexadecimal encoding of a binary string of octets. It may only contain the ASCII characters 0-9, A-F, and a-f. All legal hexadecimal strings must consist of an even number of hex digits. In certain cases, described when used below, the field is split into two or more portions using a colon `":"` — e.g., `0123456789:abcdef`. Leading zeros in a binary octet string are significant and should be present in the hexadecimal representation of that octet string. Note. hexadecimal strings were used to represent binary data in FSML versions 1.17 and before. FSML 1.50 uses base64 encoded strings instead.

base64string An ASCII character string used to denote the base64 encoding of a binary string of octets. It may only contain the ASCII characters “0-9”, “A-Z”, “a-z”, “+”, “/”, and “=”. In certain cases, described when used below, the field is split into two or more portions using a colon `":"` — e.g., `qsdfcx789:abcdef`. Leading zeros in a binary octet string are significant and should be present in the base64 representation of that octet string.

other A string depicted in **boldface** as a value represents itself. All such self-defining-constants must be in lowercase in all FSML documents.

2.2 FSML BNF Description

The following is an Extended BNF[8][9] description of the structure of FSML Electronic Documents.

BNF Meta-Notation

The meta-symbols of BNF used here are:

::= meaning “is defined as”

| meaning “or”

[ ] used to enclose optional items

” used to enclose characters or strings that represent themselves

{ } used to enclose repeated items (repeated zero or more times)

Names not enclosed in any of the above bracket symbols are called *nonterminals* and are used to define symbols internal to the BNF specification only.

```
name    ::= a sequence of characters taken from the ASCII subset
            abcdefghijlkmnopqrstuvwxyz0123456789, where the
            first character must be a letter.
dname   ::= a sequence of characters taken from the ASCII subset
```

```
            abcdefghijlkmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_-
.
            where the first character must be a letter.
pname   ::= name | ’x:’ name
tvalue  ::= ’true’ | ’false’
value   ::= a sequence of any ASCII characters
            from the set in the range of hex values
            20-7E inclusive, excluding ’<’ or ’>’
            including spaces and new-line sequences
            (new-line sequences are allowed but are not considered
            part of the value).
avalue  ::= a sequence of any ASCII characters
            from the set in the range of hex values
            20-7E inclusive, excluding ’<’ or ’>’ or ’"’
            and excluding new-line sequences.
            but including spaces,
tag     ::= ’<’ pname ’>’
fldtag  ::= tag | ’<’ pname { name ’=’ ’"’avalue ’"’ } ’>’
endtag  ::= ’<’ ’/’ pname ’>’
```

A tag in FSML is essentially the same as in SGML, but uses a more restricted set of characters. Note that whitespace is permitted between the angle brackets and the tagname, nor is any permitted inside the tagname itself. A tag may have one or more optional attributes, which consist of an attribute name, an equal sign, and an attribute value enclosed in double quotes. A tag may have zero, one, or more than one attributes. There must be no whitespace between the attribute name, the equals sign, and the attribute value. There must be exactly one space between the tag name and the first attribute, and exactly one space between multiple attributes, and no spaces between the last attribute and the closing angle bracket for the tag.

```
field   ::=   fldtag value
```

A field consists of a tag followed by a value. The tagname used in a field identifies the type of the field. Unlike in SGML, fields may not contain end tags. Spaces between the tag and the value are considered part of the value. Spaces at the end of a line are not considered part of the value.

```
block_start_tag  ::=   tag
block_end_tag:   ::=   endtag
```

A block is started by a block start tag, which is a tag with a unique tagname (which identifies the type of block). A block start tag is never followed by a value string, but must be followed immediately by another tag. The block is ended by a block end tag, which must have the same tagname as the block start tag.

```
subblock_start_tag  ::=  tag
subblock_end_tag:   ::=  endtag
```

A subblock is used to enclose a group of fields within a block or another subblock. Subblocks use the same types of start and end tag syntax as blocks. Subblocks may only appear inside blocks, or other subblock and are only used for grouping purposes (analogous to parentheses). A subblock start tag is never followed by a value string, but must be followed immediately by another tag. The subblock is ended by a subblock end tag, which must have the same tagname as the subblock start tag.

```
block   ::=  block-start-tag
             field | subblock
           { field | subblock }
             block-end-tag
```

A block consists of a block start tag, one or more fields or subblocks and a block end tag. In most cases the order of the fields or subblocks is not determined. Some fields must appear in a predetermined order. This is described when necessary.

```
docname_attr  ::= ’docname="’ dname ’"’
doctype_attr  ::= ’type="’ avalue ’"’
doc_start_tag ::= ’<fsml-doc docname="’ docname_attr ’ ’ doctype_attr ’>’  |
                  ’<fsml-doc docname="’ doctype_attr ’ ’ docname_attr ’>’
doc_end_tag   ::= ’</fsml-doc>’
```

A document start-tag is a start-tag with the tagname ’fsml-doc’ and it must contain the two attributes ’docname’ and ’type’, which can appear in any order.

```
fsmldoc       ::= doc_start_tag
                   block  |  fsmldoc
                 { block  |  fsmldoc }
                  doc_end_tag
```

An FSML document consists of a document start tag, one or more blocks and/or nested FSML documents, and a document end-tag.

Document Formatting Rules

In order for the FSML Electronic document to be easily transmitted by a variety of methods (E-mail, file transfer, storage media, etc.) it was designed to be a plain ASCII document. However certain formatting rules must be adhered to in order to ensure that most of the usual transport mechanisms, in particular E-mail systems[10], will successfully transport the FSML Electronic document unchanged.

Note that these rules supersede the white space and line end rules for SGML, in order to ensure that signed documents can be successfully verified. Therefore, the following document formatting rules are considered mandatory for document generators and receivers.

- Character Encoding
  - An FSML document consists of a sequence of octets (eight-bit bytes) whose values are from the set hex 0A, hex 0D and hex 20-7E. Values outside the permitted set are not allowed. Programs that process FSML must reject documents that contain values outside the permitted set.

    The permitted octet values correspond to characters in the ASCII[11] character encoding. Use of other character encodings (such as EBCDIC) for the values of characters is not permitted. For example, the following sequence of octet values (given in hex)

    ```
        3C 61 6D 6F 75 6E 74 3E 31 32 33 2E 34 35
    ```

    unambiguously represents the ASCII character string `"<amount>$123.45"` and vice-versa.

    Note that the octet value 9 (which corresponds to the ASCII TAB character) is not permitted in FSML documents.
  - If a character is required that is not displayable as one of the above ASCII printable characters, it should be encoded in the document using an SGML entity name for the character[1], enclosed between an ampersand and a semicolon — e.g., `&circumflex;`.

    This rule applies in particular to the > and < characters, which can only be used as SGML tag delimiters. If they are to be used in normal text, they must be encoded as &gt; and &lt;

    If an FSML implementation chooses to use numeric character entities — i.e.,

    ```
    CharacterEntity ::= ’&#’ [0-9]+ ’;’ | ’&#x’ [0-9a-fA-F]+ ’;’
    ```

    Then the decimal or hexadecimal numeric value should refer to the code point in ISO/IEC 10646 Unicode. Characters in the range `&#x0;` through `&#x7F;` are equivalent to ASCII, and characters in the range `&#x80;` through `&#xFF;` are equivalent to ISO 8859-1 Latin 1.

Note that these SGML entities will not be translated during the processing and cryptographic hashing of an FSML Electronic Document. They are only used for the purpose of display or printing of characters not in the usual ASCII subset.

- Line Formatting
  - A line-end sequence usually consists of either an ASCII Carriage Return (hex 0D) or an ASCII Line Feed (hex 0A) or both; however, this is operating-system specific. Mixed conventions are permitted.

    Consuming systems should be prepared to receive documents with mixed line-end conventions.
  - All lines in the document must be less than or equal to 76 characters in length. When it is necessary to create a line which will continue past 76 characters, a line-end sequence may be inserted into the line. (this is to protect the FSML document from E-mail systems which break up lines longer than 76 characters)

    A line-end sequence must not be inserted, either during original document creation, or when attempting to modify a document to protect against E-mail systems, if any of the following would become true:
    - It would create a line consisting of a single period (`'.'`). (this is to protect FSML documents from E-mail systems which cease processing documents past a line consisting of a single period)
    - It would create a line consisting of the string `'From '`, starting in the first position of a line. (this is to protect the line from certain E-mail systems which change such lines)
    - It is inserted after spaces, causing those spaces to be trailing spaces on a line.
    - It causes a line-end sequence to be inserted into a tag.
  - In FSML document receiving programs all line-end sequences will be removed before any processing is done on the FSML Electronic document. In particular they are not included in hash calculations, nor passed in field values to application programs.
- Space Handling
  - Any spaces at the end of a line — i.e., trailing spaces will be removed prior to processing the document. This means that spaces between the last non-space character on a line, and the line-end sequence for the line are not included in hash calculations and are not passed to applications as part of the field value. This is also done inside <adata> sub-blocks inside attached documents. (this is to protect FSML documents from E-mail systems that may truncate lines with trailing spaces, or may add trailing spaces to lines). A completely blank line — i.e., a line consisting entirely of white space, will be removed entirely before processing.
  - Embedded spaces — i.e., spaces that are not immediately before a line-end are to be processed as is — i.e., they are not to be removed before processing, hashing, etc. The same is true for leading spaces on a line — i.e., spaces following a line-end sequence.
  - Leading spaces — e.g., indentation, are allowed but not recommended, as leading spaces are not deleted from processing, and as line-ends are removed, the leading spaces on a line will be indistinguishable from the data ending the previous line, and thus may cause field length violations or field data misinterpretation. For example:

```
<tag1>abcde
     <tag2>defghi
```

would cause the value of tag1 to be `'abcde       '`. This can be used, however, to place trailing spaces into a value — i.e., if the value if tag1 were desired to contain the trailing spaces.

  - White space (either spaces or line-end sequences) may not be inserted into tags, either between the < and the tag name, or between the tag name and >, or inside the tag name. If a tag has attributes, white space may not be inserted between the attribute name and the = or between the = and the attribute value, or inside the attribute name, or between the last character of the attribute value and the >. Multiple attributes must be separated by exactly one space. Parameter values must be contained within quotes.

- Tags

  In order to allow FSML processing using the limited resources available in the Electronic Token, FSML requires that certain SGML features for tag handling not be allowed.

  - End tags (e.g., </tag>) are not allowed except for </fsml-doc>, block-end tags (defined below) and specific sub-block end tags as defined below. All fields that do not have end-tags specified in the block and field definitions below must not have end-tags.
  - SGML tag abbreviation is not supported.

Generic FSML

4.1 Generic Electronic Document Definition

This chapter explains the syntax and semantics of “Generic” FSML documents. Such documents may be used for any purpose, and need not be used for electronic checking, or even for financial applications. This chapter contains the specification which was previously published as *SDML - The Signed Document Markup Language*. This chapter and the previous chapters supersede that publication.

4.2 FSML Document Definition

Every FSML electronic document consists of one or more enclosed documents. These documents are nested, with the nesting done by enclosing earlier forms of a document inside later additions to the document. Each enclosed document is built inside a <fsml-doc> tag structure. Inside a document are one or more blocks. Blocks may appear in any order, except that the <action> block (defined below) must the first block in the document. All generic blocks other than the <action> block may occur multiple times in an FSML document.

```
<fsml-doc docname="dnamestring" type="dnamestring">
   a sequence of one or more blocks and/or nested <fsml-doc> documents
</fsml-doc>
```

Figure 4.1: Document element definition

The `docname=` attribute is a document name, assigned by the software creating the document. This name will be used when combining documents. (See Combining Documents, below). If multiple FSML documents are being created at one time, as part of one file or transmission, the creating software must ensure that the document names are unique, within the file or transmission. This name must contain a maximum of 64 characters. Note: Attribute values must be enclosed in quotes.

The `type=` attribute is a document type, used to specify the type of document. This type is used by the receiving software to ensure that it has received the correct type of document — i.e., one that it knows how to process. The document types are chosen from a list of pre-defined types, or may be types agreed upon by the sending and receiving parties, except that the latter agreed-upon types may not conflict with any pre-defined types. Note: Attribute values must be enclosed in quotes.

To prevent such conflict between pre-defined, standardized document types, and privately agreed-upon types, all privately agreed-upon document types should be prefixed with the characters “**x:**” (meaning private). For example, a document type used for auto-loan applications, agreed to be used by a pair or small group of cooperating banks, could be written as `type="x:autoloan"`. All pre-defined document types will be guaranteed not to start with the characters “**x:**”.

4.3 Generic Block Common Field Definitions

A block contains some common fields, along with other fields specific to the type of block. Except in a few cases and unless otherwise specified, the order of fields within a block is not predefined. Once a block has been signed, however, fields may not be moved or rearranged inside a block, nor may fields be added or removed.

Common Block Field Definitions Each of the blocks contains some field definitions which are common to all block types, as follows:

```
<blkname>namestring
<crit>tvalue
<vers>vstring
```

Figure 4.2: Elements common to all blocks

blkname (**required**) This is a character string which must contain a block name assigned at the time the document is created. The creating software must ensure that the block names are unique within a document. The names are used to refer to the block from other blocks.

Generally, the name chosen for the block may be any unique character string. For certain blocks a convention or rule applies when creating block names. The rules or conventions are described in the individual block descriptions.

crit (optional) A boolean (**true**/**false**) flag used to determine if a block is critical. If a block is critical, then the receiving software must be able to process the block, and must be able to recognize every field in the block. If the software cannot process a critical block, it must abort processing the entire document, or otherwise determine how to handle the document as an exceptional case. This flag is used to allow for expansion of the block types, to allow software to “ignore” block types that it doesn’t recognize, providing that they are marked non-critical by the software that created them. Certain types of blocks, such as informational messages, etc. might always be considered non-critical. Other types, such as signatures, might always be considered critical. The criticality flag is assumed to have a default of `true` unless otherwise specified as `false`. Thus, it is not required to be specified in every block.

vers (optional) A number which indicates the Version of the block. New versions may be introduced, and this number is used by receiving software to determine if it is capable of parsing/processing a block. If the version number is larger than the one understood by the receiving software, it must assume that it cannot process the block, and must use the criticality flag to determine if it can continue to process the document. If the version number is not specified, it is assumed to be 1.0. A given FSML document may contain blocks with different version numbers. The receiving software is responsible for identifying inconsistencies and conflicts that may arise if FSML blocks with different versions are mixed inappropriately within the same document.

4.4 Generic Block Definitions

Each Generic FSML block starts and ends with one of the following sets of block tags:

```
Start Tag                 End Tag

 <action>                </action>
 <signature>             </signature>
 <cert>                  </cert>
 <attachment>            </attachment>
 <message>               </message>
 <x:name>                </x:name>
```

Figure 4.3: List of generic block elements

The block types are defined as follows:

action	A block describing the action to be performed by the recipient
signature	A block with the signatures and hashes of other blocks
cert	A public key certificate
attachment	An associated document attached to an FSML document
message	An informational message, such as an error report
x:name	A privately defined block type.

4.4.1 Generic Action Block Definition

This block contains information about the action to be performed by the recipient of the Electronic Document.

```
<action>
<blkname>namestring
<crit>true
<vers>1.0
<function>namestring
<reason>namestring
</action>
```

Figure 4.4: Action block element definition

Action Block Field Definitions

function (**required**) The function field contains a character string chosen from a set of commands or verbs specific to the application of the document being sent. Each application will have a unique set of allowable functions that are supported. (see echeck specific Action block definition below).

reason (**required**) The reason field indicates the reason that the document is being transmitted to the recipient. It must be one of the following character strings.

process This indicates that the document is an original being sent to the recipient for normal processing.

resend This indicates that the document is a possible duplicate being resent to the recipient. It should only be processed if it is not a duplicate at the receiver.

test This indicates that the document is being sent as a test, and should not be fully processed (e.g., it should not transfer funds).

info This indicates that the document is being sent for informational purposes only (e.g., as part of the text of an email message) and is not to be processed.

return This indicates that the document is being sent back to the originator as a returned item. The document will usually contain a <message> block indicating the reason for the return.

4.4.2 Generic Signature Block Definition

This block contains a digital signature for another block, or set of blocks. It is required whenever the originator or approver of a block must be authenticated, or the block tamper-proofed. It also contains the reference to the certificate block containing the public key used to verify the signature. It is also used to “bind” multiple blocks together, so that the resulting compound document can be verified.

Unless otherwise specified, the data being signed consists of the entire contents of the subject block, which is defined to be everything including the start and end tags for the block. The signature must include the blockname, criticality, and version fields, if present, as well as the contents, begin, and end tags of the block.

The actual hashes of the signed blocks are included to allow verification of the binding even if the actual contents of the bound blocks are not available.

```
<signature>
<blkname>namestring
<crit>true
<vers>1.5
<sigdata>
<blockref req="true">dnamestring
<hash alg="sha">base64string
<blockref req="true">dnamestring
<hash alg="sha">base64string
  ...
<blockref req="true">dnamestring
<hash alg="sha">base64string
<nonce>valuestring
<sigref>namestring
<sigtype>namestring
<certissuer>valuestring
<certserial>number
<algorithm>valuestring
<timestamp>valuestring
<location>valuestring
<username>valuestring
<useraddr>valuestring
<userphone>valuestring
<useremail>valuestring
<useridnum>valuestring
<userotherid>valuestring
</sigdata>
<sig>base64string
</signature>
```

Figure 4.5: Generic Signature block element definition

Generic Signature Block Field Definitions

vers (**required**) The <signature> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value **1.5**.

blockref (**required**) The signature block contains one or more <blockref> fields, each of which contains the unique block name of the associated block being signed. All of the block references must appear immediately before their respective hashes (See below). The <blockref> and <hash> pairs may be repeated multiple times to sign multiple blocks. An attribute **req** is optional in the <blockref> element. If `<blockref req="true">` then removal of the referenced block invalidates the signature. If <`blockref req="false"`> then removal of the referenced block is permitted. The default, if the **req** attribute is not present, is **true**. As in other aspects of judging document and signature validity, the verifier's business rules may override the signer's assertions.

hash

(**required**) This field contains the actual hash of the respective block. Each <hash> start tag must have an attribute which specifies the algorithm used to perform the hash. The currently allowed attribute values are **md5**[2] or **sha**[4]. The `alg=` attribute is required. The use of **md5** is deprecated. Other hash algorithms may be supported in the future. It is not required that the same hash algorithm be used for each of the blockrefs in a signature block. All hashes are encoded in "network byte order", which means that the most significant bytes are leftmost (first). Note: Attribute values must be enclosed in quotes.

nonce

(**required**) This is a nonce, or one-time random number, used to "salt" the hashed data to discourage cryptanalysis attacks. See the section below on Signature Calculation. The nonce value can be any string of random ASCII characters from within the set of allowed FSML characters (see Character Encoding above), not including whitespace.

Note to Implementors:

Although any FSML character except whitespace is allowed in the <nonce> value, it is permitted, and implementors may find it convenient, to generate a random number and include it in the <nonce> field represented as a decimal integer, a floating-point number, a hexadecimal-encoded octet string, or as a base64-encoded octet string. Note that the use of this string in the hash is purely as a sequence of ASCII octets. The fact that it may have been created as an ASCII representation of a floating point number or integer, or hexadecimal number is irrelevant to its use in the hash data.

sigref

(optional) This is the block name of the <cert> block which contains the public key that can be used to verify the signature. This field, although optional, is only optional when an agreement is in place indicating that the recipient of the document does not need the certificate in order to process the document.

sigtype

(**required**) This field contains an indication of the type of signature. It must contain one of the following values for generic documents

generic This indicates a generic signature, with no attached semantics.

co-sign This indicates a co-signature, where more than one signer is signing the same document. Neither signature has precedence over the other. Each is independent.

counter-sign This indicates a counter-signature, where more than one signer is signing the same document. The counter-signer is signing the document as well as the signature(s) of the other signer(s), and is attesting to the fact that the signer has seen and agreed to, or approved the previously applied signatures.

witness This indicates a signature attesting to another signature, but not to the contents being signed.

Additional sigtype values may be defined by applications that require new signature classifications.

certissuer

(optional) This field contains the unique distinguished name of the issuer of the certificate[7]. It should only be specified if the <cert> blocks are not being sent with this document. See the description of the <certissuer> field in the <cert> block for the syntax used to specify this field.

certserial

(optional) This field contains the unique certificate serial number assigned by the issuer of the certificate. It should only be specified if the <cert> blocks are not being sent with this document.

algorithm

(**required**) This string indicates the algorithm used to sign the signature block. It may be **md5/rsa**[3] or **sha/dsa**[5] or **sha/rsa** or **sha/ecdsa**[6]. Note: Implementors of code that is used to sign FSML Electronic Documents may choose to support only one of the above possible signing algorithms. Implementors of code that is used to verify FSML Electronic Documents must support all algorithms. This ensures interoperablity. The use of md5 is deprecated.

timestamp

(optional) This field specifies the time that the document was signed. It must be in Universal time[12] (i.e., GMT) specified as CCYYMMDDThhmmssZ, where the T and Z are literal characters, and where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month, "DD" is the day, "hh" is the hour, "mm" is the minute and "ss" is the second.

username

(optional) This is an identification string containing the certificate user's name. It is optionally inserted into the document by the electronic hardware token.

This field, and the 5 following fields are optional identification data. This data is supplied by the electronic token owner to the token issuer at the time the token is initialized, but it is not certified to be correct or accurate by the token issuer. The data is inserted into the electronic token when the token is initialized, and may also be corrected or updated later by the issuer using administrative token functions and passwords.

This data is then inserted, under control of the user, into the document by the electronic token, however the data cannot be changed or deleted by the user once the document is created. The user may select, when writing a document, which of the 6 identification fields are to be inserted into the document, in any combination, or may select none of them.

useraddr

(optional) This is an identification string containing the certificate user's address. It is optionally inserted into the document by the electronic hardware token.

userphone

(optional) This is an identification string containing the certificate user's phone number. It is optionally inserted into the document by the electronic hardware token.

useremail

(optional) This is an identification string containing the certificate user's email address. It is optionally inserted into the document by the electronic hardware token.

useridnum

(optional) This is an identification string containing the certificate user's identification number. It is optionally inserted into the document by the electronic hardware token.

userotherid

(optional) This is an identification string containing any user identification the user wishes (e.g., company name). It is optionally inserted into the document by the electronic hardware token.

location

(optional) This field specifies location/country where the document was signed. The location is used to define the jurisdiction where the document was legally considered to have been signed. It is only used in circumstances where that matters.

sig (**required**) This is a base64 encoding of the actual signature data. For certain algorithms, the field is split into two portions using a colon `":"`. For DSA or ECDSA, the field contains the two portions of a DSA signature as r:s, where r and s are long base64 strings. For RSA, only a single string is specified, with no colon separator. All signatures are encoded in "network byte order", which means that the most significant bytes are leftmost (first).

Signature Calculation

The calculation of the Signature is performed as follows...

1. The <nonce> value is created as a random number. The nonce value can be any string of random ASCII characters from within the set of allowed FSML characters (see Character Encoding above) not including whitespace.

   Note to Implementors:

   Although any FSML character except whitespace is allowed in the <nonce> value, it is permitted, and implementors may find it convenient, to generate a random number and include it in the <nonce> field represented as a decimal integer, a floating-point number, a hexadecimal-encoded octet string, or as a base64-encoded octet string. Note that the use of this string in the hash is purely as a sequence of ASCII octets. The fact that it may have been created as an ASCII representation of a floating point number or integer, or hexadecimal number is irrelevant to its use in the hash data.

2. The <nonce> value is logically prepended to the subject block contents before hashing. This includes the tag string "<nonce>" — e.g., if the nonce value is 12345, the characters `<nonce>12345` are logically prepended to the subject block before hashing.

3. The hash is calculated using the contents of the subject block, (with the <nonce> prepended) including the block start tag and block end tag, with the exception of all carriage returns, line feeds, and trailing spaces on a line. Leading and embedded spaces in a line are included in the hash. SGML entities (i.e., character names enclosed between an ampersand and a semicolon) are left untranslated when hashing.

4. The resulting hash value is inserted into the <hash> entry (as base64-encoded ASCII) in the signature block. If the signature algorithm is DSA or ECDSA, the hash may be inserted without additional padding or encoding. If the signature algorithm is RSA, the hash should be padded and encoded using the specifications in the PKCS#1[13] standard.

5. Steps 2 through 4 are repeated for each block to be signed.

6. A second hash calculation is performed on the contents of the <sigdata> sub-block, which contains the previously calculated hashes, their block references, and the <nonce>. This should include all characters between the <sigdata> tag and the </sigdata> tag, not including the tags themselves, again omitting all carriage returns, line feeds, and trailing spaces. If the signature algorithm is DSA or ECDSA, the hash may be inserted without additional padding or encoding. If the signature algorithm is RSA, the hash should be padded and encoded using the specifications in the PKCS#1[13] standard. This second hash is then encrypted using the private key. The result is the signature which is inserted (as base64 encoded ASCII) into the signature block as the value for the <sig> tag.

Signature Verification

The verification of the Signature is performed as follows...

1. The following steps are repeated for each block referenced by a <blockref> tag in the signature. If the referenced block is not present, and `<blockref req="false">` was specified, the block is assumed to have been detached. The following steps are not performed for this block, and this block's absence is not considered to invalidate the document.

   (a) The <nonce> value from the signature block is logically prepended to the referenced blocks contents before hashing. This includes the tag string "<nonce>" — e.g., if the nonce value is 12345, the characters `<nonce>12345` are logically prepended to the referenced blocks contents before hashing.

   (b) A hash is calculated using the contents of the referenced block, (with the <nonce> prepended) including the block start tag and block end tag, with all characters in between, with the exception of all carriage returns, line feeds, and trailing spaces on a line. Leading and embedded spaces in a line are included in the hash. SGML entities (i.e., character names enclosed between an ampersand and a semicolon) are left untranslated when hashing. The hash algorithm to be used is specified in the `hash=` attribute in the <hash> tag for the referenced block.

   (c) The resulting hash value is compared to the <hash> entry in the signature block.

   (d) If the hashes do not match exactly, the signature fails verification.

2. The contents of the <sig> field are processed using the public key found by following the <sigref> tag, which will point (by name) to a <cert> block. Parsing of the <certdata> field in this block may be required to extract the public key — e.g. an X.509 certificate parser may be required. The signature algorithm to be used is specified in the <algorithm> field.

3. A second hash calculation is performed on the contents of the <sigdata> sub-block, which contains the previously calculated hashes, their block references, and the <nonce>. This should include all characters between the <sigdata> tag and the </sigdata> tag, not including the tags themselves, again omitting all carriage returns, line feeds, and trailing spaces. The hash algorithm to be used is specified in the <algorithm> field.

4. The processed <sig> field is compared to the hash calculated in the previous step. If this comparison fails, the signature fails verification. If the comparison succeeds, the signature has verified successfully.

4.4.3 Generic Certificate Block Definition

This block contains an encoded X.509 certificate[7].

```
<cert>
<blkname>namestring
<crit>true
<vers>1.5
<certtype>valuestring
<certissuer>valuestring
<certserial>number
<certdata>base64string
</cert>
```

Figure 4.6: Generic Certificate block element definition

Generic Certificate Block Field Definitions

blkname (**required**) The <blkname> field in a <cert> is slightly different from the “generic” <blkname>. Since the <cert> block is signed by the authority issuing the electronic token, and is probably stored in the token, it is not changeable at runtime by FSML generating software. Thus the <blkname> chosen must be guaranteed to be unique for all subsequent documents. It is recommended (but not required) that a block naming convention be used to allow this.

vers (**required**) The <cert> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value **1.5**.

certtype (**required**) This field indicates the type of certificate contained in the block. The possible values are **x509v1** or **x509v3**. This value must correspond to the data in the actual certificate contained in the <certdata> field.

certissuer (**required**) This field contains the unique distinguished name of the issuer of the certificate. The certificate issuer string uses the fields from the distinguished name in the ASN.1 X509 certificate, separated by slashes, and using a TAG= identification of the name field type. The different name fields use the following identification tags:

Country	C=
DMDName	DMD=
Commonname	CN=
Orgname	O=
Orgunit	OU=
Title	T=

Thus, an example of an issuer string would be...

```
/C=US/ST=New York/O=FIRSTBANK_ANYTOWN/OU=checking/
```

This value must correspond to the data in the actual certificate contained in the <certdata> field. Although X.509 distinguished names allow additional fields, FSML only supports the ones named above.

certserial (**required**) This field contains the unique certificate serial number assigned by the issuer of the certificate. This value must correspond to the data in the actual certificate contained in the <certdata> field.

certdata (**required**) This contains the base64-encoded binary value of the ASN.1 DER[14][15] encoded X.509 certificate.

4.4.4 Generic Attachment Block Definition

This block contains any document that is to be attached to the FSML Electronic Document (e.g., a Remittance notice, Contract, etc.).

```
<attachment>
<crit>false
<vers>1.0
<blkname>namestring
<astatus>valuestring
<adata encoding="namestring">
...
</adata>
</attachment>
```

Figure 4.7: Generic Attachment block element definition

astatus (optional) This field indicates whether the attachment is temporary or permanent. A temporary attachment is intended to be transmitted from the document creator to the first recipient. It is stripped before transmission to any subsequent recipients. A permanent attachment is intended to be kept with the document permanently and should not be removed by any recipient of the document. The contents of the field may be the word **temporary** or the word **permanent**. If the field is omitted, it defaults to **temporary**. Note: The <astatus> field is advisory only. An FSML document is not considered invalid if it is received by later recipients containing a **temporary** attachment, nor is the document invalid if a receiver strips off a **permanent** attachment.

adata (**required**) Any data may be contained in the Attachment block, between the <adata> and </adata> tags.

The `encoding=` attribute for the <adata> tag is used to specify the encoding method for the data in the sub-block. It can have the following values:

mime

If the **mime** encoding value is selected, then the following three MIME headers are required to be placed in the next 3 lines of the <adata> sub-block, immediately followed by a blank line. The header lines may appear in any order.

```
Mime-Version 1.0
Content-Type: aaaaaa/bbbbbbbb
Content-Transfer-Encoding: xxxxx
```

Any legal MIME header values may be used for aaaaaa or bbbbbbbb [10][16].

The 'xxxxx' string used to specify the Content-Transfer-Encoding value must be one of

- 7bit
- base64
- quoted-printable

If the contents of the attached document cannot be encoded using the FSML document formatting rules, described earlier, then the Content-Transfer-Encoding specification base64 should be used to "armor" the document against E-mail systems.

In addition, the encoded document may not contain the ASCII string <`/adata`> so that the FSML parser will not interpret any portion of the attached document as the ending SGML tag for the <adata> sub-block.

The actual encoded data follows the three MIME headers, separated by an empty line (i.e., a line containing only a new-line sequence, with no other characters).

An example of a MIME-encoded <attachment> block:

```
<attachment>
<blkname>att0123
<adata encoding="mime">
Mime-Version 1.0
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/C
AAAAEAAA/v///wAAAAD+////AAAAAAAAAAB7AAAA/
</adata>
</attachment>
```

text

This allows a simple ASCII document to be inserted as an attached document without need for MIME headers or encoding/decoding software. This attribute value can only be used if the attached document inside the <adata> sub-block conforms to the FSML document formatting rules. If the `encoding` attribute is not specified, it defaults to the `text` value.

4.4.5 Generic Message Block Definition

This block contains error messages and return information that indicates the reason that the attached FSML Document was not processed successfully or it may contain other information about the attached document.

```
<message>
<blkname>namestring
<crit>true
<vers>1.0
<retcode>valuestring
<msgtext>valuestring
<msgdata>
...
</msgdata>
</message>
```

Figure 4.8: Generic Message block element definition

Generic Message Block Field Definitions

retcode (**required**) This field contains a return code indicating the reason why the attached document was returned.

msgtext (**required**) This field contains a textual message explaining why the document was returned.

msgdata (optional) This field contains any other data that may be associated with the message — e.g., a report or bank statement.

4.4.6 Private Block Types

To support extension of the FSML specification, agreeing parties (document creators and document receivers) may agree to support Private Block Types. These are blocks with block start and end tags that are not defined explicitly in the FSML specification, but are created and agreed to by all communicating parties. All private block types must have start and end tags that begin with the character string `"<x:"` to distinguish them from standard block types. (e.g., <x:myblock>).

Rules for processing such blocks are as follows...

- If the private block type is not recognized by the receiving program, and the <crit> field in the block is true, or missing (default is true), then the entire document must be rejected.
- If the private block type is not recognized by the receiving program, and the <crit> field in the block is false, then the contents of the block up to the end tag is to be skipped and ignored by the receiving program.

The rules for creating such blocks are...

- If the private block is intended to be processed by the first (or nth) recipient(s) only, then the block should either have a <crit> value of false, or should be detachable by having the `<blockref req="false">` in all <signature> blocks that sign it, and the block should be detached by the first or (nth) recipient.
- The ultimate recipient will then ignore the block (if <crit> is false) or never receive it.

```
<x:name>
<blkname>namestring
<crit>true
<vers>vstring

 (Other tags to be defined by application designers)

</x:name>
```

Figure 4.9: Private block type element definition

Private Block Field Definitions Private blocks may contain any fields that the Private block definers choose. Field tag names may be chosen from those already defined for other FSML blocks, or may be new. Existing field tag names should be used only if they have the same encoding and general semantics as defined in FSML.

The <blkname>, <crit>, and <vers> fields are to be defined and used in the same manner as with all other FSML blocks.

Combining Documents

As an FSML document passes through the various processing steps at different institutions, information will need to be added to and removed from the document. This section defines a technique for combining documents such that allowable changes may be made to the document while preserving the integrity of the digital signature mechanism.

To add new information to a document, the existing document is enclosed in a <fsml-doc> tag structure, which may also enclose new blocks containing the new information. New <signature> blocks may also be contained in the new information and may sign blocks in the inner nested documents. Each new, surrounding, <fsml-doc> must also have a new <action> block, and `type` attribute, and the <action> block and `type` belonging to the outermost <fsml-doc> are used by the receiving system to determine the method used to process the modified document.

When combining original FSML documents into a larger, compound document (e.g., combining checks into a deposit), the names of the original blocks may not be unique. A document combining process must be used to handle naming conflicts when a number of documents are being combined (i.e., embedded) into a new document.

The document combining process is as follows:

1. All of the original <fsml-doc> elements are enclosed in a single new <fsml-doc> element. The original docname attributes are kept with the same contents, unless all of the combined document names are not unique. If they are not unique, new, unique names should be assigned by the combining software.
2. Any time a block name reference is required to refer to a block which is not part of the same <fsml-doc> as the one containing the reference (i.e., inter-document references) then the reference consists of the DOCNAME of the <fsml-doc> element concatenated with a period "." and then with the <blkname> of the inner block being referred to.

   This is extended if the nesting is continued to more than two levels — e.g., `'outerdoc.innerdoc.block'`.
3. Any block references inside a given <fsml-doc> must use the block name without any qualifying document name, to ensure that future document combining will not be prevented.

As an example:

If there are two original documents:

```
  <fsml-doc docname="doc1">
    <attachment>
       <blkname>block1
       ....
    </attachment>
  </fsml-doc>

  <fsml-doc docname="doc2">
    <attachment>
       <blkname>block1
       ....
    </attachment>
  </fsml-doc>
```

Figure 5.10: Example of combining documents - before

When they are combined, the result is:

```
<fsml-doc docname="newdoc">

  <fsml-doc docname="doc1">
    <attachment>
       <blkname>block1
       ...
    </attachment>
  </fsml-doc>

  <fsml-doc docname="doc2">
    <attachment>
       <blkname>block1
       ...
    </attachment>
  </fsml-doc>

  <signature>
  <blockref>doc1.block1
  ...
  <blockref>doc2.block1
  ...
  </signature>

</fsml-doc>
```

Figure 5.11: Example of combining documents - after

Any outer-document references to the <attachment> block in the first document would be `'doc1.block1'`, and the <attachment> block in the second document would be `'doc2.block1'`. References inside doc1 to any blocks in doc1 must still use the original, single level names. Similarly for internal references inside doc2.

This is extended if the nesting is continued to more than two levels — e.g., `'outerdoc.innerdoc.block'`.

echeck Specific FSML

This chapter describes the use of FSML for Electronic Checks and associated documents. All of the Generic FSML formats and rules are applicable, except that new block types are added, and certain blocks have additional fields (e.g., signature blocks).

6.1 Electronic Check Document type Definitions

The **type** attribute in the <fsml-doc> tag must be one of the following values for an echeck or related document ...

check This indicates that the document is a signed electronic check, which is usable as a payment.

endcheck This indicates that the document is a signed, endorsed check, which is usable for deposit or transfer to a third party.

certcheck This indicates that the document is a signed, certified check, which is usable as a payment and contains bank certification information certifying that the funds are available and held.

deposit This indicates that the document is a group of one or more endorsed, signed electronic checks, combined with one or more deposit slips, intended as a deposit.

return This indicates that the document is a deposit item being returned by the bank for some reason. It will contain a <message> block indicating the reason for the return.

presentment This indicates that the document is an echeck item being transmitted from one bank to another for purposes of clearing or presentment. This type of document is only valid for such inter-bank transmission. It need not be created by or processed by payer or payee applications.

cgroup This indicates that the document contains a <cashletter> block and a number of bundle and presentment subdocuments being transmitted from one bank to another for purposes of clearing or presentment. This type of document is only valid for such inter-bank transmission. It need not be created by or processed by payer or payee applications.

6.2 Electronic Check Document Global Structure

6.2.1 BNF Structure of FSML Electronic Check Documents

The following is an Extended BNF[8][9] description of the global block structure of FSML echeck Electronic Documents.

BNF Meta-Notation

The meta-symbols of BNF are:

::=	meaning “is defined as”
|	meaning “or”
[ ]	used to enclose optional items
”	used to enclose characters or strings that represent themselves
{ }	used to enclose repeated items (repeated zero or more times)
< >	used to enclose specific FSML tags.
< ( ) >	used to specify FSML blocks.

Names not enclosed in any of the above bracket symbols are called *nonterminals* and are used to define symbols internal to the BNF specification only.

Note: Blocks are not required to be in the exact order specified below, except that the <action> block must always appear as the first block in any <fsml-doc>.

Subgroup BNF definitions

First, some lower-level nonterminal definitions of signature groups, which contain <signature> blocks and their associated <account> and <cert> blocks.

```
acct_sig_group    ::=  <(signature)> <(account)> [<(cert)>]

cert_sig_group    ::=  <(signature)> [<(cert)>]

bank_sig_group    ::=  <(signature)> [<(cert)>]

tell_sig_group    ::=  <(signature)> <(account)> [<(cert)>]
```

The acct_sig_group includes the <signature>, <account>, and <cert> blocks for a user signature that requires an account — e.g., check signer, depositor.

The cert_sig_group includes the <signature>, and <cert> blocks for a user signature that does not require an account — e.g., check endorser, or Generic FSML document.

The bank_sig_group includes the <signature> and <cert> blocks used by the bank to sign the user's account and certificate, or a returned item, or some other document signed by the Bank.

The tell_sig_group includes the <signature>, <account>, and <cert> blocks for a teller's or bank officer's signature that requires an account — e.g., certified check.

The cert blocks are shown above as being optional, since...

- They may be referred to but not included in the FSML Document (by prior arrangement only).
- A second cert block which contains a second redundant instance of the same certificate need not be added.

Signed Electronic Check BNF definitions

```
signed_echeck_doc ::=
        '<fsml-doc docname="' dname '" type="check">'
        <(action)>
        <(check)>
        acct_sig_group  bank_sig_group
      { acct_sig_group  bank_sig_group }
      { <(attachment)> }
      { <(invoice)> }
        '</fsml-doc>'
```

If a check is co-signed, an additional `acct_sig_group` is added by the co-signer. The co-signer's signature signs the same blocks as the original signer. If a check is counter-signed, an additional `acct_sig_group` is added by the counter-signer. The counter-signer's signature signs the same blocks as the original signer, along with the original signer's <signature> block, which indicates that the counter-signer is attesting and agreeing with the original signature.

Certified, Signed Electronic Check BNF definitions

```
certified_echeck_doc ::=
      '<fsml-doc docname="' dname '" type="certcheck">'
      <(action)>
      <(certification)> tell_sig_group bank_sig_group
      signed_echeck_doc
      '</fsml-doc>'
```

Endorsed, Signed Electronic Check BNF definitions

```
check_doc_list ::=
    signed_echeck_doc | certified_echeck_doc


endorsed_echeck_doc   ::=
   '<fsml-doc docname="' dname '" type="endcheck">'
    <(action)>
    <(endorsement)>
    cert_sig_group  <(cert)> | acct_sig_group
  { cert_sig_group  <(cert)> }
    check_doc_list
  { <(attachment)> }
    '</fsml-doc>'
```

If an endorsement document contains a check that has either the <paytoacct> or <paytocustno> fields in the <check> block, then the endorsement document must contain an acct_sig_group instead of a cert_sig_group and the associated bank certificate. If a check is co-endorsed, an additional cert_sig_group and possibly a bank cert (if not already present) is added by the co-endorser. The co-endorsed signature signs the same blocks as the original endorser. If a check is counter-endorsed, an additional cert_sig_group and possibly a bank cert (if not already present) is added by the counter-endorser. The counter-endorser's signature signs the same blocks as the original endorser, along with the original endorser's <signature> block, which indicates that the counter-endorser is attesting and agreeing with the original signature.

Note: The <cert> block specified after the cert_sig_group above is the bank's certificate specifying the bank's public key for verification of the bank's signature on the endorser's x509 certificate.

Deposited Electronic Check BNF definitions

```
deposit_doc ::=
   '<fsml-doc docname="' dname '" type="deposit">'
    <(action)>
    <(deposit)>
  { <(deposit)> }
    acct_sig_group  bank_sig_group
  { acct_sig_group  bank_sig_group  }
    endorsed_echeck_doc
  { endorsed_echeck_doc }
   '</fsml-doc>'
```

Note: The deposit_doc can have multiple <deposit> blocks if it contains multiple account_sig_groups to specify the account data for each account being deposited into.

Returned Electronic Check BNF definitions

```
returned_item_doc ::=
   '<fsml-doc docname="' dname '" type="return">'
    <(action)>
    endorsed_echeck_doc | check_doc_list
  { bankstamp }
    '</fsml-doc>'
```

In a return, bankstamps are added by each bank which the return item has passed through during processing. Each bank stamp is added as another <bankstamp> block inside the original return document. New <bankstamp> blocks must be added after the earlier ones. If a bank has to change the contents of the <action> block, then the original return is wrapped in a new document and the bankstamp is placed in the new document, along with the new <action> block.

Presentment Item BNF definitions

```
presentment_item_doc ::=
   '<fsml-doc docname="' dname '" type="presentment">'
    <(action)>
    endorsed_echeck_doc
  { bankstamp }
    '</fsml-doc>'
```

```
modified_presentment_item_doc ::=
   '<fsml-doc docname="' dname '" type="presentment">'
    <(action)>
    presentment_item_doc | modified_presentment_item_doc
  { bankstamp }
    '</fsml-doc>'
```

In a presentment, bankstamps are added by each bank which the presentment item has passed through during processing. Each bank stamp is added as another <bankstamp> block inside the original presentment document. New <bankstamp> blocks must be added after the earlier ones. If a bank has to change the contents of the <action> block, then the original presentment is wrapped in a new document (modified_presentment_item_doc, above) and the bankstamp is placed in the new document, along with the new <action> block.

Cashletter group BNF definitions

```
bundle_doc ::=
   '<fsml-doc docname="' dname '" type="bundle">'
    <(bundle)>
    presentment_item_doc | modified_presentment_item_doc
    { presentment_item_doc }
    '</fsml-doc>'
```

```
cgroup_doc ::=
   '<fsml-doc docname="' dname '" type="cgroup">'
    <(action)>
    <(cashletter)>
    bundle-doc
  { bundle-doc }
   '</fsml-doc>'
```

Note: A cgroup document contains a cashletter, and a number of bundles, each of which contain a number of presentment items. Bundle documents are not usable FSML documents outside of a cgroup. They are used for purposes of grouping presentment items only. No signatures or certificates are added to cgroups or bundles, and no action blocks are needed in bundle documents.

6.2.2 Global Structure - Signed Electronic Check

Global Block Structure Detail - Signed Electronic Check				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action>	C1	payment/process		
<check>	C2			
<signature>	C3	hash of <action>	C1	signer’s signature
		hash of <check>	C2	
		hash of signer’s <account>	C4	
		hashes of any <attachments>	C6	
		hash of any <invoice>		
		reference to <account>	C4	
		<sigtype>check		
<account>	C4	issuer/serial of <cert>	C5	signer’s account block
<cert>	C5			signer’s certificate
<attachment>	C6			attachments, invoice
<signature>	C7	hash of signer’s <account> block	C4	bank’s signature
		hash of signer’s <cert> block	C5	
		reference to bank’s <cert>	C8	
		<sigtype>bankacct		
<cert>	C8			bank’s certificate

```
<fsml-doc docname="C" type="check">
  <action>     <blkname>C1 ... </action>
  <check>      <blkname>C2 ... </check>
  <signature>  <blkname>C3 ... </signature>
  <account>    <blkname>C4 ... </account>
  <cert>       <blkname>C5 ... </cert>
  <attachment> <blkname>C6 ... </attachment>  (optional)
  <signature>  <blkname>C7 ... </signature>
  <cert>       <blkname>C8 ... </cert>
</fsml-doc>
```

6.2.3 Global Structure - Co-Signed Electronic Check

Global Block Structure Detail - Co-signed Electronic Check				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
signed echeck	Cn			original echeck
<signature>	S1	hash of <action> hash of co-signer's <account> hash of any original <invoice> hash of any original <attachment> hash of original <check> reference to co-signer's <account> <sigtype>co-sign	C1 S2 C6 C2 S2	co-signer's signature
<account>	S2	issuer/serial of co-signer's <cert>	S3	co-signer's account block
<cert>	S3			co-signer's certificate
<signature>	S4	hash of co-signer's <account> block hash of co-signer's <cert> block reference to bank's <cert> <sigtype>bankacct	S2 S3 S5	bank's signature
<cert>	S5			bank's certificate

```
<fsml-doc docname="S" type="check">
  <action>     <blkname>C1 ... </action>
  <check>      <blkname>C2 ... </check>
  <signature>  <blkname>C3 ... </signature>
  <account>    <blkname>C4 ... </account>
  <cert>       <blkname>C5 ... </cert>
  <attachment> <blkname>C6 ... </attachment>  (optional)
  <signature>  <blkname>C7 ... </signature>
  <cert>       <blkname>C8 ... </cert>
  <signature>  <blkname>S1 ... </signature>
  <account>    <blkname>S2 ... </account>
  <cert>       <blkname>S3 ... </cert>
  <signature>  <blkname>S4 ... </signature>
  <cert>       <blkname>S5 ... </cert>
</fsml-doc>
```

6.2.4 Global Structure - Counter-signed Electronic Check

Global Block Structure Detail - Counter-signed Electronic Check				
Blocks	Block	Block References	Ref	Block Function
	No.	and Contents	Block	
signed echeck	Cn			original echeck
<signature>	S1	hash of <action>	C1	counter-signer’s signature
		hash of counter-signer’s <account>	S2	
		hash of any original <invoice>		
		hash of any original <attachment>	C6	
		hash of original <check>	C2	
		hash of original <signature>	C3	
		reference to counter-signer’s <account>	S2	
		<sigtype>counter-sign		
<account>	S2	issuer/serial of counter-signer’s <cert>	S3	co-signer’s account block
<cert>	S3			counter-signer’s certificate
<signature>	S4	hash of counter-signer’s <account> block	S2	bank’s signature
		hash of counter-signer’s <cert> block	S3	
		reference to bank’s <cert>	S5	
		<sigtype>bankacct		
<cert>	S5			bank’s certificate

```
<fsml-doc docname="S" type="check">
  <action>     <blkname>C1 ... </action>
  <check>      <blkname>C2 ... </check>
  <signature>  <blkname>C3 ... </signature>
  <account>    <blkname>C4 ... </account>
  <cert>       <blkname>C5 ... </cert>
  <attachment> <blkname>C6 ... </attachment>  (optional)
  <signature>  <blkname>C7 ... </signature>
  <cert>       <blkname>C8 ... </cert>
  <signature>  <blkname>S1 ... </signature>
  <account>    <blkname>S2 ... </account>
  <cert>       <blkname>S3 ... </cert>
  <signature>  <blkname>S4 ... </signature>
  <cert>       <blkname>S5 ... </cert>
</fsml-doc>
```

6.2.5 Global Structure - Certified, Signed Electronic Check

Global Block Structure Detail - Certified Signed Electronic Check				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action>	Q1	payment/process		
<certification>	Q2			certification block
<signature>	Q3	hash of <action>	Q1	certifier's signature
		hash of original <check>	C2	
		hash of original <signature>	C3	
		hash of new <certification>	Q2	
		reference to bank's <account>	Q5	
		<sigtype>certification		
<cert>	Q4			certifier's certificate
<account>	Q5	issuer/serial of <cert>	Q4	certifier's account block
<signature>	Q6	hash of certifier's <account> block	Q5	bank's signature
		hash of certifier's <cert> block	Q4	
		reference to bank's <cert>	Q7	
		<sigtype>bankacct		
<cert>	Q7			bank's certificate
signed echeck	Cn			original echeck

```
<fsml-doc docname="Q" type="certcheck">
  <action>       <blkname>Q1 ... </action>
  <certification><blkname>Q2 ... </certification>
  <signature>    <blkname>Q3 ... </signature>
  <cert>         <blkname>Q4 ... </cert>
  <account>      <blkname>Q5 ... </account>
  <signature>    <blkname>Q6 ... </signature>
  <cert>         <blkname>Q7 ... </cert>
  <fsml-doc docname="C" type="check">
    <action>       <blkname>C1 ... </action>
    <check>        <blkname>C2 ... </check>
    <signature>    <blkname>C3 ... </signature>
    <account>      <blkname>C4 ... </account>
    <cert>         <blkname>C5 ... </cert>
    <attachment>   <blkname>C6 ... </attachment>  (optional)
    <signature>    <blkname>C7 ... </signature>
    <cert>         <blkname>C8 ... </cert>
  </fsml-doc>
</fsml-doc>
```

6.2.6 Global Structure - Endorsed Electronic Check

Global Block Structure Detail - Endorsed Electronic Check				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action>	E1	payment/process		
<endorsement>	E2			endorsement block
<signature>	E3	hash of <action>	E1	endorser’s signature
		hash of original <check>	C2	
		hash of original <signature>	C3	
		hash of new <endorsement>	E2	
		hashes of any new <attachments>	E5	
		reference to endorser’s <cert>	E4	
		<sigtype>endorsement		
<cert>	E4			endorser’s certificate
<attachment>	E5			new attachments
<cert>	E6			bank’s certificate
signed echeck	Cn			original echeck

```
<fsml-doc docname="E" type="endcheck">
  <action>     <blkname>E1 ... </action>
  <endorsement><blkname>E2 ... </endorsement>
  <signature>  <blkname>E3 ... </signature>
  <cert>       <blkname>E4 ... </cert>
  <attachment> <blkname>E5 ... </attachment>  (optional)
  <cert>       <blkname>E6 ... </cert>
  <fsml-doc docname="C" type="check">
    <action>     <blkname>C1 ... </action>
    <check>      <blkname>C2 ... </check>
    <signature>  <blkname>C3 ... </signature>
    <account>    <blkname>C4 ... </account>
    <cert>       <blkname>C5 ... </cert>
    <attachment> <blkname>C6 ... </attachment>  (optional)
    <signature>  <blkname>C7 ... </signature>
    <cert>       <blkname>C8 ... </cert>
  </fsml-doc>
</fsml-doc>
```

6.2.7 Global Structure - Co-endorsed Electronic Check

Global Block Structure Detail - Co-endorsed Electronic Check				
Blocks	Block	Block References	Ref	Block Function
	No.	and Contents	Block	
endorsed echeck	En			original endorsed echeck
<signature>	M1	hash of <action>	E1	co-endorser's signature
		hash of original <endorsement>	E2	
		hash of original <check>	C2	
		hash of original <signature>	C3	
		hashes of original <attachments>	E5	
		reference to co-endorser's <cert>	M2	
		<sigtype>co-endorse		
<cert>	M2			co-endorser's certificate
<cert>	M3			bank's certificate

```
<fsml-doc docname="E" type="endcheck">
  <action>     <blkname>E1 ... </action>
  <endorsement><blkname>E2 ... </endorsement>
  <signature>  <blkname>E3 ... </signature>
  <cert>       <blkname>E4 ... </cert>
  <attachment> <blkname>E5 ... </attachment>  (optional)
  <cert>       <blkname>E6 ... </cert>
  <signature>  <blkname>M1 ... </signature>
  <cert>       <blkname>M2 ... </cert>
  <cert>       <blkname>M3 ... </cert>
  <fsml-doc docname="C" type="check">
    <action>     <blkname>C1 ... </action>
    <check>      <blkname>C2 ... </check>
    <signature>  <blkname>C3 ... </signature>
    <account>    <blkname>C4 ... </account>
    <cert>       <blkname>C5 ... </cert>
    <attachment> <blkname>C6 ... </attachment>  (optional)
    <signature>  <blkname>C7 ... </signature>
    <cert>       <blkname>C8 ... </cert>
  </fsml-doc>
</fsml-doc>
```

6.2.8 Global Structure - Counter-endorsed Electronic Check

Global Block Structure Detail - Counter-endorsed Electronic Check				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
endorsed echeck	En			original endorsed echeck
<signature>	M1	hash of <action>	E1	co-endorser's signature
		hash of original <endorsement>	E2	
		hash of original endorser's <signature>	E3	
		hash of original <check>	C2	
		hash of original <signature>	C3	
		hashes of original <attachments>	E5	
		reference to co-endorser's <cert>	M2	
		<sigtype>counter-endorse		
<cert>	M2			co-endorser's certificate
<cert>	M3			bank's certificate

```
<fsml-doc docname="E" type="endcheck">
  <action>     <blkname>E1 ... </action>
  <endorsement><blkname>E2 ... </endorsement>
  <signature>  <blkname>E3 ... </signature>
  <cert>       <blkname>E4 ... </cert>
  <attachment> <blkname>E5 ... </attachment>  (optional)
  <cert>       <blkname>E6 ... </cert>
  <signature>  <blkname>M1 ... </signature>
  <cert>       <blkname>M2 ... </cert>
  <cert>       <blkname>M3 ... </cert>
  <fsml-doc docname="C" type="check">
    <action>     <blkname>C1 ... </action>
    <check>      <blkname>C2 ... </check>
    <signature>  <blkname>C3 ... </signature>
    <account>    <blkname>C4 ... </account>
    <cert>       <blkname>C5 ... </cert>
    <attachment> <blkname>C6 ... </attachment>  (optional)
    <signature>  <blkname>C7 ... </signature>
    <cert>       <blkname>C8 ... </cert>
  </fsml-doc>
</fsml-doc>
```

6.2.9 Global Structure - Deposited Electronic Check

Global Block Structure Detail - Deposited Electronic Check				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action>	D1	deposit/process		
<deposit>	D2			deposit slip
<deposit>	D2.n	deposit block for split deposits		deposit slip (may be multiple)
<signature>	D3	hash of <action>	D1	depositor's signature
		hashes of original <endorsement>	E2	
		hash of original endorser's <signature>	E3	
		hash of new <deposit> block(s)	D2	
		hash of new <deposit> block(s)	D2.n	for split deposits (may be multiple)
		hash of depositor's <account>	D4	
		reference to depositor's <account>	D4	
		<sigtype>deposit		
<account>	D4	issuer/serial of <cert>	D5	depositor's account block
<account>	D4.n	issuer/serial of <cert>	D5.n	split account block (may be multiple)
<cert>	D5			depositor's certificate
<cert>	D5.n			depositor's certificate (may be multiple)
<signature>	D6	hash of depositor's <account> block	D4	bank's signature
		hash of depositor's <cert> block	D5	
		reference to bank's <cert>	D7	
		<sigtype>bankacct		
<signature>	D6.n	hash of depositor's <account> block	D4.n	bank's signature (may be multiple)
		hash of depositor's <cert> block	D5.n	
		reference to bank's <cert>	D7	
		<sigtype>bankacct		
<cert>	D7			bank's certificate
endorsed echecks	Cn,En			original echecks, endorsements

```
<fsml-doc docname="D" type="deposit">
  <action>     <blkname>D1 ... </action>
  <deposit>    <blkname>D2 ... </deposit>
  <signature>  <blkname>D3 ... </signature>
  <account>    <blkname>D4 ... </account>
  <cert>       <blkname>D5 ... </cert>
  <signature>  <blkname>D6 ... </signature>
  <cert>       <blkname>D7 ... </cert>
  <fsml-doc docname="E" type="endcheck">
    <action>     <blkname>E1 ... </action>
    <endorsement><blkname>E2 ... </endorsement>
    <signature>  <blkname>E3 ... </signature>
    <cert>       <blkname>E4 ... </cert>
    <attachment> <blkname>E5 ... </attachment>  (optional)
    <signature>  <blkname>E6 ... </signature>
    <cert>       <blkname>E7 ... </cert>
    <fsml-doc docname="C" type="check">
      <action>     <blkname>C1 ... </action>
      <check>      <blkname>C2 ... </check>
      <signature>  <blkname>C3 ... </signature>
      <account>    <blkname>C4 ... </account>
      <cert>       <blkname>C5 ... </cert>
      <attachment> <blkname>C6 ... </attachment>  (optional)
      <signature>  <blkname>C7 ... </signature>
      <cert>       <blkname>C8 ... </cert>
    </fsml-doc>
  </fsml-doc>
</fsml-doc>
```

6.2.10 Global Structure - Returned Electronic Check

Global Block Structure Detail - Returned Electronic Check				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action> returned echecks <bankstamp>	R1 E R2	payment/return		endorsed echecks

```
<fsml-doc docname="R" type="return">
  <action>     <blkname>R1 ... </action>
  <bankstamp>  <blkname>R2 ... </bankstamp>
  <fsml-doc docname="E" type="endcheck">
    <action>     <blkname>E1 ... </action>
    <endorsement><blkname>E2 ... </endorsement>
    <signature>  <blkname>E3 ... </signature>
    <cert>       <blkname>E4 ... </cert>
    <attachment> <blkname>E5 ... </attachment>  (optional)
    <signature>  <blkname>E6 ... </signature>
    <cert>       <blkname>E7 ... </cert>
    <fsml-doc docname="C" type="check">
      <action>     <blkname>C1 ... </action>
      <check>      <blkname>C2 ... </check>
      <signature>  <blkname>C3 ... </signature>
      <account>    <blkname>C4 ... </account>
      <cert>       <blkname>C5 ... </cert>
      <attachment> <blkname>C6 ... </attachment>  (optional)
      <signature>  <blkname>C7 ... </signature>
      <cert>       <blkname>C8 ... </cert>
    </fsml-doc>
  </fsml-doc>
</fsml-doc>
```

6.2.11 Global Structure - Presentment Item

Global Block Structure Detail - Presentment Item				
Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action> echeck <bankstamp>	P1 C P2	present/process		echeck

```
<fsml-doc docname="P" type="presentment">
  <action>     <blkname>P1 ... </action>
  <bankstamp>  <blkname>P2 ... </bankstamp>
  <fsml-doc docname="C" type="check">
    <action>     <blkname>C1 ... </action>
    <check>      <blkname>C2 ... </check>
    <signature>  <blkname>C3 ... </signature>
    <account>    <blkname>C4 ... </account>
    <cert>       <blkname>C5 ... </cert>
    <attachment> <blkname>C6 ... </attachment>  (optional)
    <signature>  <blkname>C7 ... </signature>
    <cert>       <blkname>C8 ... </cert>
  </fsml-doc>
</fsml-doc>
```

6.2.12 Global Structure - Bundle document

Global Block Structure Detail - Bundle Document

Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action>	B1	present/process		
<bundle>	B2			bundle totals
Presentment Docs	Pn			echecks in bundle

```
<fsml-doc docname="B" type="bundle">
  <action>     <blkname>B1 ... </action>
  <bundle>     <blkname>B2 ... </bundle>
  <fsml-doc docname="P1" type="presentment">
  </fsml-doc>
  <fsml-doc docname="P2" type="presentment">
  </fsml-doc>
  <fsml-doc docname="P3" type="presentment">
  </fsml-doc>
</fsml-doc>
```

6.2.13 Global Structure - Cgroup document

Global Block Structure Detail - Cgroup Document

Blocks	Block No.	Block References and Contents	Ref Block	Block Function
<action>	L1	present/process		
<cashletter>	L2			cgroup totals
Bundle Docs	Bn			bundles of checks

```
<fsml-doc docname="B" type="cgroup">
  <action>     <blkname>L1 ... </action>
  <cashletter> <blkname>L2 ... </cashletter>
  <fsml-doc docname="B1" type="bundle">
  </fsml-doc>
  <fsml-doc docname="B2" type="bundle">
  </fsml-doc>
  <fsml-doc docname="B3" type="bundle">
  </fsml-doc>
</fsml-doc>
```

6.3 Electronic Check Block Definitions

This section describes the blocks, fields, and other information specific to the use of FSML for Electronic Check documents.

Each echeck Specific FSML block element starts and ends with one of the following sets of block tags:

```
Start Tag                 End Tag

 <action>                <action>
 <signature>             </signature>
 <check>                 </check>
 <deposit>               </deposit>
 <endorsement>           </endorsement>
 <certification>         </certification>
 <account>               </account>
 <cert>                  </cert>
 <attachment>            </attachment>
 <invoice>               </invoice>
 <message>               </message>
 <bankstamp>             </bankstamp>
 <bundle>                </bundle>
 <cashletter>            </cashletter>
 <x:name>               </x:name>
```

Figure 6.12: List of block elements

The block types are defined as follows:

action A block describing the action to be performed by the recipient

signature A block with the signatures and hashes of other blocks

check An electronic check

deposit A deposit slip, attached to one or more checks

endorsement An electronic endorsement, attached to a check

certification A certification, used to create a certified check

account A block containing account information

cert A public key certificate

attachment An associated document attached to an FSML document

invoice An invoice/remittance document containing payment information

message	A message in a returned document
bankstamp	A block containing processing status information
bundle	A block containing bundle totals
cashletter	A block containing cashletter totals
x:name	A Private Extension block

6.3.1 echeck Action Block Definition

echeck action blocks are the same as Generic Action blocks. The function field definition for the echeck application must be one of the following character strings ...

payment This indicates that the document is a check being sent as payment to a payee.

transfer This indicates that the document is a check being sent to a bank to be cashed and transferred. [1]

deposit This indicates that the document is a deposit containing a check or group of checks being sent to a bank for deposit.

certify This indicates that the document is a check being sent to a bank for certification.

stop This indicates that the document is a stop payment being sent to a bank.

inquiry This indicates that the document is an inquiry being sent to a bank or other Electronic document processing entity. [2]

present This indicates that the document is being transmitted from one bank to another for presentment (clearing).

re-present This indicates that the document is being transmitted from one bank to another for presentment a second time. [3]

[1] To be described in a later edition of this specification

[2] To be described in a later edition of this specification

[3] To be described in a later edition of this specification

6.3.2 echeck Signature Block Definition

The echeck Signature Block is the same as the Generic Signature Block, except for the descriptions of some of the fields.

```
<signature>
<blkname>namestring
<crit>true
<vers>1.5
<sigdata>
<blockref req="true">dnamestring
<hash alg="sha">base64string
<blockref req="true">dnamestring
<hash alg="sha">base64string
  ...
<blockref req="true">dnamestring
<hash alg="sha">base64string
<nonce>valuestring
<sigref>namestring
<sigtype>valuestring
<certissuer>namestring
<certserial>number
<algorithm>namestring
<timestamp>valuestring
<location>valuestring
<username>valuestring
<useraddr>valuestring
<userphone>valuestring
<useremail>valuestring
<useridnum>valuestring
<userotherid>valuestring
</sigdata>
<sig>base64string
</signature>
```

Figure 6.13: echeck Signature block element definition

echeck Signature Block Field Definitions

vers (**required**) The <signature> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value “1.5”.

blockref (**required**) The signature block contains one or more <blockref> fields, each of which contains the unique block name of the associated block being signed. All of the block references must appear immediately before their respective hashes (see below). The <blockref>

and <hash> pairs may be repeated multiple times to sign multiple blocks. An attribute **req** is optional in the <blockref> element. If <`blockref req="true"`> then removal of the referenced block invalidates the signature. If <`blockref req="false"`> then removal of the referenced block is permitted. The default if “req” is not present is “true”. As in other aspects of judging document and signature validity, the verifier’s business rules may override the signer’s assertions.

hash

(**required**) This field contains the actual hash of the respective block. Each <hash> start tag must have an attribute which specifies the algorithm used to perform the hash. The currently allowed attribute values are **md5**[2] or **sha**[4]. The `alg=` attribute is required. The use of **md5** is deprecated. Other hash algorithms may be supported in future. It is not required that the same hash algorithm be used for each of the blockrefs in a signature block. All hashes are encoded in “network byte order”, which means that the most significant bytes are leftmost (first). Note: Attribute values must be enclosed in quotes.

nonce

(**required**) This is a nonce, or one-time random number, used to “salt” the hashed data to discourage cryptanalysis attacks. The nonce value can be any string of random ASCII characters from within the set of allowed FSML characters (see Character Encoding above) not including whitespace.

Note to Implementors:

Although any FSML character except whitespace is allowed in the <nonce> value, it is permitted, and implementors may find it convenient, to generate a random number and include it in the <nonce> field represented as a decimal integer, a floating-point number, a hexadecimal-encoded octet string, or as a base64-encoded octet string. Note that the use of this string in the hash is purely as a sequence of ASCII octets. The fact that it may have been created as an ASCII representation of a floating point number or integer, or hexadecimal number is irrelevant to its use in the hash data.

sigref

(optional) This is the block name of the <account> block which contains a reference to the certificate block, or it is the block name of the <cert> block itself, for signatures that don’t need account blocks. This field, although optional, is only optional when an agreement is in place indicating that the recipient of the document does not need the certificate in order to process the document.

sigtype

(**required**) This field contains a indication of the type of signature. It must contain one of the following values...

check This indicates that the signer is the originator of a check, and is the only signer, or the first signer.

endorsement This indicates that the signer is the endorser of a check, and is the only endorser, or the first endorser.

deposit This indicates that the signer is the depositor of a check.

co-sign This indicates a co-signature, where more than one signer is signing the same document. Neither signature has precedence over the other. Each is independent.

counter-sign This indicates a counter-signature, where more than one signer is signing the same document. The counter-signer is signing the document as well as the signature of the other signers, and is attesting to the fact that the signer has seen and agreed to the previous signatures.

co-endorse This indicates a co-endorsement, where more than one endorser is signing the same document. Neither signature has precedence over the other. Each is independent.

counter-endorse This indicates a counter-endorsement, where more than one endorser is signing the same document. The counter-endorser is signing the document as well as the signature of the other endorsers, and is attesting to the fact that the signer has seen and agreed to the previous endorsement signatures.

log-signature This indicates that the signature is generated by the Electronic Checkbook when signing the checkbook log.

bankacct This indicates that the signature is generated by the Bank and is signing the account credentials (i.e., The customers <account> and <cert> blocks).

bank This indicates that the signature is generated by the Bank and is signing a bank-generated document, such as a return. <account> and <cert> blocks).

certification This indicates that the signature is generated by the Bank and is signing a certified check.

endorse-over This indicates that the signature is an endorsement to a third party.

certissuer (optional) This field contains the unique distinguished name of the issuer of the certificate[7]. It should only be specified if the <account> and <cert> blocks are not being sent with this document, and only when the blocks being signed do not require an account — e.g., an endorsement. See the description of the <certissuer> field in the <cert> block for the syntax used to specify this field.

certserial (optional) This field contains the unique certificate serial number assigned by the issuer of the certificate. It should only be specified if the <account> and <cert> blocks are not being sent with this document, and only when the blocks being signed do not require an account — e.g., an endorsement.

algorithm (**required**) This string indicates the algorithm used to sign the signature block. It may be **md5/rsa**[3] or **sha/dsa**[5] or **sha/rsa** or **sha/ecdsa**[6]. Note: Implementors of code that is used to sign FSML Electronic Documents may choose to support only one of the above possible signing algorithms. Implementors of code that is used to verify FSML Electronic Documents must support all algorithms. This ensures interoperablity. The use of md5 is deprecated.

timestamp (optional) This field specifies the time that the document was signed. It must be in Universal time (i.e., GMT) specified as CCYYMMDDThhmmssZ, where the T and Z are literal characters, and where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month, "DD" is the day, "hh" is the hour, "mm" is the minute and ss is the second[12].

location (optional) This field specifies location/country where the document was signed. The location is used to define the jurisdiction where the document was legally considered to have been signed. It is only used in circumstances where that matters.

username (optional) This is an identification string containing the account user's name. It is optionally inserted into the check by the Electronic Checkbook hardware token.

This field, and the 5 following fields are considered the electronic equivalent of the data usually printed on a paper check by the Check Printing company. This data is supplied by the checkbook owner to the bank at the time the electronic checking account is established but it is not certified to be correct or accurate by the bank. The data is inserted into the Electronic Checkbook when the Checkbook is initialized at the bank, and may also be corrected or updated later by the bank using administrative checkbook functions and passwords.

This data is then inserted, under control of the user, into the check by the Electronic Checkbook, however the data cannot be changed or deleted by the user once the check is created. It therefore supplies a form of identification sometimes required by check guarantee organizations or merchants. The user may select, when writing a check, which of the 6 identification fields are to be inserted into the check, in any combination, or may select none of them.

useraddr (optional) This is an identification string containing the account user's address. It is optionally inserted into the check by the Electronic Checkbook hardware token.

userphone (optional) This is an identification string containing the account user's phone number. It is optionally inserted into the check by the Electronic Checkbook hardware token.

useremail (optional) This is an identification string containing the account user's email address. It is optionally inserted into the check by the Electronic Checkbook hardware token.

useridnum (optional) This is an identification string containing the account user's identification number. It is optionally inserted into the check by the Electronic Checkbook hardware token.

userotherid (optional) This is an identification string containing any user identification the user wishes (e.g., company name). It is optionally inserted into the check by the Electronic Checkbook hardware token.

sig (**required**) This is a base64 encoding of the actual signature data. For certain algorithms, the field is split into two portions using a colon `":"`. For DSA or ECDSA, the field contains the two portions of the signature as r:s, where r and s are long base64 strings. For RSA, only a single string is specified, with no colon separator. All signatures are encoded in "network byte order", which means that the most significant bytes are leftmost (first).

Signature Calculation

The calculation of the Signature is performed as follows...

1. The <nonce> value is created (by the electronic checkbook) as a random number. The nonce value can be any string of random ASCII characters from within the set of allowed FSML characters (see Character Encoding above) not including whitespace.

   Note to Implementors:

   Although any FSML character except whitespace is allowed in the <nonce> value, it is permitted, and implementors may find it convenient, to generate a random number and include it in the <nonce> field represented as a decimal integer, a floating-point number, a hexadecimal-encoded octet string, or as a base64-encoded octet string. Note that the use of this string in the hash is purely as a sequence of ASCII octets. The fact that it may have been created as an ASCII representation of a floating point number or integer, or hexadecimal number is irrelevant to its use in the hash data.

2. The <nonce> value is logically prepended to the subject block contents before hashing. This includes the tag string "<nonce>" — e.g., if the nonce value is 12345, the characters `<nonce>12345` are logically prepended to the subject block before hashing.

3. The hash is calculated using the contents of the subject block, (with the <nonce> prepended) including the block start tag and block end tag, with the exception of all carriage returns, line feeds, and trailing spaces on a line. Leading and embedded spaces in a line are included in the hash. SGML entities (i.e., character names enclosed between an ampersand and a semicolon) are left untranslated when hashing.

4. The resulting hash value is inserted into the <hash> entry (as base64-encoded ASCII) in the signature block.

5. Steps 2 through 4 are repeated for each block to be signed.

6. A second hash calculation is performed on the contents of the <sigdata> sub-block, which contains the previously calculated hashes, their block references, and the <nonce>. This should include all characters between the <sigdata> tag and the </sigdata> tag, not including the tags themselves, again omitting all carriage returns, line feeds, and trailing spaces. If the signature algorithm is DSA or ECDSA, the hash may be inserted without additional padding or encoding. If the signature algorithm is RSA, the hash should be padded and encoded using the specifications in the PKCS#1[13] standard. This second hash is then signed using the private key in the electronic checkbook. The result is the signature which is inserted (as base64-encoded ASCII) into the signature block as the value for the <sig> tag.

Signature Verification

The verification of the Signature is performed as follows...

1. The following steps are repeated for each block referenced by a <blockref> tag in the signature. If the referenced block is not present, and `<blockref req="false">` was specified, the block is assumed to have been detached. The following steps are not performed for this block, and this block's absence is not considered to invalidate the document.

   (a) The <nonce> value from the signature block is logically prepended to the referenced blocks contents before hashing. This includes the tag string "<nonce>" — e.g., if the nonce value is 12345, the characters `<nonce>12345` are logically prepended to the referenced blocks contents before hashing.

   (b) A hash is calculated using the contents of the referenced block, (with the <nonce> prepended) including the block start tag and block end tag, with all characters in between, with the exception of all carriage returns, line feeds, and trailing spaces on a line. Leading and embedded spaces in a line are included in the hash. SGML entities (i.e., character names enclosed between an ampersand and a semicolon) are left untranslated when hashing. The hash algorithm to be used is specified in the `hash=` attribute in the <hash> tag for the referenced block.

   (c) The resulting hash value is compared to the <hash> entry in the signature block.

   (d) If the hashes do not match exactly, the signature fails verification.

2. The contents of the <sig> field are processed using the public key found by following the <sigref> tag. This tag will either point to an <account> block, or a <cert> block. If the <sigref> tag points (by name) to a <cert> block, the public key will be found in the <certdata> field in that block. (Parsing of the <certdata> field may be required to extract the public key — e.g., an X.509 certificate parser may

be required). If the <sigref> field points to an account block, the account block will, in turn, point to a <cert> block via the <certissuer> and <certserial> fields. The <cert> block whose <certissuer> and <certserial> fields match those in the <account> block contains the public key. The signature algorithm to be used is specified in the <algorithm> field.

3. A second hash calculation is performed on the contents of the <sigdata> sub-block, which contains the previously calculated hashes, their block references, and the <nonce>. This should include all characters between the <sigdata> tag and the </sigdata> tag, not including the tags themselves, again omitting all carriage returns, line feeds, and trailing spaces. The hash algorithm to be used is specified in the <algorithm> field.

4. The processed <sig> field is compared to the hash calculated in the previous step. If this comparison fails, the signature fails verification. If the comparison succeeds, the signature has verified successfully.

6.3.3 Check Block Definition

This block contains the key data for an FSML Electronic Check.

Multiple signers/certificates may be required, as determined by the restrictions field in the signer's account block.

```
<check>
<blkname>namestring
<crit>true
<vers>1.5
<checkdata>
<checknum>numstring
<dateissued>valuestring
<datevalid>valuestring
<country>namestring
<amount>amountstring
<currency>valuestring
<payto>valuestring
<paytobank>valuestring
<paytoacct>valuestring
<paytocustno>valuestring
<paytoid>valuestring
<paytoidns>valuestring
</checkdata>
<checkbook>numstring
<restrictions>valuestring
<payeracct>valuestring
<memo>valuestring
<info>valuestring
<conditions>valuestring
<legalnotice>valuestring
<vara>valuestring
<varb>valuestring
<varc>valuestring
<vard>valuestring
<vare>valuestring
<varf>valuestring
<varg>valuestring
<varh>valuestring
<vari>valuestring
</check>
```

Figure 6.14: Check block element definition

Check Block Field Definitions

vers (**required**) The <check> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value “1.5”.

checkdata (**required**) This is an enclosing sub-block. It is used to contain all of the check fields that will be interpreted and/or logged by the Electronic Checkbook hardware token. To simplify parsing by this token, the <checkdata> sub-block contents must be in the order specified.

checknum (**required**) This is the unique check number created by the Electronic Checkbook hardware token.

dateissued (**required**) This is the effective date of the check, supplied by the check issuer. It is not necessarily the date the check was written. The date must be specified in the ISO standard[12] format CCYYMMDD, where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month and “DD” is the day. Document recipients are free to process or ignore this date as they choose.

datevalid (**required**) This is the effective date of validity for the check, supplied by the check issuer. It is not necessarily the date the check was written. Currently, it should always be the same date as the <dateissued>. Other uses and values for this field will be described in a later edition of this specification. The date must be specified in the ISO standard[12] format CCYYMMDD, where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month and “DD” is the day.

country (optional) This is the 2 letter ISO country code[17] of the location where the check is to be considered written.

amount (**required**) A decimal number containing the amount of the check.

currency (**required**) A 3 letter ISO currency code[18].

payto (**required**) This is a string which is the name or other check-issuer specified identification of the payee. This field is used for informational purposes only, — i.e., creation of statement information. It is not verified against other data.

The following five fields form a subunit which identifies one of the possible payees for the check. If multiple payees are being specified, then the subunit may be repeated, with the fields in the same order for each payee (excluding optional fields). See the section on payto verification for an explanation of the use of these fields.

paytobank (optional) This is a field which if specified must be accompanied by either the <paytoacct> field, or the <paytocustno> field, and which contains the bank code of the payee.

paytoacct (optional) This is a field which if specified must be accompanied by the <paytobank> field, and which contains the account number of the payee.

paytocustno (optional) This is a field which if specified must be accompanied by the <paytobank> field. It contains the customer number of the payee at the payees bank. Some banks may use this in lieu of an account number.

paytoid (optional) This is a field which contains a unique identification of the payee.

paytoidns (optional) This is a field which, if specified, must be accompanied by the <paytoid> field, and which specifies the namespace in which the <paytoid¿> field is unique. If omitted, the <paytoid> field must be globally unique.

checkbook (**required**) This is an integer, supplied by the Electronic Checkbook hardware token, which is the bank-unique serial number of the checkbook.

restrictions (optional) This is a string containing restriction information about the specific check. The field may be repeated. It must be one of the following character strings.

- **duration p**nyn**m**n**d**

- **for deposit only**
- **all payees must endorse**

The **p**n**y**n**m**n**d** is an ISO standard[12] method of representing duration, where each "n" is a one or two digit number, and the **p** character is required. The numbers before **y**, **m**, and **d**, represent years, months and days, respectively. The duration (valid lifetime of a check) defaults to 60 days if not otherwise specified here.

payeracct (optional) This is a field containing a character string which is the account information of the payer at the payees business — i.e., the number that the payee uses to determine who is paying, or why it is being paid. This is not a bank account number. As an example, this is the payer's account number at the electric utility, on a check used to pay an electricity bill.

memo (optional) A character string field, used for any purpose the check issuer wishes. It is not processed by the bank.

info (optional) A character string field, used for any purpose the check issuing software wishes. It is not usually set or seen by the user, but may be used by the issuing software to communicate information to the receiving software; For example, the version of the software that created the document. It is not processed by the bank.

conditions (optional) A character string field, used to specify any conditions between the check issuer and endorser. Not processed by the bank.

legalnotice (**optional**) If present, this field must be inserted by any software that creates a new <check> block as containing one of the following two character strings.... **This instrument subject to check law** for normal echecks, or **This instrument subject to U.S Treasury check law** for Treasury echecks. Software that receives and processes echecks may check that the field is non-blank if present, but must not check that the strings contain the above values, as other values may be possible in future. This field is for legal notification purposes.

vara (optional)

varb (optional)

varc (optional)

vard (optional)

vare (optional)

varf (optional)

varg (optional)

varh (optional)

vari (optional) These fields may contain data which may be substituted into the MICR information used by the back-end systems at the bank or financial institution. The <micrmaskc> field in the <account> block determines which fields can be substituted, where they occur in the MICR line, and what format the data will take after substitution. The creator of the check determines the contents of the fields. Examples might be a location, or a special account number, or any other information that the check creator wishes to place in the MICR line and which the bank allows. One example is that certain U.S. Treasury checks place the check duration in the MICR line.

6.3.4 Deposit Block Definition

This block contains a electronic deposit slip, which is bound (via a signature block) to one or more endorsement blocks before being sent to a bank or other financial institution for deposit. The associated endorsement blocks must also have check blocks bound to them.

```
<deposit>
<blkname>namestring
<crit>true
<vers>1.5
<amount>amountstring
<currency>valuestring
<date>valuestring
<country>valuestring
<items>number
<bankacct>valuestring
<vara>valuestring
<varb>valuestring
<varc>valuestring
<vard>valuestring
<vare>valuestring
<varf>valuestring
<varg>valuestring
<varh>valuestring
<vari>valuestring
</deposit>
```

Figure 6.15: Deposit block element definition

Deposit Block Field Definitions

vers (**required**) The <deposit> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value **1.5**.

date (**required**) This is the effective date of the deposit slip, supplied by the depositor. It is not necessarily the date the deposit slip was created. The date must be specified in the ISO standard[12] format CCYYMMDD, where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month and “DD” is the day.

amount (**required**) A decimal number containing the total amount of the deposit.

currency (**required**) A 3 letter ISO currency code[18].

items (**required**) An integer specifying the total number of checks or other items being deposited.

country (optional) A 2 letter ISO Country code[17].

bankacct (**required**) This is a string containing the account number of this account in the issuing bank. This indicates the account that the funds are being deposited into.

vara (optional)

varb (optional)

varc (optional)

vard (optional)

vare (optional)

varf (optional)

varg (optional)

varh (optional)

vari (optional) These fields may contain data which may be substituted into the MICR information used by the back-end systems at the bank or financial institution. The <micrmaskd> field in the <account> block determines which fields can be substituted, where they occur in the MICR line, and what format the data will take after substitution. The creator of the deposit determines the contents of the fields. Examples might be a location, or a special account number, or any other information that the deposit creator wishes to place in the MICR line and which the bank allows.

6.3.5 Endorsement Block Definition

This block contains a digital endorsement of a financial document, usually a check. It must be bound (via a signature block) to the check it endorses.

```
<endorsement>
<blkname>namestring
<crit>true
<vers>1.5
<endorsedata>
<date>valuestring
<country>valuestring
<payto>valuestring
<paytobank>valuestring
<paytoacct>valuestring
<paytocustno>valuestring
<paytoid>valuestring
<paytoidns>valuestring
</endorsedata>
<checkbook>numstring
<restrictions>valuestring
<memo>valuestring
</endorsement>
```

Figure 6.16: Endorsement block element definition

Endorsement Block Field Definitions

vers (**required**) The <endorsement> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value “1.5”.

endorsedata (**required**) This is an enclosing sub-block. It is used to contain all of the endorsement fields that will be interpreted and/or logged by the Electronic Checkbook hardware token. To simplify parsing by this token, the <endorsedata> sub-block contents must be in the order specified. Since all of the fields enclosed in the <endorsedata> sub-block are optional, the sub-block may be empty. It is required to have the <endorsedata> and </endorsedata> tags in any case.

date (optional) This is the effective date of the endorsement, supplied by the endorser. It is not necessarily the date the endorsement was created. The date must be specified in the ISO standard[12] format CCYYMMDD, where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month and “DD” is the day.

country (optional) This is the 2 letter ISO country code of the location where the endorsement is to be considered written[17].

payto (optional) This is a string which is the name or other endorser identification of the ultimate payee or next holder in due course. This field is used for informational purposes only — i.e., creation of statement information. It is not verified against other data.

The following five fields form a subunit which identifies one of the possible payees for the check. If multiple payees are being specified, then the subunit may be repeated, with the 4 fields in the same order for each payee (excluding optional fields). See the section on payto verification for an explanation of the use of these fields.

paytobank (optional) This field if specified must be accompanied by either the <paytoacct> field, or the <paytocustno> field, and which contains the bank code of the ultimate payee.

paytoacct (optional) This field if specified must be accompanied by the <paytobank> field, and which contains the account number of the ultimate payee.

paytocustno (optional) This field if specified must be accompanied by the <paytobank> field. It contains the customer number of the ultimate payee at the their bank. Some banks may use this in lieu of an account number.

paytoid (optional) This is a field which contains a unique identification of the payee.

paytoidns (optional) This is a field which, if specified, must be accompanied by the <paytoid> field, and which specifies the namespace in which the <paytoid¿> field is unique. If omitted, the <paytoid> field must be globally unique.

checkbook (**required**) This is an integer, supplied by the Electronic Checkbook hardware token, which is the bank-unique serial number of the checkbook.

restrictions (optional) This is a string containing restriction information about the specific check being endorsed. The field may be repeated. It must be one of the following character strings.

- **for deposit only**

memo (optional) A character string field, used for any purpose the endorsement issuer wishes.

6.3.6 Certification Block Definition

This block contains a digital certification of a check — i.e., a statement by the bank that the funds are available and are held in the account until the check clears. It is created by a bank officer or teller and must be bound by a signature block to the check that it certifies.

```
<certification>
<blkname>namestring
<crit>true
<vers>1.0
<date>valuestring
<country>valuestring
<checkbook>numstring
<cserial>numstring
</certification>
```

Figure 6.17: Certification block element definition

Certification Block Field Definitions

date (**required**) This is the effective date of the certification, supplied by the certifier — i.e., the bank. The date must be specified in the ISO standard[12] format CCYYMMDD, where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month and "DD" is the day.

country (optional) This is the 2 letter ISO country code of the location where the certification is to be considered written[17].

checkbook (**required**) This is an integer, supplied by the Electronic Checkbook hardware token, which is the bank-unique serial number of the hardware token that is signing the certified check.

cserial (optional) This is a bank-unique serial number of the certified check, used for audit-trail and identification purposes in the issuing bank.

6.3.7 Account Block Definition

This block contains information about the account of the check issuer, or endorser. It is always used in combination with a certificate block.

```
<account>
<blkname>namestring
<crit>true
<vers>1.5
<bankcode>valuestring
<bankacct>valuestring
<bankser>numstring
<custno>numstring
<expdate>valuestring
<accttitle>valuestring
<accttype>valuestring
<bankname>valuestring
<bankaddr>valuestring
<bankphone>valuestring
<bankfax>valuestring
<bankemail>valuestring
<acctrest>valuestring
<sigrest>valuestring
<certissuer>valuestring
<certserial>number
<micrmaskc>valuestring
<micrmaskd>valuestring
</account>
```

Figure 6.18: Account block element definition

Account Block Field Definitions

blkname (**required**) The <blkname> field in an <account> block is slightly different than the "generic" <blkname>. Since the <account> block is signed by the bank issuing the electronic token, and is stored in the token, it is not changeable at runtime by FSML generating software. Thus the <blkname> chosen must be guaranteed to be unique for all subsequent documents. It is recommended (but not required) that a block naming convention be used to allow this. The recommended convention is that the name be suffixed with information that is unique to the account block, so that the same name would never be used by other account blocks in the same FSML document. As an example, an account block issued by a bank whose Bank Routing Code is 123456789, for a customer whose account number is 987654321 might have a blockname of `acct-123456789-987654321`.

vers (**required**) The <account> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value "1.5".

bankcode (**required**) This is a string containing the unique bank routing code of the issuing bank.

bankacct (**required**) This is a string containing the account number of this account in the issuing bank.

bankser (**required**) This is a number containing the account block serial number of this account in the issuing bank. This must be unique for all account blocks within an issuing bank code.

custno (optional) This is a bank-issued unique number that identifies the customer. This number may be used by the payer to uniquely specify the payee, without specifying the individual account, and thus may be assigned to several accounts for the same customer.

expdate (**required**) This is the expiration date of this account block. The date must be specified in the ISO standard[12] format CCYYMMDD, where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month and "DD" is the day.

accttitle (optional) This is a string containing the account title.

accttype (optional) This is a string containing the account type.

bankname (optional) This is a string containing the bank's name.

bankaddr (optional) This is a string containing the bank's address.

bankphone (optional) This is a string containing the bank's phone number.

bankfax (optional) This is a string containing the bank's fax number.

bankemail (optional) This is a string containing the bank's email address.

acctrest (optional) This is a string containing any restrictions on the account. It must be one of the following character strings. The field may be repeated.

- **minimum amount** nnnnnnn.nn ccc
- **maximum amount** nnnnnnn.nn ccc
- n **signatures required**
- n **signatures required above amount** nnnnn.nn ccc

- **special processing**
- **currency** ccc
- **duration p**n**y**n**m**n**d**

The “n” in the above restrictions represents a *number*. The “nnnnnnnn.nn” in the above restrictions represents an *amountstring*. The “ccc” is a 3 letter ISO currency code[18]. This indicates the currency being specified in the **amount**, or the currency that checks are being restricted to by the bank (in the **currency** ccc restriction). The **p**n**y**n**m**n**d** is an ISO standard[12] method of representing duration, where each “n” is a one or two digit number, and the **p** character is required. The Numbers before **y**, **m**, and **d**, represent years, months and days, respectively. The duration (valid lifetime of a check) defaults to 60 days if not otherwise specified here or in the <restrictions> field in the <check> block. If specified in both places, the shortest duration takes precedence.

certissuer (**required**) This field contains the unique distinguished name of the issuer of the certificate[7] which can be used to verify signatures on FSML documents containing this account block. See the description of the <certissuer> field in the <cert> block for the syntax used to specify this field.

certserial (**required**) This field contains the unique certificate serial number assigned by the issuer of the certificate which can be used to verify signatures on FSML documents containing this account block.

sigrest (optional) This field specifies what types of signatures are allowed to be created using this account block as credentials. If the field is not present, then any type of signature may be made, but signature verifiers may use other business rules in judging the validity of such signatures. One or more of the following codes may be specified, separated by colons ":"...

chk	Allows a signature with a <sigtype> of **check**
end	Allows a signature with a <sigtype> of **endorsement**
dep	Allows a signature with a <sigtype> of **deposit**
cos	Allows a signature with a <sigtype> of **co-sign**
cts	Allows a signature with a <sigtype> of **counter-sign**
coe	Allows a signature with a <sigtype> of **co-endorse**
cte	Allows a signature with a <sigtype> of **counter-endorse**
log	Allows a signature with a <sigtype> of **log-signature**
act	Allows a signature with a <sigtype> of **bankacct**
crt	Allows a signature with a <sigtype> of **certification**
edo	Allows a signature with a <sigtype> of **endorse-over**
bnk	Allows a signature with a <sigtype> of **bank**
gen	Allows a signature with a <sigtype> of **generic**

micrmaskc (optional) This field is used when creating the MICR-line data for an echeck. The <micrmaskc> field must be present if the <sigrest> field is absent or if it is present and contains ”chk”.

micrmaskd (optional) This field is used when creating the MICR-line data for an edeposit. The <micrmaskd> field must be present if the <sigrest> field is absent or if it is present and contains ”dep”.

These two fields contain information created by the Issuing Bank that enables a program receiving the echeck (usually at the Bank of First Deposit) to build the equivalent of the MICR line on paper checks. These MICR-line fields are defined in ANSI X9.13-1990[19]. This transmission information is also defined in ANSI X9.37-1994[20]. These fields contain a set of characters that encode the rules used to combine variable information from the other areas of the FSML (echeck or edeposit) with fixed information in these fields to create the bank-specific MICR line information.

Each field contains exactly 7 subfields separated by colons. Each subfield represents the desired format for the respective MICR field. The fields are specified in the order...

```
field7:field6:field5:field4:field3:field2:field1
```

which corresponds to the order in which the fields appear on a paper check. All 7 fields must be specified — i.e.,there must be exactly 6 colon separators, however fields may be left empty, — i.e., two colons may be adjacent. Any characters from the set `"0123456790/-+abcdefghijklmnopqrstuvwxyz"` and space, may be in the subfield, and represent themselves. MICR separator characters may be represented via a backslash escape mechanism...

- \a represents the MICR Amount Symbol
- \u represents the MICR OnUs/Account/Aux On Us/Serial symbol
- \t represents the ABA symbol
- \\ represents a backslash
- \% represents a percent symbol

If a `%` character is present in the subfield, it begins a substitution specification. Substitution specifications have the syntax `%[[-|0][`*field-length*`]`*field-specifier* where:

- An optional leading - indicates that the result is to be left justified or ...
- An optional leading 0 indicates that the result is to be zero padded (on the left).

  If neither a leading ”-” or ”0” is specified, then the data is inserted right justified into the micr line field, padded on the left (if padding is necessary) with spaces.
- An optional field length of one or two digits. If the field length is omitted, the actual size of the data determines the size to be inserted, and no padding or justification is performed.
- A field-specifier, which can be one of the following...

  m This indicates that the amount is to be inserted into the MICR field. When amounts are substituted into the MICR line, the decimal points (if any) are removed, and the amount is expressed in a currency dependent format. For USD, the amount is in integer cents.

  n This specifies that the check number is to be inserted into the MICR field.

t This specifies that the account number is to be inserted into the MICR field.

a This specifies that the contents of the <vara> field in the <check> or the <deposit> block be substituted into the MICR field.

b This specifies that the contents of the <varb> field in the <check> or the <deposit> block be substituted into the MICR field.

c This specifies that the contents of the <varc> field in the <check> or the <deposit> block be substituted into the MICR field.

d This specifies that the contents of the <vard> field in the <check> or the <deposit> block be substituted into the MICR field.

e This specifies that the contents of the <vare> field in the <check> or the <deposit> block be substituted into the MICR field.

f This specifies that the contents of the <varf> field in the <check> or the <deposit> block be substituted into the MICR field.

g This specifies that the contents of the <varg> field in the <check> or the <deposit> block be substituted into the MICR field.

h This specifies that the contents of the <varh> field in the <check> or the <deposit> block be substituted into the MICR field.

i This specifies that the contents of the <vari> field in the <check> or the <deposit> block be substituted into the MICR field.

The <micrmaskc> and <micrmaskd> fields may only be broken (have a newline inserted) immediately after a colon — i.e., between subfields. They must not be broken inside a subfield.

Some examples of a <micrmaskc> specification would be....

```
::343242424::8653903030:%n:%m
77432423234::31234990::98878%n:88898890:%05s:%m

If a document is counter-signed, then the micrmask fields from
the account block of the original signer are used when generating
a MICR line.  If a document is co-signed, then either signer’s
account block micrmask fields may be used.  It is assumed that
they will be the same, or equivalent.
```

6.3.8 echeck Certificate Block Definition

This block contains an encoded X.509 certificate[7]. It is identical to the Generic Certificate Block.

6.3.9 echeck Attachment Block Definition

This block contains any document that is to be attached to the FSML Electronic Document (e.g., a Remittance notice, Contract, etc.). It is identical to the Generic Attachment Block.

6.3.10 Invoice Block Definition

This block contains invoice information used by a merchant or other payee to request that the payer create a check using the information contained in the invoice. It is also used as remittance information by the payer to be attached to the check being used to pay the invoice.

```
<invoice>
<blkname>namestring
<crit>true
<vers>1.5
<custacct>valuestring
<amount>amountstring
<currency>valuestring
<payto>valuestring
<paytobank>valuestring
<paytoacct>valuestring
<paytocustno>valuestring
<paytoid>valuestring
<paytoidns>valuestring
<date>valuestring
<remittance>valuestring
<invdata>
...
</invdata>
</invoice>
```

Figure 6.19: Invoice block element definition

Invoice Block Field Definitions

vers (**required**) The <invoice> block is now at version 1.5, which is not the default and thus the <vers> field must be present and must contain the value “1.5”.

custacct (optional) This field contains the customers account number or code in the merchants (payees) accounting system. It should be returned in the <payeracct> field in the check used to pay the invoice.

amount (**required**) A decimal number containing the amount being invoiced, or the amount due.

currency (**required**) A 3 letter ISO currency code[18].

payto (**required**) This is a string which is the name or other check-issuer specified identification of the payee. This field is used for informational purposes only — i.e., creation of statement information. It is not verified against other data.

The following five fields form a subunit which identifies one of the possible payees for the check. If multiple payees are being specified, then the subunit may be repeated, with the fields in the same order for each payee (excluding optional fields). See the section on payto verification for an explanation of the use of these fields.

paytobank (optional) This is a field which if specified must be accompanied by either the <paytoacct> field, or the <paytocustno> field, and which contains the bank code of the payee.

paytoacct (optional) This is a field which if specified must be accompanied by the <paytobank> field, and which contains the account number of the payee.

paytocustno (optional) This is a field which if specified must be accompanied by the <paytobank> field. It contains the customer number of the payee at the payees bank. Some banks may use this in lieu of an account number.

paytoid (optional) This is a field which contains a unique identification of the payee.

paytoidns (optional) This is a field which, if specified, must be accompanied by the <paytoid> field, and which specifies the namespace in which the <paytoid¿> field is unique. If omitted, the <paytoid> field must be globally unique.

date (optional) The date that the payment is due. The date must be specified in the ISO standard[12] format CCYYMMDD, where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month and “DD” is the day.

remittance (optional) This field contains any remittance identification number or string that is being used to correlate this payment with other systems. It may contain any number or other identifier that would indicate to the recipient which invoice or which remittance item is associated with this FSML document.

invdata (optional) This field contains any other data that may be associated with the invoice — e.g., an purchase order or other purchase information.

6.3.11 echeck Message Block Definition

This block contains error messages and return information that indicate the reason that the attached FSML Document was not processed successfully or it may contain other information about the attached document. It is identical to the Generic Message Block.

6.3.12 Bankstamp Block Definition

This block contains processing information analogous to the bank stamps placed on the back of a paper check as it is processed by the banks and other institutions as it flows through the processing infrastructure.

```
<bankstamp>
<blkname>namestring
<crit>true
<vers>1.0
<date>valuestring
<timestamp>valuestring
<stampserial>numstring
<bankcode>valuestring
<serverid>valuestring
<stampdata>valuestring
</bankstamp>
```

Figure 6.20: Bankstamp block element definition

Bankstamp Block Field Definitions

date — (**required**) This is the effective date of the bank stamp. The date must be specified in the ISO standard[12] format CCYYMMDD, where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month and "DD" is the day.

timestamp — (**required**) This field specifies the time that the document was stamped. It must be in Universal time (i.e., GMT) specified as CCYYMMDDThhmmssZ, where the T and Z are literal characters, and where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month, "DD" is the day, "hh" is the hour, "mm" is the minute and ss is the second[12].

stampserial — (**required**) This field contains a bank-unique serial number, used for tracking purposes within the bank that creates the <bankstamp> block.

bankcode — (**required**) This is a string containing the unique bank routing code of the bank.

serverid — (optional) This is a string containing the unique bank echeck server identification, which may be used if a bank has more than one echeck server and needs to identify which one processed the document and created the bank stamp.

stampdata — (optional) This is a string containing any additional information the bank wishes to include in its bank stamps.

6.3.13 Bundle Block Definition

This block contains totals for bundles of presentment items. It is used within a bundle document, which must also be enclosed in a cgroup document. These documents are only used for presentment of echecks between banks. Payer and Payee software need never generate or parse <bundle> blocks.

```
<bundle>
<blkname>namestring
<crit>true
<vers>1.0
<items>number
<amount>amountstring
</bundle>
```

Figure 6.21: Bundle block element definition

Bundle Block Field Definitions

items (**required**) An integer specifying the total number of echecks contained in this bundle.

amount (**required**) A decimal number containing the total of all of the <amount> fields in all of the echecks contained in this bundle.

6.3.14 Cashletter Block Definition

This block contains totals for cgroup documents, which contain a number of bundle documents, which in turn contain a number of presentment items. These documents are only used for presentment of echecks between banks. Payer and Payee software need never generate or parse <cashletter> blocks.

```
<cashletter>
<blkname>namestring
<crit>true
<vers>1.0
<items>number
<amount>amountstring
<bundles>number
</cashletter>
```

Figure 6.22: Cashletter block element definition

Cashletter Block Field Definitions

items (**required**) An integer specifying the total number of echecks contained in all of the bundles in this document.

amount (**required**) A decimal number containing the total of all of the <amount> fields in all of the echecks contained in all of the bundles in this document.

bundles (**required**) A decimal number containing the number of bundle documents contained in this document.

Certificate Guidelines

Public key certificates (a.k.a., certificates) used in the FSTC echeck application space are industry standard X.509 certificates. X.509 certificates are specified using the ASN.1 abstract syntax specification language [14][21][22][23] with a default concrete syntax defined by the Distinguished Encoding Rules (DER)[15]. The primary purpose of an X.509 certificate is to bind the name of some entity to a public key. Certification Authorities (CA's) are used to issue certificates by digitally signing each certificate issued. A third party that uses a certificate to match a public key to a named entity can rely on the binding between key and name by checking the CA's digital signature.

Certificates can be used for authentication purposes (e.g., digital signatures) or for data encryption (i.e., key exchange). However, the only use for certificates in echeck documents defined by FSML is in determining the authenticity of digital signatures.

The Public Key Infrastructure used to issue and manage echeck-related certificates is currently defined as a pure hierarchy with a single root CA issuing certificates to subordinate CA's which in turn issue certificates to either end entities (e.g., users) or other CA's.

Within the FSTC echeck application context, there are two broad categories of certificates: CA certificates used to authenticate Certification Authorities, and end-entity certificates used to authenticate signers of echecks and related documents. Typically, there will be at least one Policy Certification Authority (PCA) that will issue certificates to CA's representing financial institutions. The PCA certificate will often be self-signed (i.e., a root certificate) since the PCA may not have any higher-level PCA from which it would receive a certificate. CA's operated by fianancial institutions will only issue end-entity certificates while PCA's will only issue certificates to subordinate CA's.

All X.509 certificates contain Distinguished Names (DNs) for both the issuing CA and the subject of the certificate (either another CA or end entitity). Certificates are uniquely identified by the Distinguished Name of the issuing CA and a certificate serial number supplied by the issuer. The echeck application uses a shortened version of the generalized DN employing only the following four fields:

countryName	a 2-letter code for the country where the certificates are issued
organizationName	name of the organization responsible for the PCA or CA
organizationUnitName	designates application context for certificate
dmdName	used to indicate the name space for the commonName
commonName	identifier of the end entity, not used in echeck CA certificates

Distinguished Names for the various categories of certificates currently defined for use in the echeck application are described as follows:

Policy CA (issuer of CA certificates to financial institutions)	
countryName	per ISO standard for 2-letter country codes
organizationName	name of authority, typically a regulatory body
organizationUnitName	”eCheck PCA”

Financial Institution CA	
countryName	per ISO standard for 2-letter country codes
organizationName	name of financial institution operating CA
organizationUnitName	”eCheck CA”

Authorized electronic checking account user or owner	
countryName	per ISO standard for 2-letter country codes
organizationName	name of financial institution that issued cert
organizationUnitName	”checking”
dmdName	defines name space for commonName (optional)
commonName	identifier for account user or owner

Officer or teller authorized to sign certified echecks	
countryName	per ISO standard for 2-letter country codes
organizationName	name of financial institution that issued cert
organizationUnitName	”operations”
dmdName	defines name space for commonName (optional)
commonName	identifier for officer or teller

It is important to note that the subject Distinguished Name for an end entity cert differs from the Distinguished Name of the issuing CA only by the addition of the dmdName and commonName elements. While this is not true for certificates in general, it is a convention adhered to within the echeck application space. In other words, end entity certificatesubject Distinguished Names inherit the countryName, organizationName and organizationUnitName from the issuing CA.

All CA Distinguished Names must be globally unique to comply with X.509 standards. Similarly, CA’s should issue end entity certificates with globally unique subject Distinguished Names. However, a CA can issue multiple certificates for the same subject, as would be the case for a renewed certificate or a certificate that replaces the public key for a subject.

For CA’s operated by financial institutions, use of the dmdName is optional. If the dmdName field is not used, then the commonName must be unique across all subjects for which certificates are issued by a given CA. Typically, the financial institution will enter the electronic checking account number into the commonName field when dmdName is not used. However, it is strongly recommended that this echeck account number not be the same as the associated true checking account (e.g., DDA) number. Since the commonName element will be visible to any party that can read an echeck, it is recommended from a fraud prevention perspective that disclosure of the commonName element not betray the true account number.

The intent of using the combination of dmdNamd and commonName is to allow financial institutions to issue end entity certificates with subject Distinguished Names that use a more universally recognized commonName. The dmdName field specifies what name space the commonName (uniquely) belongs to. For example, the dmdName field could specify that the name space is the Internet Domain Name system, thereby allowing the commonName to be a domain name or URL. Since domain names are guaranteed to be globally unique by the Internet naming authorities, the resulting subject Distinguished Name would certainly be unique.

A specific use for the combination of dmdName and commonName is to enable payers to characterize intended payees using the <paytoidns> and <paytoid> values within echecks. The dmdName field would map to <paytoidns>, while the commonName field would map to <paytoid>. Payees could prove that they are the intended recipient of the echeck payment by endorsing the echeck or signing a deposit slip using a certificate that contains identical matches for the payto values in the subject Distinguished Name dmdName and commonName fields.

The approach to forming names of subjects in echeck certificates (and FSML account blocks) aims to achieve flexibility and compliance with existing regulations and business practices. For example, individuals may need or desire anonymity in some of their payment transactions. Businesses, on the other hand, may need to be named using a widely recognized brand or company name. At the same time, disclosure of names, such as account numbers, should not introduce new opportunities for fraud or other abuse. This is why no one naming convention can be employed for all subjects in echeck certificates.

Within FSML, certificates are treated as binary data that is encapsulated within a Certificate Block. The <cert> block repeats the Distinguished Name of the certificate issuer and the serial number for the actual certificate, which together, uniquely identify the certificate.

For an electronic checking account to be established, an end-entity certificate must be issued along with an FSML Account Block that will in turn refer back to the certificate by referencing the issuer Distinguished Name and certificate serial number. The current convention is that certificates and account blocks will be issued together, and will stay in a one-to-one correspondence. The CA that issues the certificate will also sign the certificate block and account block using FSML signature rules, which will result in a third FSML block, the <signature> block. It is sometimes useful to refer collectively to the <cert>, <account> and <signature> blocks as the electronic checking “account credentials.”

A set of account credentials are typically associated with an electronic checkbook–i.e., a “crypto token”– that securely stores the corresponding private key and performs signature operations on echecks and other FSML documents. Where crypto tokens are used as electronic checkbooks, the process of issuing account credentials must be integrated with the production of the electronic checkbook, since the private/public key pair will be generated within the crypto token. Only the public key will be disclosed by the crypto token in order to include it in a request to the CA to issue account credentials for the electronic checkbook.

The X.509 standards also define mechanisms for revoking certificates. A certificate might have to be revoked if the corresponding private key is lost or compromised, if the user (subject) of the certificate has not complied with stated policies, or if the user wishes to terminate use of the certificate (e.g., they close their electronic checking account). Within the echeck application context, different approaches to revocation are taken for CA certificates vs. end entity certificates.

If a CA certificate must be revoked, then a standard X.509 (v2) Certificate Revocation List (CRL) will be published by the PCA that issued the CA cert. Since the total number of financial institutions that any one PCA would issue certificates for is on the order of 10,000; and since the number of CA certificate revocations is likely to be a small percentage, CRL’s should serve as an effective means for informing the financial industry of revoked CA certificates.

For end entity echeck certificates, though, the problem of certificate revocation is quite different. In reality, the certificate represents rights to use a specific checking account (e.g., DDA). More precisely, an echeck account certificate reflects authorization to use a specific electronic checking account, where multiple electronic checking accounts may map into a single traditional checking account, or DDA.

From the perspective of the financial institution that issues certificates for electronic checking accounts, the critical revocation issue is preventing further unauthorized use of the echeck account. This requires that steps to deactivate the account be taken on the various echeck server systems and legacy check processing systems that actually process transactions against the implicated account. In other words, the necessary condition is that the electronic checking account be revoked, and the consequential revocation of the account holder’s cert is essentially a housekeeping task.

The ability for financial institutions to directly control the status of the accounts they manage in nearly a real-time manner–even for paper checks–provides effective mechanisms for revoking electronic checking accounts. If an echeck account is closed, then no further deposits or echeck payments will be accepted for the account, although most financial institutions will treat outstanding transactions already in the pipeline on an exception basis. For example, should an account holder lose their private key (perhaps because the smartcard containing their electronic checkbook is lost or stolen) prudent behavior would be to immediately notify their financial institution so that the echeck account will be closed. Once, closed, the account holder is protected since no further payments will be processed against the account.

The implication of this approach is that there is no absolute requirement based on current practices to actively revoke echeck account certificates. Even if echeck certificates are revoked, no mechanisms currently exist to notify everyone who might receive an echeck from the revoked account. This situation is directly analogous to use of paper checks, where a traditional checking account can be closed, but the financial institution is not in a position to receive back all blank checks previously delivered to the account holder.

While CRL's could be published by financial institution echeck CA's, the scale of the potential echeck user base makes this infeasible in the short term. However, existing services used in the paper check context–such as check verification or check guarantee services–could easily be enhanced to provide similar risk mitigation for recipients of echecks. Furthermore, the Internet makes if feasible for recipients of echecks to contact the financial institution where the account resides to determine if a given echeck is drawn on a valid account. Financial institutions are also able to indicate funds availability for echecks drawn on their accounts.

Life cycle management of certificates, account credentials and electronic checkbooks (a.k.a., crypto tokens) is a complex topic that is beyond the scope of this document on FSML. Suffice it to say that certificates are always issued with defined validity periods (expressed as start and expiration dates), and will hence have to be renewed or replaced on a regular basis. Furthermore, life cycle management of end entity certificates will be intertwined with the management of electronic checking accounts. Because echeck account credentials contain a rich set of features and options, it is likely that they will be updated on a regular basis to allow for changes in parameters. Financial institutions will have considerable flexibility in deciding how to manage account credentials for their customers, providing that they adhere to policies established by the PCA, regulators and supervisors.

ASN.1 Definition of X.509 Certificates

This section provides a listing of the ASN.1[14][21][22][23] source code used to define certificates and CRLs for use within the echeck application context. This source code may be compiled to produce working certificate parsers and concrete encoders using DER or an equivalent concrete encoding specification (e.g., PER). In-line comments are used to further clarify the intended scope and use of ASN.1 code modules and associated data fields.

Both version 1 and 3 certificates are defined by the provided ASN.1 source code. Version 1 certificates would not use any version 3 extensions, nor would they include the optional issuerUniqueID and subjectUniqueID. Only version 2 CRLs are defined.

The ASN.1 source follows...

```
-- Modules
--
--  507 ANSI-X9-62
--   16 FSMLCertificates
--  740 FSMLCertificateManagement
--  796 FSMLCertificateRevocationList
--  183 FSMLExtensions
--  779 FSML-PKCSPlus
--  381 SupportingDefinitions

-- Last updated June 15, 1999
-- Prior update March 9, 1999
-- Prior update February 21, 1999


FSMLCertificates DEFINITIONS EXPLICIT TAGS ::= BEGIN

--
-- This FSMLCertificates module provides a collection of ASN.1
-- notation needed to support the management of X.509 public
-- key certificates in an electronic checking environment.
--

-- EXPORTS All;

IMPORTS
```

```
   ecdsa-with-SHA1, id-ecPublicKey, Parameters
      FROM ANSI-X9-62

   DSAParameters, DSAPublicKey, id-dsa, id-dsa-with-sha1
      FROM FSMLCertificateManagement

   authorityKeyIdentifier, basicConstraints, certificatePolicies,
   cRLNumber, issuerAltName, keyUsage, nameConstraints,
   privateKeyUsagePeriod, subjectAltName
      FROM FSMLExtensions

   id-sha1-with-rsa-signature, id-rsaEncryption
      FROM FSML-PKCSPlus

   Name, UniqueIdentifier
      FROM SupportingDefinitions;

-- basic certificate definition

Certificate ::= SIGNED { EncodedCertificateInfo }

EncodedCertificateInfo ::= TYPE-IDENTIFIER.&Type( CertificateInfo )

CertificateInfo ::= SEQUENCE {
   version              [0] Version DEFAULT v1,
   serialNumber         CertificateSerialNumber,
   signature            AlgorithmIdentifier{{SignatureAlgorithms}},
   issuer               Name,
   validity             Validity,
   subject              Name,
   subjectPublicKeyInfo  SubjectPublicKeyInfo,
   issuerUniqueID       [1] IMPLICIT UniqueIdentifier  OPTIONAL,
   subjectUniqueID      [2] IMPLICIT UniqueIdentifier  OPTIONAL,
   extensions           [3] Extensions { CertExtensions }  OPTIONAL
}

Version ::= INTEGER { v1(0), v3(2) } (v1 | v3 )  -- No v2 support

CertificateSerialNumber ::= INTEGER

AlgorithmIdentifier { ALGORITHM-ID:IOSet } ::= SEQUENCE {
   algorithm   ALGORITHM-ID.&id({IOSet}),
   parameters  ALGORITHM-ID.&Type({IOSet}{@algorithm}) OPTIONAL
}

SignatureAlgorithms ALGORITHM-ID ::= {
   sha1WithRSAsignature  |
   ecdsaWithSHA1         |
   dsa-with-sha1,
   ...
}

Validity ::= SEQUENCE {
   notBefore  Time,
   notAfter   Time
}

SubjectPublicKeyInfo ::= SEQUENCE {
   algorithm         AlgorithmIdentifier {{SupportedAlgorithms}},
   subjectPublicKey  BIT STRING
}

SupportedAlgorithms ALGORITHM-ID ::= {
   rsaEncryption    |
   ecPublicKeyType  |
   dsa,
   ...
}

Time ::= CHOICE {
   utcTime          UTCTime,
   generalizedTime  GeneralizedTime
}

Extensions { IOSet } ::= SEQUENCE OF Extension { IOSet }

Extension { IOSet } ::= SEQUENCE {
   extnID     EXTENSION.&id({IOSet}),
   critical   EXTENSION.&critical({IOSet}{@extnID})  DEFAULT FALSE,
   extnValue  OCTET STRING
      -- An "octet hole", the value of extnValue contains the DER
      -- encoding of a value of type &ExtnType, an open type, for
      -- the extension object identified by extnId.
}

EXTENSION ::= CLASS {
   &id         OBJECT IDENTIFIER UNIQUE,
   &critical   BOOLEAN DEFAULT FALSE,
   &ExtnType
}
 WITH SYNTAX
    { SYNTAX &ExtnType [CRITICAL &critical] IDENTIFIED BY &id }

CertExtensions EXTENSION ::= {
   authorityKeyIdentifier      |
   basicConstraints            |
   certificatePolicies         |
   issuerAltName               |
   keyUsage                    |
   nameConstraints             |
   privateKeyUsagePeriod       |
   subjectAltName,
   ...  -- Expect others in future revisions of this standard --
}

-- information object classes --

ALGORITHM-ID ::= CLASS {
   &id    OBJECT IDENTIFIER  UNIQUE,
   &Type  OPTIONAL
}
  WITH SYNTAX { OID &id [PARMS &Type] }

-- parameterized types --

SIGNED { ToBeSigned } ::= SEQUENCE {
   toBeSigned  ToBeSigned,
   algorithm   AlgorithmIdentifier{{SignatureAlgorithms}},
   signature   BIT STRING
}


-- supported cryptosystem key type information objects

dsa ALGORITHM-ID ::= {
   OID id-dsa PARMS DSAParameters
}

ecPublicKeyType ALGORITHM-ID ::= {
   OID id-ecPublicKey PARMS Parameters
}

rsaEncryption ALGORITHM-ID ::= {
   OID id-rsaEncryption PARMS NULL
}

-- signature algorithm information objects

dsa-with-sha1 ALGORITHM-ID ::= {
   OID id-dsa-with-sha1 PARMS NULL
}

ecdsaWithSHA1 ALGORITHM-ID ::= {
   OID ecdsa-with-SHA1 PARMS Parameters
}
```

```
sha1WithRSAsignature ALGORITHM-ID ::= {
   OID id-sha1-with-rsa-signature PARMS NULL
}


END  -- FSMLCertificates --



FSMLExtensions DEFINITIONS IMPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

IMPORTS

   CertificateSerialNumber, EXTENSION
      FROM FSMLCertificates

   DirectoryString {}, id-ce, Name, ub-name
      FROM SupportingDefinitions;


-- key and policy information extensions

authorityKeyIdentifier EXTENSION ::= {             -- Criticality:false
   SYNTAX         AuthorityKeyIdentifier
   IDENTIFIED BY  id-ce-authorityKeyIdentifier
}

AuthorityKeyIdentifier ::= SEQUENCE {
   keyIdentifier             [0] KeyIdentifier  OPTIONAL,
   authorityCertIssuer       [1] GeneralNames  OPTIONAL,
   authorityCertSerialNumber [2] CertificateSerialNumber  OPTIONAL
}
 ( WITH COMPONENTS  { ...,
      authorityCertIssuer        PRESENT,
      authorityCertSerialNumber  PRESENT } |
   WITH COMPONENTS  {...,
      authorityCertIssuer        ABSENT,
      authorityCertSerialNumber  ABSENT } )

KeyIdentifier ::= OCTET STRING

keyUsage EXTENSION ::= {                           -- Criticality:true
   SYNTAX         KeyUsage
   CRITICAL       TRUE
   IDENTIFIED BY  id-ce-keyUsage
}
```

```
KeyUsage ::= BIT STRING {
   digitalSignature  (0),
-- nonRepudiation    (1), not used --
   keyEncipherment   (2),
   dataEncipherment  (3),
   keyAgreement      (4),
   keyCertSign       (5),
   cRLSign           (6),
   encipherOnly      (7),
   decipherOnly      (8)
}

privateKeyUsagePeriod EXTENSION ::= {              -- Criticality:true
   SYNTAX         PrivateKeyUsagePeriod
   CRITICAL       TRUE
   IDENTIFIED BY  id-ce-privateKeyUsagePeriod
}

PrivateKeyUsagePeriod ::= SEQUENCE {
   notBefore  [0] GeneralizedTime  OPTIONAL,
   notAfter   [1] GeneralizedTime  OPTIONAL
}
 ( WITH COMPONENTS  {..., notBefore PRESENT} |
   WITH COMPONENTS  {..., notAfter  PRESENT} )

certificatePolicies EXTENSION ::= {               -- Criticality:false
   SYNTAX         CertificatePoliciesSyntax
   IDENTIFIED BY  id-ce-certificatePolicies
}

CertificatePoliciesSyntax ::=
   SEQUENCE SIZE(1..MAX) OF PolicyInformation

PolicyInformation ::= SEQUENCE {
   policyIdentifier  CertPolicyId,
   policyQualifiers  PolicyQualifiers  OPTIONAL
}

PolicyQualifiers ::=  SEQUENCE SIZE(1..MAX) OF PolicyQualifierInfo

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
   policyQualifierId
      CERT-POLICY-QUALIFIER.&id({SupportedPolicyQualifiers}),
   qualifier
      CERT-POLICY-QUALIFIER.&Qualifier({SupportedPolicyQualifiers}
            {@policyQualifierId})  OPTIONAL
}
```

```
SupportedPolicyQualifiers CERT-POLICY-QUALIFIER ::= {
   ...
}

CERT-POLICY-QUALIFIER ::= CLASS {
   &id         OBJECT IDENTIFIER  UNIQUE,
   &Qualifier  OPTIONAL
}
 WITH SYNTAX
    { POLICY-QUALIFIER-ID &id [QUALIFIER-TYPE &Qualifier] }


-- certificate subject and issuer attributes extensions

subjectAltName EXTENSION ::= {          -- Criticality:false (or true)
   SYNTAX         GeneralNames
-- CRITICAL       TRUE --
   IDENTIFIED BY  id-ce-subjectAltName
}

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
   otherName                  [0] INSTANCE OF OTHER-NAME,
   rfc822Name                 [1] IA5String,
   dNSName                    [2] IA5String,
   -- x400Address             [3] ORAddress,      (not used)
   directoryName              [4] EXPLICIT Name,
   ediPartyName               [5] EDIPartyName,
   uniformResourceIdentifier  [6] IA5String,
   iPAddress                  [7] OCTET STRING,
   registeredID               [8] OBJECT IDENTIFIER
}

OTHER-NAME ::= TYPE-IDENTIFIER

EDIPartyName ::= SEQUENCE {
   nameAssigner  [0] DirectoryString {ub-name} OPTIONAL,
   partyName     [1] DirectoryString {ub-name}
}

issuerAltName EXTENSION ::= {           -- Criticality:false (or true)
   SYNTAX         GeneralNames
-- CRITICAL       TRUE --
   IDENTIFIED BY  id-ce-issuerAltName
}

-- certification path constraints extensions
```

```
basicConstraints EXTENSION ::= {                   -- Criticality:true
   SYNTAX         BasicConstraintsSyntax
   CRITICAL       TRUE
   IDENTIFIED BY  id-ce-basicConstraints
}

BasicConstraintsSyntax ::= SEQUENCE {
   cA                 BOOLEAN DEFAULT FALSE,
   pathLenConstraint  INTEGER (0..MAX)  OPTIONAL
}

-- Basic CRL extensions --

cRLNumber EXTENSION ::= {                         -- Criticality:false
   SYNTAX         CRLNumber
   IDENTIFIED BY  id-ce-cRLNumber
}

CRLNumber ::= INTEGER (0..MAX)

nameConstraints EXTENSION ::= {                    -- Criticality:true
   SYNTAX         NameConstraintsSyntax
   CRITICAL       TRUE
   IDENTIFIED BY  id-ce-nameConstraints
}


NameConstraintsSyntax ::= SEQUENCE {
   permittedSubtrees  [0] GeneralSubtrees OPTIONAL,
   excludedSubtrees   [1] GeneralSubtrees OPTIONAL
}

GeneralSubtrees ::= SEQUENCE SIZE(1..MAX) OF GeneralSubtree

GeneralSubtree ::= SEQUENCE {
   base     GeneralName,
   minimum  [0] BaseDistance DEFAULT 0,
   maximum  [1] BaseDistance  OPTIONAL
}

BaseDistance ::= INTEGER (0..MAX)

-- object identifier assignments --

id-ce-subjectKeyIdentifier       OBJECT IDENTIFIER ::= { id-ce 14 }
id-ce-keyUsage                   OBJECT IDENTIFIER ::= { id-ce 15 }
id-ce-privateKeyUsagePeriod      OBJECT IDENTIFIER ::= { id-ce 16 }
id-ce-subjectAltName             OBJECT IDENTIFIER ::= { id-ce 17 }
```

```
id-ce-issuerAltName              OBJECT IDENTIFIER ::= { id-ce 18 }
id-ce-basicConstraints           OBJECT IDENTIFIER ::= { id-ce 19 }
id-ce-cRLNumber                  OBJECT IDENTIFIER ::= { id-ce 20 }
id-ce-nameConstraints            OBJECT IDENTIFIER ::= { id-ce 30 }
id-ce-certificatePolicies        OBJECT IDENTIFIER ::= { id-ce 32 }
id-ce-authorityKeyIdentifier     OBJECT IDENTIFIER ::= { id-ce 35 }

END  -- FSMLExtensions --



SupportingDefinitions DEFINITIONS EXPLICIT TAGS ::= BEGIN

--
-- This SupportingDefinitions module provides a collection of
-- ASN.1 notation needed to support the management of public
-- key certificates in a financial services environment.
--

-- EXPORTS All;

-- IMPORTS None;

--
-- Defined in ISO/IEC 9594-6|X.520 module SelectedAttributeTypes
--

DirectoryString { INTEGER:maxSize } ::= CHOICE {
   teletexString    TeletexString (SIZE(1..maxSize)),
   printableString  PrintableString (SIZE(1..maxSize)),
   universalString  UniversalString (SIZE(1..maxSize)),
   bmpString        BMPString (SIZE(1..maxSize)),
   utf8String       UTF8String (SIZE(1..maxSize))
}

UniqueIdentifier ::= BIT STRING

commonName ATTRIBUTE ::= {
   WITH SYNTAX  DirectoryString { ub-common-name }
   ID           id-at-commonName
}

countryName ATTRIBUTE ::= {
   WITH SYNTAX  CountryName
   ID           id-at-countryName
}

CountryName ::= PrintableString (SIZE(2))  -- IS 3166 codes only
```

```
organizationName ATTRIBUTE ::= {
   WITH SYNTAX  DirectoryString { ub-organization-name }
   ID           id-at-organizationName
}

organizationalUnitName ATTRIBUTE ::= {
   WITH SYNTAX  DirectoryString { ub-organizational-unit-name }
   ID           id-at-organizationalUnitName
}

dmdName ATTRIBUTE ::= {
   WITH SYNTAX  DirectoryString{ ub-common-name }
   ID           id-at-dmdName
}

RDNameAttributes ATTRIBUTE ::= {
   commonName              |
   countryName             |
   organizationName        |
   organizationalUnitName  |
   dmdName
   --
   -- Expect no other DN components
   --
}

--
-- Defined in ISO/IEC 9594-6|X.520 module UpperBounds
--

ub-name  INTEGER ::= 32768

ub-common-name               INTEGER ::=  64
ub-organization-name         INTEGER ::=  64
ub-organizational-unit-name  INTEGER ::=  64

--
-- Defined in ISO/IEC 9594-2|X.501 module InformationFramework
--

Name ::= CHOICE {  -- only one possibility for now --
   rdnSequence  RDNSequence
}

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET SIZE(1..MAX) OF
                                       AttributeTypeAndValue
```

```
AttributeTypeAndValue ::= SEQUENCE {
   type   ATTRIBUTE.&id({RDNameAttributes}),
   value  ATTRIBUTE.&Type({RDNameAttributes}{@type})
}

--
-- This simplified version of the ATTRIBUTE class defined in
-- ISO/IEC 9594-2|X.501 module InformationFramework produces
-- syntax conformant with that Directory standard, but only
-- includes those fields needed for public key certificates.
--

ATTRIBUTE ::= CLASS {
   &Type,
   &id     OBJECT IDENTIFIER UNIQUE
}
  WITH SYNTAX { WITH SYNTAX &Type ID &id }

--
-- Defined in ISO/IEC 9594-2|X.501 module UsefulDefinitions
--

ds OBJECT IDENTIFIER ::= {
   joint-iso-ccitt ds(5) }  -- Directory Services

id-ce OBJECT IDENTIFIER ::= { ds 29 }  -- Certificate Extension

id-at OBJECT IDENTIFIER ::= { ds 4 }  -- Attribute Type

id-at-commonName              OBJECT IDENTIFIER ::= { id-at  3 }
id-at-countryName             OBJECT IDENTIFIER ::= { id-at  6 }
id-at-organizationName        OBJECT IDENTIFIER ::= { id-at 10 }
id-at-organizationalUnitName  OBJECT IDENTIFIER ::= { id-at 11 }
id-at-dmdName                 OBJECT IDENTIFIER ::= { id-at 54 }

END  -- SupportingDefinitions --



ANSI-X9-62 { iso(1) member-body(2) us(840) 10045 module(4) 1 }
   DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All;

-- IMPORTS None;

ansi-X9-62 OBJECT IDENTIFIER ::= {
      iso(1) member-body(2) us(840) 10045
}
```

```
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {         -- Finite field
   fieldType   FIELD-ID.&id({IOSet}),
   parameters  FIELD-ID.&Type({IOSet}{@fieldType})
}

FieldTypes FIELD-ID ::= {
   { Prime-p            IDENTIFIED BY prime-field              } |
   { Characteristic-two IDENTIFIED BY characteristic-two-field },
   ...
}

FIELD-ID ::= TYPE-IDENTIFIER  -- ISO/IEC 8824-2:1995(E), Annex A

id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1) }

prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }

characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }

Prime-p ::= INTEGER    -- Finite field F(p), p is an odd prime

Characteristic-two ::= SEQUENCE {
   m           INTEGER,          -- Field size 2ˆm
   basis       CHARACTERISTIC-TWO.&id({BasisTypes}),
   parameters  CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}

BasisTypes CHARACTERISTIC-TWO::= {
   { NULL         IDENTIFIED BY  gnBasis  } |
   { Trinomial    IDENTIFIED BY  tpBasis  } |
   { Pentanomial  IDENTIFIED BY  ppBasis  },
   ...
}

-- Trinomial basis representation of F2ˆm
-- Integer k for reduction polynomial xm + xk + 1
--
Trinomial ::= INTEGER

Pentanomial ::= SEQUENCE {
--
-- Pentanomial basis representation of F2ˆm
-- reduction polynomial integers k1, k2, k3
-- f(x) = x**m + x**k3 + x**k2 + x**k1 + 1
--
   k1  INTEGER,
   k2  INTEGER,
   k3  INTEGER
}

CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER

id-characteristic-two-basis OBJECT IDENTIFIER ::= {
   characteristic-two-field basisType(3) }

-- The object identifiers gnBasis, tpBasis and ppBasis name
-- three kinds of basis for characteristic-two finite fields

gnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis  OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }

FieldElement ::= OCTET STRING    -- Finite field element

ECPoint  ::= OCTET STRING    -- Elliptic curve point

ECParameters  ::= SEQUENCE {   -- Elliptic curve parameters
   version   INTEGER { ecpVer1(1) } (ecpVer1),
   fieldID   FieldID {{FieldTypes}},
   curve     Curve,
   base      ECPoint,            -- Base point G
   order     INTEGER,            -- Order n of the base point
   cofactor  INTEGER  OPTIONAL,  -- The integer h = #E(Fq)/n
   ...
}

Curve  ::= SEQUENCE {
   a     FieldElement,       -- Elliptic curve coefficient a
   b     FieldElement,       -- Elliptic curve coefficient b
   seed  BIT STRING OPTIONAL
}

-- When a digital signature is identified by the object identifier
-- ecdsa-with-SHA1 (as in module FSMLCertificates), the digital
-- signature shall be encoded as a value of type ECDSA-Sig-Value.
-- Objects such as X.509 certificates and CRLs represent digital
-- signatures as a bit string. Where a certificate or CRL is
-- signed with ECDSA and SHA-1, the entire encoding of a value
-- of type ECDSA-Sig-Value shall be the value (in ASN.1 TLV format)
-- of the bit string signature.

ECDSA-Sig-Value ::= SEQUENCE {
   r  INTEGER,
   s  INTEGER
}

id-ecSigType OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
```

```
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType 1 }


-- An elliptic curve public key value of ECPoint, an OCTET
-- STRING value, is mapped to BIT STRING value subjectPublicKey
-- as follows: the most significant octet of the octet string
-- value becomes the most significant bits of the bit string
-- value and the least significant octet of the octet string
-- value becomes the least significant bits of the bit string.

SubjectPublicKeyInfo ::= SEQUENCE {
   algorithm         AlgorithmIdentifier {{ECPKAlgorithms}},
   subjectPublicKey  BIT STRING
}

AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
   algorithm   ALGORITHM.&id({IOSet}),
   parameters  ALGORITHM.&Type({IOSet}{@algorithm})
}

ECPKAlgorithms ALGORITHM ::= {
   ecPublicKeyType,
   ...
}

ecPublicKeyType ALGORITHM ::= {
   Parameters IDENTIFIED BY id-ecPublicKey
}

ALGORITHM ::= TYPE-IDENTIFIER

id-publicKeyType OBJECT IDENTIFIER  ::= { ansi-X9-62 keyType(2) }

id-ecPublicKey OBJECT IDENTIFIER ::= { id-publicKeyType 1 }

-- The public key Parameters are defined in X9.62 as a choice of
-- three alternatives. This allows detailed specification of all
-- possible required values using choice alternative ecParameters,
-- the use of an efficient namedCurve as an object identifier
-- substitute for a particular set of elliptic curve domain
-- parameters, or implicitlyCA in specific domains to indicate
-- that the parameters are explicitly defined elsewhere.

Parameters ::= CHOICE {
   ecParameters  ECParameters,
   namedCurve    CURVES.&id({CurveNames}),
   implicitlyCA  NULL
}
```

```
CurveNames CURVES ::= {
   { ID c2pnb163v1 }   |   -- J.4.1,   example 1 --
   { ID c2pnb163v2 }   |   -- J.4.1,   example 2 --
   { ID c2pnb163v3 }   |   -- J.4.1,   example 3 --
   { ID c2pnb176w1 }   |   -- J.4.2,   example 1 --
   { ID c2tnb191v1 }   |   -- J.4.3,   example 1 --
   { ID c2tnb191v2 }   |   -- J.4.3,   example 2 --
   { ID c2tnb191v3 }   |   -- J.4.3,   example 3 --
   { ID c2onb191v4 }   |   -- J.4.3,   example 4 --
   { ID c2onb191v5 }   |   -- J.4.3,   example 5 --
   { ID c2pnb208w1 }   |   -- J.4.4,   example 1 --
   { ID c2tnb239v1 }   |   -- J.4.5,   example 1 --
   { ID c2tnb239v2 }   |   -- J.4.5,   example 2 --
   { ID c2tnb239v3 }   |   -- J.4.5,   example 3 --
   { ID c2onb239v4 }   |   -- J.4.5,   example 4 --
   { ID c2onb239v5 }   |   -- J.4.5,   example 5 --
   { ID c2pnb272w1 }   |   -- J.4.6,   example 1 --
   { ID c2pnb304w1 }   |   -- J.4.7,   example 1 --
   { ID c2tnb359v1 }   |   -- J.4.8,   example 1 --
   { ID c2pnb368w1 }   |   -- J.4.9,   example 1 --
   { ID c2tnb431r1 }   |   -- J.4.10,  example 1 --
   { ID prime192v1 }   |   -- J.5.1,   example 1 --
   { ID prime192v2 }   |   -- J.5.1,   example 2 --
   { ID prime192v3 }   |   -- J.5.1,   example 3 --
   { ID prime239v1 }   |   -- J.5.2,   example 1 --
   { ID prime239v2 }   |   -- J.5.2,   example 2 --
   { ID prime239v3 }   |   -- J.5.2,   example 3 --
   { ID prime256v1 },      -- J.5.3,   example 1 --
   ... -- others --
}

CURVES ::= CLASS {
   &id  OBJECT IDENTIFIER UNIQUE
}
  WITH SYNTAX { ID &id }

ellipticCurve OBJECT IDENTIFIER ::= { ansi-X9-62 curves(3) }

c-TwoCurve OBJECT IDENTIFIER ::= {
   ellipticCurve characteristicTwo(0) }

primeCurve OBJECT IDENTIFIER ::= { ellipticCurve prime(1) }

c2pnb163v1 OBJECT IDENTIFIER ::= { c-TwoCurve  1 }
c2pnb163v2 OBJECT IDENTIFIER ::= { c-TwoCurve  2 }
c2pnb163v3 OBJECT IDENTIFIER ::= { c-TwoCurve  3 }
c2pnb176w1 OBJECT IDENTIFIER ::= { c-TwoCurve  4 }
c2tnb191v1 OBJECT IDENTIFIER ::= { c-TwoCurve  5 }
```

```
c2tnb191v2 OBJECT IDENTIFIER ::= { c-TwoCurve  6 }
c2tnb191v3 OBJECT IDENTIFIER ::= { c-TwoCurve  7 }
c2onb191v4 OBJECT IDENTIFIER ::= { c-TwoCurve  8 }
c2onb191v5 OBJECT IDENTIFIER ::= { c-TwoCurve  9 }
c2pnb208w1 OBJECT IDENTIFIER ::= { c-TwoCurve 10 }
c2tnb239v1 OBJECT IDENTIFIER ::= { c-TwoCurve 11 }
c2tnb239v2 OBJECT IDENTIFIER ::= { c-TwoCurve 12 }
c2tnb239v3 OBJECT IDENTIFIER ::= { c-TwoCurve 13 }
c2onb239v4 OBJECT IDENTIFIER ::= { c-TwoCurve 14 }
c2onb239v5 OBJECT IDENTIFIER ::= { c-TwoCurve 15 }
c2pnb272w1 OBJECT IDENTIFIER ::= { c-TwoCurve 16 }
c2pnb304w1 OBJECT IDENTIFIER ::= { c-TwoCurve 17 }
c2tnb359v1 OBJECT IDENTIFIER ::= { c-TwoCurve 18 }
c2pnb368w1 OBJECT IDENTIFIER ::= { c-TwoCurve 19 }
c2tnb431r1 OBJECT IDENTIFIER ::= { c-TwoCurve 20 }
prime192v1 OBJECT IDENTIFIER ::= { primeCurve  1 }
prime192v2 OBJECT IDENTIFIER ::= { primeCurve  2 }
prime192v3 OBJECT IDENTIFIER ::= { primeCurve  3 }
prime239v1 OBJECT IDENTIFIER ::= { primeCurve  4 }
prime239v2 OBJECT IDENTIFIER ::= { primeCurve  5 }
prime239v3 OBJECT IDENTIFIER ::= { primeCurve  6 }
prime256v1 OBJECT IDENTIFIER ::= { primeCurve  7 }

END  -- ANSI-X9-62 --



FSMLCertificateManagement DEFINITIONS IMPLICIT TAGS ::= BEGIN

--
-- This module is a subset of ISO 15782-1 ASN.1 module named
-- CertificateManagement { iso(1) member-body(2) us(840)
-- x9-57(10040) module(1) certificateManagement(1) }
--

-- EXPORTS All

-- IMPORTS None

DSAPublicKey ::= INTEGER -- public key y

DSAParameters ::= SEQUENCE {
   prime1  INTEGER,  -- modulus p
   prime2  INTEGER,  -- modulus q
   base    INTEGER   -- base g
}

-- object identifier assignments
```

```
x9-57Algorithm OBJECT IDENTIFIER ::= {
   iso(1) member-body(2) us(840) x9-57(10040) algorithm(4)
}

id-dsa OBJECT IDENTIFIER ::= {
   x9-57Algorithm dsa(1)
}

id-dsa-with-sha1 OBJECT IDENTIFIER ::= {
   x9-57Algorithm dsa-with-sha1(3)
}


END  -- FSMLCertificateManagement --



FSML-PKCSPlus DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- The following definitions are taken from the SET
-- ASN.1 module named SetPKCS7Plus.

pkcs-1 OBJECT IDENTIFIER ::= {
   iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 1 }

id-rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }

id-sha1-with-rsa-signature  OBJECT IDENTIFIER ::= { pkcs-1 5 }


END



FSMLCertificateRevocationList
      DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All;

IMPORTS

   AlgorithmIdentifier{}, CertificateSerialNumber, EXTENSION,
   Extensions {}, SignatureAlgorithms, SIGNED {}, Time
      FROM FSMLCertificates

   authorityKeyIdentifier, cRLNumber
      FROM FSMLExtensions

   Name
```

```
      FROM SupportingDefinitions;


CRL ::= SIGNED { EncodedCRLInfo }

EncodedCRLInfo ::=
      TYPE-IDENTIFIER.&Type( CertificateRevocationListInfo )

CertificateRevocationListInfo ::= SEQUENCE {
   version              INTEGER { crliVer2(1) } ( crliVer2 ),
   signature            AlgorithmIdentifier {{SignatureAlgorithms}},
   issuer               Name,
   thisUpdate           Time,
   nextUpdate           Time,
   revokedCertificates  CRLEntryList  OPTIONAL,
   crlExtensions        [0] CRLExtensions  OPTIONAL
}

CRLExtensions ::= Extensions { CRLSet }

CRLSet EXTENSION ::= {
   authorityKeyIdentifier |
   cRLNumber,
   ...
}

CRLEntryList ::= SEQUENCE OF CRLEntry

CRLEntry ::= SEQUENCE{
   userCertificate     CertificateSerialNumber,
   revocationDate      Time,
   crlEntryExtensions  CRLEntryExtensions  OPTIONAL
}

CRLEntryExtensions ::= Extensions { CRLEntrySet }

CRLEntrySet EXTENSION ::= { ... }  -- None defined for FSML use


END  -- FSMLCertificateRevocationList --
```

Elliptic Curve cryptographic recommendations

The key sizes recommended are 163-bit and 283-bit. 163-bit is recommended for user key sizes, but 283-bit may be desirable for certificate creation. The availability of OIDs for both key sizes enables a choice of security strengths now and in the future.

Two curve recommendations for 163-bit ECC, already assigned OIDs, are...

```
sect163k1 = 1 3 132 0 1
sect163r1 = 1 3 132 0 2
```

A new OID which will be assigned for 163-bit ECC is...

```
sect163r2 = 1 3 132 0 15
```

Two new OIDs which will be assigned for 283-bit ECC are...

```
sect283k1 = 1 3 132 0 16
sect283r1 = 1 3 132 0 17
```

The parameters corresponding to all the OIDs above can be found in [6]. sect163k1, sect163r2, sect283k1, and sect283r1 are among the curves recommended by NIST for federal government use in [24]. sect163r1 is currently available more widely than sect163r2 in COTS products.

Field Summary

Below is a summary of the attributes of each of the entities or fields allowed in an FSML Electronic Document

10.1 Field Attributes Table (part 1)

Field Attribute Summary - Part One							
Field Name	Containing Blocks	Min Size	Max Size	Size Code	Opt	Logged	Notes
blkname	all	1	64	P			
crit	all	4	5	F	Yes		
vers	all	1	8	P	Yes		
acctrest	<account>	1	256	P	Yes		
accttitle	<account>	1	76	P	Yes		
accttype	<account>	1	76	P	Yes		
adata	<attachment>	1	N/A		Yes		
algorithm	<signature>	7	9	F			
amount	multiple types	1	21	B		Yes	
astatus	<attachment>	9	9	F	Yes		
bankacct	<account> <deposit>	1	20	B			
bankaddr	<account>	1	76	P	Yes		
bankcode	<account> <bankstamp>	9	9	B	Yes		
bankemail	<account>	1	76	P	Yes		
bankfax	<account>	1	76	P	Yes		
bankname	<account>	1	76	P	Yes		
bankphone	<account>	1	76	P	Yes		
bankser	<account>	1	16	B			
bundles	<cashletter>	1	8	P			
blockref	<signature>	1	76	P			
certdata	<cert>	1	N/A				
certissuer	multiple types	1	256	P	Yes		
certserial	multiple types	1	16	P	Yes		
certtype	<cert>	6	6	F			
checkbook	multiple types	1	16	B			
checknum	<checkdata>	1	19	B		Yes	
conditions	<check>	1	76	P	Yes		
country	multiple types	2	2	F	Yes	Yes	
cserial	<certification>	1	16	P	Yes		
currency	multiple types	3	3	F	Yes	Yes	
custno	<account>	1	76	P	Yes		

10.2 Field Attributes Table (part 2)

Field Attribute Summary - Part Two							
Field	Containing	Min	Max	Size	Opt	Logged	Notes
Name	Blocks	Size	Size	Code			
custacct	<invoice>	1	76	P	Yes		
date	multiple types	8	8	F	Yes	Yes	
dateissued	<checkdata>	8	8	F		Yes	
datevalid	<checkdata>	8	8	F		Yes	
expdate	<account>	8	8	F			
function	<action>	1	16	F			
hash	<signature>	1	256	F			
info	<check>	1	76	P	Yes		
invdata	<invoice>	1	N/A		Yes		
items	multiple types	1	8	P			
legalnotice	<check>	1	76	P			
location	<signature>	1	76	P	Yes		
memo	<check>	1	76	P	Yes		
micrmaskc	<account>	1	128	B	Yes		
micrmaskd	<account>	1	128	B	Yes		
msgtext	<message>	1	76	P			
msgdata	<message>	1	N/A		Yes		
nonce	<signature>	8	16	P			
payeracct	<check>	1	76	P	Yes		
payto	multiple types	1	76	L		Yes	1
paytoacct	multiple types	1	20	B	Yes	Yes	
paytobank	multiple types	9	9	B	Yes	Yes	
paytocustno	multiple types	1	76	L	Yes	Yes	
paytoid	multiple types	1	76	P	Yes	Yes	
paytoidns	multiple types	1	76	P	Yes	Yes	
reason	<action>	1	16	F			
remittance	<invoice>	1	76	P	Yes		
restrictions	<check> <endorsement>	1	256	P			
retcode	<message>	1	8	F			
serverid	<bankstamp>	1	8	P			
sig	<signature>	42	256	F			
sigref	<signature>	1	76	P	Yes		
sigrest	<account>	1	76	P	Yes		
sigtype	<signature>	1	16	P		Yes	
stampdata	<bankstamp>	N/A	N/A				
stampserial	<bankstamp>	1	15	P			
timestamp	<signature> <bankstamp>	16	16	F	Yes		

10.3 Field Attributes Table (part 3)

Field Attribute Summary - Part Three							
Field Name	Containing Blocks	Min Size	Max Size	Size Code	Opt	Logged	Notes
useraddr	<signature>	1	76	P	Yes		
useremail	<signature>	1	76	P	Yes		
useridnum	<signature>	1	76	P	Yes		
username	<signature>	1	76	P	Yes		
userotherid	<signature>	1	76	P	Yes		
userotherid	<signature>	1	76	P	Yes		
vara	<check>	1	76	P	Yes		
varb	<check>	1	76	P	Yes		
varc	<check>	1	76	P	Yes		
vard	<check>	1	76	P	Yes		
vare	<check>	1	76	P	Yes		
varf	<check>	1	76	P	Yes		
varg	<check>	1	76	P	Yes		
varh	<check>	1	76	P	Yes		
vari	<check>	1	76	P	Yes		

Notes:

1 Only the first 16 bytes are logged.

The Size Code column in the above tables indicates the reason that the specified Max Sizes were chosen, as follows:

Max Size Reason codes	
Code	Rationale
B	Required by Banking standards
F	Format of data dictates size
L	Limited by Logging area size limitations
P	Practicality considerations limit size

Document Verification

The steps required to verify an FSML document, both syntactically and cryptographically are discussed in this chapter.

11.1 Verifying Document Contents

The following tables describe the verification requirements for Document Contents.

Verification Table - Document Verification Rules - Part One

Document	<Block>/Document	Required	Notes
checkdoc	<action>	x	Function must be `payment`. Reason must be `process`, `test`, `resend`, or `info`. If reason is not `process` or `resend`, don't process financially.
	<check>	x	
	<signature>	x	Signer's signature. See Signature Rules table.
	<signature>		Co-signer's (or counter-signer's) signature. See Signature Rules table.
	<account>	x	Signer's account. Refers to signer's certificate.
	<cert>		Signer's certificate. See Certificate Verification rules.
	<account>		co-signer's (or counter-signer's) account. Refers to co-signer's certificate.
	<cert>		co-signer's (or counter-signer's) certificate. See Certificate Verification rules.
	<signature>	x	Bank's signature on signer's account. See Signature Rules table.
	<signature>		Bank's signature on co-signer's (or counter-signer's account. See Signature Rules table.
	<cert>		Bank's certificate. See Certificate Verification Rules
	<attachment>		0 or more. Must be signed.
	<invoice>		0 or more. Must be signed.
depositdoc	<action>	x	Function must be `deposit`. Reason must be `process`, `resend` or `test`. If reason is `resend` process if not a duplicate, otherwise ignore. If reason is `test`, don't process financially.
	<deposit>	x	One for each account being deposited into
	<signature>	x	Depositor's signature. See Signature Rules table.
	<account>	x	Depositor's account. Refers to depositor's certificate.
	<cert>		Depositor's certificate. See Cert Verification rules
	<account>		Second account. Refers to depositor's certificate.
	<cert>		Second certificate. See Cert Verification rules
	<signature>	x	Bank's signature. See Signature Rules table.
	<cert>		Bank's certificate. See Certificate Verification Rules
	<signature>		Bank's signature on second account. See Signature Rules table.
	endcheck	x	Same number of endchecks as described in deposit block. Total of amounts must be same as amount in deposit block.

Verification Table - Document Verification Rules - Part Two			
Document	<Block>/Document	Required	Notes
endcheck	<action>	x	Function must be `payment`. Reason must be `process`, `test`, `resend`, or `info`. If reason is not `process` or `resend`, don't process financially.
	<endorsement>	x	
	<signature>	x	Endorser's signature. See Signature Rules table.
	<signature>		Co-endorser's (or counter-endorser's) signature. See Signature Rules table.
	<account>		Endorser's account. Required if check has paytobank
	<cert>		Endorser's certificate. See Certificate Verification Rules
	<cert>		co-endorser's (or counter-endorser's) certificate. See Certificate Verification rules.
	<signature>		Bank's signature. Required if account block present.
	<cert>		Bank's certificate. See Certificate Verification Rules Note: There are no direct FSML references to this block, if the bank's signature is omitted
	checkdoc	x	
	<attachment>		0 or more. Must be signed.
certcheck	<action>	x	Function must be `payment`. Reason must be `process`, `test`, `resend`, or `info`. If reason is not `process` or `resend`, don't process financially.
	<certification>	x	
	<signature>	x	certifier's signature. See Signature Rules table.
	<account>	x	certifier's account. Refers to certifier's certificate.
	<cert>		certifier's certificate. See Certificate Verification Rules
	<signature>	x	Bank's signature. See Signature Rules table.
	<cert>		Bank's certificate. See Certificate Verification Rules
	checkdoc	x	
presentment	<action>	x	Function must be `present`. Reason must be `process`, `test`, `resend`, or `info`. If reason is not `process` or `resend`, don't process financially.
	doclist	x	
	<bankstamp>		Should match the bank that the document was received from, but the server can't find out.
return	<action>	x	Function must be `present`. Reason must be `info`, `test`, `resend`, or `return`. If reason is not `resend` or `return`, don't process financially.
	doclist	x	
	<bankstamp>		

11.2 Verifying Block Contents

The following tables describe the verification requirements for Block Contents.

Verification Table - Block Verification Rules - Part One

Block	Field	Type	Notes
account	bankcode	string	Component of duplicate detection when in checkdoc
	bankacct	string	Verify against customer DB, if on-us check or deposit. Component of duplicate detection when in checkdoc
	bankser	integer	Verify against customer DB, if on-us check or deposit.
	custno	string	See payto verification section
	expdate	ISO date	Verify against customer DB, if on-us check or deposit. Test for stale date (date < current date is invalid).
	accttitle	string	Verify against customer DB if present, if on-us check or deposit.
	accttype	string	Verify against customer DB if present, if on-us check or deposit.
	bankname	string	Verify against customer DB if present, if on-us check or deposit.
	bankaddr	string	Verify against customer DB if present, if on-us check or deposit.
	bankphone	string	Verify against customer DB if present, if on-us check or deposit.
	bankfax	string	Verify against customer DB if present, if on-us check or deposit.
	bankemail	string	Verify against customer DB if present, if on-us check or deposit.
	acctrest	string	
	sigrest	string	Verify against sigtype in sig block
	certissuer	string	
	certserial	integer	
	micrmaskc	string	Verify according to encoding rules.
	micrmaskd	string	Verify according to encoding rules.
action	function	string	Note: <action> block must be the first block in a document.
	reason	string	
attachment	astatus	string	value must be `temporary` or `permanent`
	adata	string	Parameter `"encoding"` must be mime or text. If encoding is mime, and if verifying the field, must verify the adata field contains the 3 required MIME header lines: (`mime-version`, `content-type`, `content-transfer-encoding`)

Verification Table - Block Verification Rules - Part Two			
Block	Field	Type	Notes
bankstamp	date	ISO date	
	timestamp	UCT string	
	stampserial	integer	
	bankcode	integer	
	serverid	string	
	stampdata	string	
cert	certtype	string	`x509v1` or `x509v3`
	certissuer	string	must match any references to this
	certserial	integer	must match any references to this
	certdata	b64string	ASN.1/X.509
certification	date	ISO date	
	country	ISO country	
	checkbook	integer	
	cserial	integer	
check			Checkdata components are ordered.
	checknum	integer	Component of Duplicate Detection
	dateissued	ISO date	Sanity verification
	datevalid	ISO date	
	country	ISO country	
	amount	decimal	Non-negative value.
	currency	ISO currency	
	payto	string	This and following payto's form a subunit. For multiple payees these appear in repeated checkdatas for each payee. See Payto Verification
	paytobank	string	Payee bank code. If present, must be accompanied by paytoacct or paytocustno.
	paytoacct	string	Must be accompanied by paytobank.
	paytocustno	string	Must be accompanied by paytobank.
	paytoid	string	Must be accompanied by paytoidns.
	paytoidns t	string	Must be accompanied by paytoid.
	checkbook	integer	Component of duplicate detection.
	restrictions	string	Must be one of: `duration pnynmnd for deposit only` `all payees must endorse`
	payeracct	string	Unverified at payee.
	memo	string	Unverified
	info	string	Unverified
	conditions	string	
	legalnotice	string	Don't verify content; just non-blank if present

Verification Table - Block Verification Rules - Part Three			
Block	Field	Type	Notes
deposit	amount	decimal	must match sum of amount fields in checks, must be non-negative.
	currency	ISO currency	
	date	ISO date	
	country	ISO country	
	bankacct	string	
	items	integer	must match actual number of endorsed check items
endorsement			Endorsedata components are ordered.
	date	ISO date	
	country	ISO country	
	checkbook	integer	
	restrictions	string	Must be blank or `for deposit only`
	memo	string	
	payto	string	This and following payto's form a subunit. For multiple payees these appear in repeated checkdatas for each payee. See Payto Verification
	paytobank	string	Must be accompanied by paytoacct or paytocustno
	paytoacct	string	Must be accompanied by paytobank
	paytocustno	string	Must be accompanied by paytobank
	paytoid	string	Must be accompanied by paytoidns.
	paytoidns t	string	Must be accompanied by paytoid.

Verification Table - Block Verification Rules - Part Four			
Block	Field	Type	Notes
invoice	custacct	string	
	amount	decimal	
	currency	ISO currency	
	payto	string	
	paytobank	string	See check/paytobank. The bank has no responsibility for comparing this and the remaining payto items to those in the check or endorsement.
	paytoacct	string	See check/paytoacct
	paytocustno	string	See check/paytocustno
	paytoid	string	See check/paytoid
	paytoidns t	string	See check/paytoidns
	date	ISO date	
	remittance	string	Content only of concern to recipient.
	invdata	string	
message	retcode	string	
	msgtext	string	
	msgdata	string	
signature	blockref	string	Length limits depth of document nesting.
	hash	b64string	Param `alg` must be one of legal choices (Legal choices are `sha` or `md5`)
	nonce	integer	Must have at least 8 ascii characters. Spaces are not allowed.
	sigref	string	
	sigtype	string	Must be from list.
	certissuer	string	
	certserial	integer	
	algorithm	string	`md5/rsa` or `sha/dsa` or `sha/rsa` or `sha/ecdsa`. Verifiers must support all.
	timestamp	UCT string	
	location	string	
	username	string	Unverified field
	useraddr	string	Unverified field
	userphone	string	Unverified field
	useremail	string	Unverified field
	useridnum	string	Unverified field
	userotherid	string	Unverified field
	sig	b64string	with colon ":" delimiter for DSA or ECDSA
(private)	blkname	string	
	crit	string	`true` or `false`. Defaults to `true`.
	vers	decimal	defaults to `1.0`

11.3 Verifying Signatures

The following tables describe the verification requirements for Signatures.

Signature Rules Table				
Document	Signer	Sigref	sigtype	Blocks hashed
check	user	account	check	<action> <check> <account> <attachment> <invoice>
co-signed check	co-signer	account	co-sign	<action> <account> <attachment> <invoice> 1.<check>
counter-signed check	counter-signer	account	counter-sign	<action> <account> <attachment> <invoice> 1.<check> 1.<signature>
certified check	bank-teller	account	certification	<action> <certification> <account> 1.<check> 1. <signature>
endorsement	endorser	account cert	endorse	<action> <endorsement> <account> <attachment> 1.<check> 1.<signature>
co-endorsed check	co-endorser	account cert	co-endorse	<action> <endorsement> <attachment> 1.<endorsement>
counter-endorsed check	counter-endorser	account cert	counter-endorse	<action> <endorsement> <attachment> 1.<endorsement> 1.<signature>
deposit	depositor	account	deposit	<action> <deposit> <account> 1.<endorsement> 1.<signature> . .
all	bank	cert	bankacct	<account> <cert>

Notation. 1.<block> means <block> in document nested one below this one.

11.4 Payto Verification

There are three pairs of fields that may be used in a check, endorsement, or invoice to uniquely specify the person or institution being paid. These pairs of fields are...

- The <paytobank>/<paytoacct> pair
- The <paytobank>/<paytocustno> pair
- The <paytoid>/<paytoidsn> pair

A check can be written using any of the above 3 identification pairs. If it does not contain any of them, then no payto verification can occur, and the check may be endorsed and deposited by anybody.

There are sets of rules for payto verification of the endorsement document and rules for payto verification of the deposit document, as follows...

- If the <check> block contains any of the above 3 pairs then when the check is endorsed the information in the check must be verified by matching related information in the endorsement document, as follows...
  - If the <check> block in the check document contains <paytoacct>/<paytobank> fields then the endorsement document must contain an <account> block, and the <paytoacct> field in the <check> block in the check document must match the <bankacct> field in the <account> block in the endorsement document Similarly the <paytobank> field in the <check> block in the check document must match the <bankcode> field in the <account> block in the endorsement document
  - If the <check> block in the check document contains <paytocustno>/<paytobank> fields then the endorsement document must contain an <account> block, and the <paytocustno> field in the <check> block in the check document must match the <custno> field in the <account> block in the endorsement document Similarly the <paytobank> field in the <check> block in the check document must match the <bankcode> field in the <account> block in the endorsement document
  - If the <check> block in the check document contains <paytoid>/<paytoidns> fields then the endorsement document does not have to contain an account block, and the <paytoid>/<paytoidns> fields have to match fields in the Subject Distinguished Name in the X.509 certificate in the <cert> block used to sign the endorsement. The <paytoid> field must match the “Common Name” portion of the Subject Distinguished Name, and the <paytons> field must match the “DMDName” portion of the Subject Distinguished Name. If the <paytoidns> field is omitted, then the <paytoid> field and the “Common Name” which it matches must be globally unique, and the “DMDName” portion of the Subject Distinguished Name must not be present.
- When the endorsed check is deposited, there are two possible scenarios for verification of the deposit.
  1. The <endorsement> block in the endorsement document being deposited does not contain any of the three payto field pairs. This is the normal case, and implies that the money is being deposited to the same bank and account specified by the payto information in the check document. In this case, the payto information in the check document, which has been matched against information in the endorsement document, must also match the same information in the deposit document

2. The <endorsement> block in the endorsement document contains one of the above payto field pairs, e.g. <paytoacct>/<paytobank>, <paytocustno>/<paytobank>, <paytoid>/<paytoidns>. This indicates an "endorsement for transfer" is being performed. Then the above matching rules for matching between the check document and the endorsement document must be applied again but this time between the endorsement document and the deposit document. Note. This "endorsement for transfer" mechanism can only be applied once — i.e., a check "endorsed for transfer" cannot be further transferred but must be deposited.

11.5 Verifying Certificates

The rules for verifying certificates are...

1. Any certificates omitted (with prior agreement) must be obtained from local database or other means, using <certissuer> and <certserial> fields in referring block (<signature> or <account>).
2. Certificates may be verified by bytewise compare against copies kept in bank database, or cryptographically using public key of root, or via both methods.
3. effective date of document must fall between not-before and not-after dates in X509 certificate.
4. bank certificates must be checked to determine whether they have been revoked.

The cryptographic verification process for certificates in an FSML Document is as follows.

For each subdocument in the FSML document, perform the following steps...

1. Locate all the <cert> blocks in the subdocument. Extract the <certdata> field contents, convert the base64 string to binary, and parse and extract the x509 contents.
2. For each certificate in the subdocument, perform the following steps...
   (a) If the <certissuer> field contents in the <cert> block to be verified is the name of the root, skip step 2b, and use the root public key as the public key in step 2d.
   (b) Find another certificate in the same subdocument whose x509 "subject" distinguished name matches the "issuer" name in the certificate being verified(the <certissuer> field contents in the FSML must be the same as the issuer name in the X509, either one may be used). If it cannot be found in the same subdocument, a local cache or database of certificates may be used. Extract its subject public key for use in step 2d.
   (c) Extract the Signed Certificate data from the x509 in the certificate to be verified.
   (d) Calculate the hash on the data obtained in step 2c, using the hash algorithm specified in the X509 algorithm field.
   (e) Verify that the hash obtained in step 2d, when signed using the public key (obtained in step 2b or 2c), matches the claimed signature from the X509.
   (f) Verify that the current date falls between the notBefore and notAfter dates in the X509 certificate.
   (g) Verify that the certificate is not in a certificate revocation list (if appropriate).

Bibliography

[1] ISO, International Organization for Standardization, 1, rue de Varembe, Case Postale 56, CH-1211, Geneva 20, Switzerland http://www.itu.org. *ISO 8879 Information Processing Systems - Text and Office Systems - Standard Generalized Markup Language (SGML)*, 1988.

[2] R. Rivest. *RFC 1321 The MD5 Message-Digest Algorithm*. IETF, http://www.ietf.org, April 1992.

[3] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[4] U.S. Department of Commerce / National Institute of Standards and Technology, http://www.nist.gov. *FIPS Pub 180-1 - Secure Hash Standard*, April 1995.

[5] U.S. Department of Commerce / National Institute of Standards and Technology, http://www.nist.gov. *FIPS Pub 186 - Digital Signature Standard*, May 1993.

[6] Standards for Efficient Cryptography Group, http://www.secg.org/. *GEC1: Recommended Elliptic Curve Domain Parameters*, February 1999.

[7] CCITT, International Telegraphic Union, General Secretariat, Place Des Nations, CH-1211, Geneva 20, Switzerland. http://www.itu.org. *CCITT X.509 The Directory - Authentication Framework*, January 1995.

[8] Nicklaus Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions. *Communications of the ACM*, 22(11):822–823, November 1977.

[9] John Backus and Peter Naur. The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. *Proceedings of the International Conference on Information Processing*, June 1959.

[10] N. Borenstein and N. Freed. *RFC 2045 MIME (Multipurpose Internet Mail Extensions) - Part One: Format of Internet Message Bodies*. IETF, http://www.ietf.org, November 1996.

[11] American National Standards Institute, X3.4. *US-ASCII. Coded Character Set – 7-bit American Standard Code for Information Interchange*, 1986.

[12] ISO, International Organization for Standardization, 1, rue de Varembe, Case Postale 56, CH-1211, Geneva 20, Switzerland http://www.iso.ch. *ISO 8601 Data elements and interchange formats - Information interchange - Representation of dates and times*, 1988.

[13] RSA Laboratories, 2955 Campus Drive, Suite 400, San Mateo, CA 94403-2507 - http://www.rsa.com/rsalabs/. *PKCS1: RSA Encryption Standard - Version 1.5*, November 1993.

[14] ITU, International Telecommunications Union Place des Nations, CH-1211, Geneva 20, Switzerland. http://www.itu.org. *ISO/IEC 8824-1:1998 Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*, 1998.

[15] ITU, International Telecommunications Union Place des Nations, CH-1211, Geneva 20, Switzerland. http://www.itu.org. *ISO/IEC 8825-1:1998 Information Technology - Abstract Syntax Notation One (ASN.1): Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, 1998.

[16] N. Borenstein and N. Freed. *RFC 2046 MIME (Multipurpose Internet Mail Extensions) - Part Two: Media Types*. IETF, http://www.ietf.org, November 1996.

[17] ISO, International Organization for Standardization, 1, rue de Varembe, Case Postale 56, CH-1211, Geneva 20, Switzerland http://www.iso.ch. *ISO 3166 Codes for the representations of names of countries*, 1993.

[18] ISO, International Organization for Standardization, 1, rue de Varembe, Case Postale 56, CH-1211, Geneva 20, Switzerland http://www.iso.ch. *ISO 4217 Codes for the representations of currencies and funds*, 1995.

[19] American National Standards Institute, http://www.ansi.org. *X9.13 Placement and location of MICR Printing*, 1990.

[20] American National Standards Institute, http://www.ansi.org. *X9.37 Specification for Electronic Check Interchange*, 1994.

[21] ITU, International Telecommunications Union Place des Nations, CH-1211, Geneva 20, Switzerland. http://www.itu.org. *ISO/IEC 8824-2:1998 Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification*, 1998.

[22] ITU, International Telecommunications Union Place des Nations, CH-1211, Geneva 20, Switzerland. http://www.itu.org. *ISO/IEC 8824-3:1998 Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification*, 1998.

[23] ITU, International Telecommunications Union Place des Nations, CH-1211, Geneva 20, Switzerland. http://www.itu.org. *ISO/IEC 8824-4:1998 Information Technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications*, 1998.

[24] National Institute for Standards and Technology, http://csrc.nist.gov/encryption/. *Recommended Elliptic Curves for Federal Government Use*, May 1999.

[25] K. Moore. *RFC 2047 MIME (Multipurpose Internet Mail Extensions) - Part Three: Message Header Extensions for Non-ASCII Text*. IETF, http://www.ietf.org, November 1996.

[26] N. Borenstein and N. Freed. *RFC 2049 MIME (Multipurpose Internet Mail Extensions) - Part Five: Conformance Criteria and Examples*. IETF, http://www.ietf.org, November 1996.

Example Documents

Below is an example of an Electronic Check.

```
<fsml-doc docname="echeck187" type="check">
<action>
<blkname>act1
<crit>true
<vers>1.0
<function>payment
<reason>process
</action>
<check>
<blkname>check2
<crit>true
<vers>1.0
<checkdata>
<checknum>187
<dateissued>19970519
<datevalid>19970519
<country>us
<amount>100000.00
<currency>usd
<payto>Chili Pepper
</checkdata>
<checkbook>2048
<legalnotice>This instrument subject to check law
</check>
<signature>
<blkname>sig7
<crit>true
<vers>1.5
<sigdata>
<blockref>act1
<hash alg="sha">J4t/NI7s44IqSMTRl/1bkgABwug=
<blockref>check2
<hash alg="sha">vFnS/lVm9QaRDFAgtijkE24cazk=
<blockref>acct-111111111-00000001
<hash alg="sha">fF51C8MwtSVgeCQPOmzDTBjy1Zg=
<nonce>9D9BC5AA75
```

```
<sigref>acct-111111111-00000001
<sigtype>check
<algorithm>sha/dsa
<location>us
</sigdata>
<sig>
JInh43b1zYIydAELCmAo6j8nY/I=:KQuV+PAs9mFrnDoD3wtQKVoWIpU=
</signature>
<account>
<blkname>acct-111111111-00000001
<crit>true
<vers>1.5
<bankcode>111111111
<bankacct>00000001
<bankser>00000001
<expdate>19971231
<accttitle>John Q. Echecker
<accttype>checking
<bankname>BankA
<bankaddr>123 BankA Blvd, New York NY
<bankphone>(212)555-1234
<bankfax>(212)555-1235
<bankemail>echeck@banka.com
<acctrest>maximum amount 1.00 usd
<sigrest>chk:dep
<certissuer>/C=US/ST=MD/O=BANKA/OU=checking/
<certserial>1
<micrmaskc>:::111111111:00000001:%05n:%21m
<micrmaskd>:::111111111:00000001::%21m
</account>
<cert>
<blkname>cert-111111111-00000001
<crit>true
<vers>1.5
<certtype>x509v1
<certissuer>/C=US/ST=MD/O=BANKA/OU=checking/
<certserial>1
<certdata>
MIIB8DCCAbACAQEwCQYHKoZIzjgEAzA9MQswCQYDVQQGEwJVUzELMAkGA1UECBMC
TUQxDjAMBgNVBAoTBUJBTktBMREwDwYDVQQLEwhjaGVja2luZzAeFw05NzA0MTQw
MTU5MDBaFw05NzEwMTEwMTU5MDBaMFAxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJN
RDEOMAwGA1UEChMFQkFOS0ExETAPBgNVBAsTCGNoZWNraW5nMREwDwYDVQQDEwgx
MTExMDBBMTCB7jCBpgYHKoZIzjgEATCBmgJAjfKklEkidqo9JXWbsGhpy+rA2Dr7
jQz3y7gyTw14guXQdi/FtyEOr8Lprawyq3qsSWk9+/g3JMLsBzbuMcgCkQIUx3Mh
jHN+yO6ZO08t7TD0jtrOkV8CQGJtAng56goTQTFjpVtMtQApnVUilWzvyzv/EPOZ
ziwuccud5fokur9Y5beVIZJcnMQun29GSwiMxXKvU+bXiAIDQwACQGvv+18gw5/+
4ulXRZnrYGvOmRAIb5/38i+qci58sCjx0vrbzX+/T9Fq/8kNS6grKBn7p1SHPy9J
tvyMiVKK5ZEwCQYHKoZIzjgEAwMvADAsAhRST3iIPK/BCqc77R3cJPL06CEEKQIU
HIk1bb56d3VfEB51AxCRDOMl234=
</cert>
<signature>
<blkname>banksig6
<crit>true
<vers>1.5
<sigdata>
<blockref>acct-111111111-00000001
<hash alg="sha">fF51C8MwtSVgeCQPOmzDTBjy1Zg=
<blockref>cert-111111111-00000001
<hash alg="sha">1xzwMBQg7/rXMxC8k79xyotRTVY=
<nonce>9D9BC5AA75
<sigref>cert-111111111
<sigtype>bankacct
<algorithm>sha/dsa
<location>us
</sigdata>
<sig>
JInh43b1zYIydAELCmAo6j8nY/I=:mBnAYXvAb7Pm+EWU865jlQvEr7A=
</signature>
<cert>
<blkname>cert-111111111
<crit>true
<vers>1.0
<certtype>x509v1
<certissuer>/C=US/ST=MD/O=FSTC/OU=checking CA/
<certserial>1
<certdata>
MIIB3zCCAZ8CAQEwCQYHKoZIzjgEAzA/MQswCQYDVQQGEwJVUzELMAkGA1UECBMC
TUQxDTALBgNVBAoTBEZTVEMxFDASBgNVBAsTC2NoZWNraW5nIENBMB4XDTk3MDQx
NDAxNTEwMFoXDTk3MTAxMTAxNTEwMFowPTELMAkGA1UEBhMCVVMxCzAJBgNVBAgT
Ak1EMQ4wDAYDVQQKEwVCQU5LQTERMA8GA1UECxMIY2hlY2tpbmcwge4wgaYGByqG
SM44BAEwgZoCQI3ypJRJInaqPSV1m7BoacvqwNg6+40M98u4Mk8NeILl0HYvxbch
Dq/C6a2sMqt6rElpPfv4NyTC7Ac27jHIApECFMdzIYxzfsjumTtPLe0w9I7azpFf
AkBibQJ4OeoKE0ExY6VbTLUAKZ1VIpVs78s7/xDzmc4sLnHLneX6JLq/WOW3lSGS
XJzELp9vRksIjMVyr1Pm14gCA0MAAkBzJYSvMZO5CPQdOqaGgEIcZmHqJZ1BRtSP
IlhmmhiUYzddTDgEAniQOZLRii7Km2b0BGliunomQzymYhIxS9vrMAkGByqGSM44
BAMDLwAwLAIUVqCoTpl+sHct1ZJ1Mzjp1lsHlXUCFCaarpGAHZ6AuABKiSJeJ5FQ
ROpA
</cert>
</fsml-doc>
```

SGML Document Type Definition (DTD)

```
<!SGML  "ISO 8879:1986"
--                                      --
--  DTD for FSML Electronic Documents   --
--  First Draft  27 Feb 1996            --
--  Written by J. Kravitz IBM Research  --
--  Last Revision 02 Apr 1999           --
--  Version  1.50.0                     --
--                                      --

CHARSET
     BASESET  "ISO 646:1983//CHARSET
               International Reference Version (IRV)//ESC 2/5 4/0"
     DESCSET  0   9   UNUSED
              9   2   9
              11  2   UNUSED
              13  1   13
              14  18  UNUSED
              32  95  32
              127 1   UNUSED
     BASESET  "ISO Registration Number 100//CHARSET
               ECMA-94 Right Part of Latin Alphabet Nr. 1//ESC 2/13 4/1"
     DESCSET  128 32  UNUSED
              160 95  32
              255  1  UNUSED

CAPACITY SGMLREF
TOTALCAP 150000
GRPCAP 150000

SCOPE    DOCUMENT
SYNTAX
         SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
                           19 20 21 22 23 24 25 26 27 28 29 30 31 127 255
         BASESET  "ISO 646:1983//CHARSET
                   International Reference Version (IRV)//ESC 2/5 4/0"
         DESCSET  0 128 0
         FUNCTION RE          13
```

```
                  RS          10
                  SPACE       32
                  TAB SEPCHAR  9
         NAMING   LCNMSTRT ""
                  UCNMSTRT ""
                  LCNMCHAR "-"
                  UCNMCHAR "-"
                  NAMECASE GENERAL YES
                           ENTITY  NO
         DELIM    GENERAL  SGMLREF
                  SHORTREF SGMLREF
         NAMES    SGMLREF
         QUANTITY SGMLREF
                  NAMELEN  34
                  TAGLVL   100
                  LITLEN   1024
                  GRPGTCNT 150
                  GRPCNT   64

FEATURES
  MINIMIZE
    DATATAG  NO
    OMITTAG  YES
    RANK     NO
    SHORTTAG NO
  LINK
    SIMPLE   NO
    IMPLICIT NO
    EXPLICIT NO
  OTHER
    CONCUR   NO
    SUBDOC   NO
    FORMAL   YES
  APPINFO    NONE
>

<!DOCTYPE fsml [

<!ELEMENT fsml o o ( fsml-doc )>

<!ELEMENT fsml-doc - -      (
                             (
                             action        ,
                             (
                              fsml-doc     |
                              signature    |
                              check        |
                              deposit      |
                              endorsement  |
                              account      |
                              cert         |
                              attachment   |
                              invoice      |
                              message      |
                              bankstamp    |
                              bundle       |
                              cashletter
                              )+
                             )
                            )>

<!ATTLIST fsml-doc docname CDATA #REQUIRED
                   type    CDATA #REQUIRED >

<!ELEMENT action    - -     (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             function    &
                             reason
                             )
                             )>

<!ELEMENT signature - -     (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             sigdata     ,
                             sig
                              )
                             )>

<!ELEMENT sigdata   - -     (
                             ( blockref , hash )+ &
                             nonce       &
                             sigref?     &
                             sigtype     &
                             (certissuer , certserial)?  &
                             algorithm   &
                             timestamp?  &
                             location?   &
                             username?   &
                             useraddr?   &
                             userphone?  &
                             useremail?  &
                             useridnum?  &
                             userotherid?
                            )>

<!ELEMENT check     - -     (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             checkdata   &
                             checkbook   &
                             restrictions* &
                             payeracct?  &
                             memo?       &
                             info?       &
                             conditions? &
                             vara?       &
                             varb?       &
                             varc?       &
                             vard?       &
                             vare?       &
                             varf?       &
                             varg?       &
                             varh?       &
                             vari?       &
                             legalnotice
                             )
                            )>

<!ELEMENT checkdata - -     (
                             checknum     ,
                             dateissued   ,
                             datevalid    ,
                             country?     ,
                             amount       ,
                             currency     ,
                             (
                             payto        ,
                             (paytobank   , ( paytoacct | paytocustno) )*
                             (paytoid     , paytoidns )*
                             )+
                            )>

<!ELEMENT deposit   - -     (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             amount      &
                             currency    &
                             date        &
                             country?    &
                             items       &
                             bankacct    &
                             vara?       &
                             varb?       &
                             varc?       &
                             vard?       &
                             vare?       &
                             varf?       &
                             varg?       &
                             varh?       &
                             vari?
                             )
                            )>

<!ELEMENT endorsement - -   (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             endorsedata &
                             checkbook   &
                             restrictions* &
                             memo?
                             )
                            )>

<!ELEMENT endorsedata - -   (
                             date?        ,
                             country?     ,
                             amount?      ,
                             currency?    ,
                             (
                             payto        ,
                             (paytobank   , ( paytoacct | paytocustno) )*
                             (paytoid     , paytoidns )*
                             )*
                            )>

<!ELEMENT certification - - (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             date        &
                             country?    &
                             checkbook   &
                             cserial?
                             )
                            )>

<!ELEMENT account   - -         (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             bankcode    &
                             bankacct    &
                             bankser     &
                             custno      &
                             expdate     &
                             accttitle?  &
                             accttype?   &
                             bankname?   &
                             bankaddr?   &
                             bankphone?  &
                             bankfax?    &
                             bankemail?  &
                             acctrest*   &
                             sigrest?    &
                             certissuer  &
                             certserial  &
                             micrmaskc?  &
                             micrmaskd?  &
                             )
                            )>

<!ELEMENT cert      - -         (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             certtype    &
                             (certissuer , certserial)  &
                             certdata
                             )
                            )>

<!ELEMENT attachment - -        (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             astatus?    ,
                             adata
                            )>

<!ELEMENT invoice   - -     (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             custacct    &
                             amount      &
                             currency    ),
                             (
                             payto        ,
                             (paytobank   , ( paytoacct | paytocustno) )*
                             (paytoid     , paytoidns )*
                             )+, (
                             date?       &
                             remittance? &
                             invdata?
                             )
                            )>

<!ELEMENT message    - -    (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             retcode     &
                             msgtext     &
                             msgdata?
                             )
                            )>

<!ELEMENT bankstamp  - -    (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             date        &
                             timestamp   &
                             stampserial &
                             serverid?   &
                             stampdata?  &
                             bankcode
                             )
                            )>

<!ELEMENT bundle     - -    (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             items       &
                             amount
                             )
                            )>

<!ELEMENT cashletter - -    (
                             (
                             blkname     ,
                             crit?       ,
                             vers?
                             ),(
                             items       &
                             amount      &
                             bundles
                             )
                            )>

<!ELEMENT blkname       - O (#PCDATA)>
<!ELEMENT crit          - O (#PCDATA)>
<!ELEMENT vers          - O (#PCDATA)>

<!ELEMENT acctrest      - O (#PCDATA)>
<!ELEMENT accttitle     - O (#PCDATA)>
<!ELEMENT accttype      - O (#PCDATA)>
<!ELEMENT adata         - - (#CDATA)>
<!ATTLIST adata encoding (mime | text)  text >
<!ELEMENT algorithm     - O (#PCDATA)>
<!ELEMENT amount        - O (#PCDATA)>
<!ELEMENT astatus       - O (#PCDATA)>
<!ELEMENT bankacct      - O (#PCDATA)>
<!ELEMENT bankaddr      - O (#PCDATA)>
<!ELEMENT bankcode      - O (#PCDATA)>
<!ELEMENT bankemail     - O (#PCDATA)>
<!ELEMENT bankfax       - O (#PCDATA)>
<!ELEMENT bankname      - O (#PCDATA)>
<!ELEMENT bankphone     - O (#PCDATA)>
```

```
<!ELEMENT bankser       - O (#PCDATA)>
<!ELEMENT blockref      - O (#PCDATA)>
<!ATTLIST blockref req (true | false) true >
<!ELEMENT bundles       - O (#PCDATA)>
<!ELEMENT certdata      - O (#PCDATA)>
<!ELEMENT certissuer    - O (#PCDATA)>
<!ELEMENT certserial    - O (#PCDATA)>
<!ELEMENT certtype      - O (#PCDATA)>
<!ELEMENT checkbook     - O (#PCDATA)>
<!ELEMENT checknum      - O (#PCDATA)>
<!ELEMENT conditions    - O (#PCDATA)>
<!ELEMENT country       - O (#PCDATA)>
<!ELEMENT cserial       - O (#PCDATA)>
<!ELEMENT currency      - O (#PCDATA)>
<!ELEMENT custacct      - O (#PCDATA)>
<!ELEMENT custno        - O (#PCDATA)>
<!ELEMENT date          - O (#PCDATA)>
<!ELEMENT dateissued    - O (#PCDATA)>
<!ELEMENT datevalid     - O (#PCDATA)>
<!ELEMENT expdate       - O (#PCDATA)>
<!ELEMENT function      - O (#PCDATA)>
<!ELEMENT hash          - O (#PCDATA)>
<!ATTLIST hash alg (md5 | sha) #REQUIRED >
<!ELEMENT info          - O (#PCDATA)>
<!ELEMENT invdata       - - (#PCDATA)>
<!ELEMENT items         - O (#PCDATA)>
<!ELEMENT legalnotice   - O (#PCDATA)>
<!ELEMENT location      - O (#PCDATA)>
<!ELEMENT memo          - O (#PCDATA)>
<!ELEMENT micrmaskc     - O (#PCDATA)>
<!ELEMENT micrmaskd     - O (#PCDATA)>
<!ELEMENT msgtext       - O (#PCDATA)>
<!ELEMENT msgdata       - - (#PCDATA)>
<!ELEMENT nonce         - O (#PCDATA)>
<!ELEMENT payeracct     - O (#PCDATA)>
<!ELEMENT payto         - O (#PCDATA)>
<!ELEMENT paytoacct     - O (#PCDATA)>
<!ELEMENT paytobank     - O (#PCDATA)>
<!ELEMENT paytocustno   - O (#PCDATA)>
<!ELEMENT paytoid       - O (#PCDATA)>
<!ELEMENT paytoidns     - O (#PCDATA)>
<!ELEMENT reason        - O (#PCDATA)>
<!ELEMENT remittance    - O (#PCDATA)>
<!ELEMENT restrictions  - O (#PCDATA)>
<!ELEMENT retcode       - O (#PCDATA)>
<!ELEMENT serverid      - O (#PCDATA)>
<!ELEMENT sig           - O (#PCDATA)>
<!ELEMENT sigref        - O (#PCDATA)>
<!ELEMENT sigrest       - O (#PCDATA)>
```

```
<!ELEMENT sigtype       - O (#PCDATA)>
<!ELEMENT stampdata     - O (#PCDATA)>
<!ELEMENT stampserial   - O (#PCDATA)>
<!ELEMENT timestamp     - O (#PCDATA)>
<!ELEMENT useraddr      - O (#PCDATA)>
<!ELEMENT useremail     - O (#PCDATA)>
<!ELEMENT useridnum     - O (#PCDATA)>
<!ELEMENT username      - O (#PCDATA)>
<!ELEMENT userotherid   - O (#PCDATA)>
<!ELEMENT userphone     - O (#PCDATA)>
<!ELEMENT vara          - O (#PCDATA)>
<!ELEMENT varb          - O (#PCDATA)>
<!ELEMENT varc          - O (#PCDATA)>
<!ELEMENT vard          - O (#PCDATA)>
<!ELEMENT vare          - O (#PCDATA)>
<!ELEMENT varf          - O (#PCDATA)>
<!ELEMENT varg          - O (#PCDATA)>
<!ELEMENT varh          - O (#PCDATA)>
<!ELEMENT vari          - O (#PCDATA)>

]>
```

Differences between FSML 1.17 and FSML 1.50

D.1 Summary of differences

FSML 1.17 had significant syntactic and semantic differences from FSML 1.50, which are summarized here.

The following rules and specifications describe the differences between version 1.17 and version 1.50 FSML syntax and semantics. Note that mixture of FSML 1.50 and FSML 1.17 blocks in a single document is supported. In most cases this has no effect on the correctness of the document.

The only instance where mixing of FSML 1.17 blocks and FSML 1.50 blocks can have an impact on the verification of a document is when one of the pair of <signature> and <account> blocks is of one version and one is of the other version. Since the <sigtype> field in the <signature> block may be verified against the <sigrest> field in the <account> block, the following rules should be used....

sigrest/sigtype matching rules		
<signature>	<account>	Rule
FSML 1.17	FSML 1.50	Verifier cannot determine type of signature, except by context (which blocks are signed). This may be used to determine if <sigrest> allows this type of signature.
FSML 1.50	FSML 1.17	No verification is possible. <account> block allows any type of signature.

Hex Encoding

FSML 1.17 syntax used hexadecimal ASCII encoding to encode binary data. All binary data in FSML 1.50 syntax is now encoded using base64 encoding. This includes the <hash>, <sig>, and <certdata> fields. Any block containing hexadecimal-encoded binary data *must* have a <vers> lower than 1.5. Blocks whose version is 1.5 or later and which contain binary data *must* encode the data using base64 instead of hex ASCII.

Hash Contents

FSML 1.17 signature semantics calculated block hashes without including the block start and block end tags in the hash calculation. The hashed data consisted of all the bytes (ASCII octets) between the ending > of the start tag and the starting < of the end tag, exclusive of the angle brackets. FSML 1.50 signature semantics include the block start and block end tags in the hash calculations, including the angle brackets and the tag name strings they enclose. Any <signature> block containing <hash> values calculated without inclusion of the referenced block begin and end tags must have a <vers> lower than 1.5.

blockref attribute

FSML 1.50 syntax for the <blockref> tag now supports the "req=" attribute. This was not supported by FSML 1.17 syntax. Any signature block containing a <blockref> tag with a "req=" attribute must not have a <vers> lower than 1.5.

Private Blocks

Private blocks (blocks whose begin tags start with "x:" were not supported prior to FSML 1.50. However, since mixing of FSML 1.17 blocks and FSML 1.50 blocks is permitted, private blocks may be generated in a document where all other blocks use FSML 1.17 syntax.

New Fields

A number of fields were added in FSML 1.50. The following fields were not supported in FSML 1.17. Blocks containing these fields must not have a <vers> lower than 1.5.

- The <sigtype> field in the <signature> block.
- The <info> field in the <check> block.
- The <sigrest> field in the <account> block.
- The <paytoid> and <paytoidns> fields in the <check>, <endorsement> and <invoice> blocks.
- The <custno> field in the <account> block.
- The <micrmaskc> and <micrmaskd> fields in the <account> block, which replace the <micrmaskc1>, <micrmaskc2>, <micrmaskc3>, <micrmaskd1>, <micrmaskd2>, <micrmaskd3> fields which were used in the the FSML 1.17 version of the <account> block.
- the <vara> through <vari> fields in the <check> block or <deposit> block.

Deleted Fields

A number of fields were deleted in FSML 1.50. The following fields were defined in FSML 1.17 but never used. The are no longer allowed in any FSML document.

- The <paytokey> field in the <check>, <endorsement> and <invoice> blocks.
- The <amount> and <currency> fields in the <endorsement> block.

co-signing

Nesting of documents to add co-signers and co-endorsers is no longer supported in FSML 1.50 syntax. Co-signatures and counter-signatures are added as blocks into the existing document. The FSML 1.17 syntax for co-signatures, counter-signatures, co-endorsers and counter-endorsers was never fully defined or implemented, and is therefore deprecated.

bankstamps

Nesting of documents to add new bankstamps is no longer used in FSML 1.50 syntax. Additional bankstamps are added as blocks into the existing document. FSML 1.17 syntax used an outer document to contain new bankstamps.

certificates

FSML 1.17 <cert> blocks only allowed X.509 Version 1 certificates. FSML 1.50 <cert> blocks allow either X.509 Version 1 or X.509 Version 3 certificates.

algorithms

Support for the Elliptic Curve cryptographic signature algorithm ECDSA was added to FSML 1.50.

D.2 Definitions of FSML 1.17 Deprecated Blocks

This section describes the blocks which have been changed from FSML 1.17 to FSML 1.50. Use of these blocks, while deprecated, is still allowed.

D.2.1 Version 1.0 echeck Signature Block Definition

```
<signature>
<blkname>namestring
<crit>true
<vers>1.0
<sigdata>
<blockref>dnamestring
<hash alg="sha">hexstring
<blockref>dnamestring
<hash alg="sha">hexstring
  ...
<blockref>dnamestring
<hash alg="sha">hexstring
<nonce>valuestring
<sigref>namestring
<certissuer>namestring
<certserial>number
<algorithm>namestring
<timestamp>valuestring
<location>valuestring
<username>valuestring
<useraddr>valuestring
<userphone>valuestring
<useremail>valuestring
<useridnum>valuestring
<userotherid>valuestring
</sigdata>
<sig>hexstring
</signature>
```

Figure D.23: Version 1.0 echeck Signature block element definition

Version 1.0 echeck Signature Block Field Definitions

blockref (**required**) The signature block contains one or more <blockref> fields, each of which contains the unique block name of the associated block being signed. All of the block references must appear immediately before their respective hashes (see below). The <blockref> and <hash> pairs may be repeated multiple times to sign multiple blocks.

hash

(**required**) This field contains the actual hash of the respective block. Each <hash> start tag must have an attribute which specifies the algorithm used to perform the hash. The currently allowed attribute values are **md5**[2] or **sha**[4]. The `alg=` attribute is required. The use of **md5** is deprecated. Other hash algorithms may be supported in future. It is not required that the same hash algorithm be used for each of the blockrefs in a signature block. All hashes are encoded in “network byte order”, which means that the most significant bytes are leftmost (first). Note: Attribute values must be enclosed in quotes.

nonce

(**required**) This is a nonce, or one-time random number, used to “salt” the hashed data to discourage cryptanalysis attacks. The nonce value can be any string of random ASCII characters from within the set of allowed FSML characters (see Character Encoding above) not including whitespace.

Note to Implementors:

Although any FSML character except whitespace is allowed in the <nonce> value, it is permitted, and implementors may find it convenient, to generate a random number and include it in the <nonce> field represented as a decimal integer, a floating-point number, a hexadecimal-encoded octet string, or as a base64-encoded octet string. Note that the use of this string in the hash is purely as a sequence of ASCII octets. The fact that it may have been created as an ASCII representation of a floating point number or integer, or hexadecimal number is irrelevant to its use in the hash data.

sigref

(optional) This is the block name of the <account> block which contains a reference to the certificate block, or it is the block name of the <cert> block itself, for signatures that don’t need account blocks. This field, although optional, is only optional when an agreement is in place indicating that the recipient of the document does not need the certificate in order to process the document.

certissuer

(optional) This field contains the unique distinguished name of the issuer of the certificate[7]. It should only be specified if the <account> and <cert> blocks are not being sent with this document, and only when the blocks being signed do not require an account — e.g., an endorsement. See the description of the <certissuer> field in the <cert> block for the syntax used to specify this field.

certserial

(optional) This field contains the unique certificate serial number assigned by the issuer of the certificate. It should only be specified if the <account> and <cert> blocks are not being sent with this document, and only when the blocks being signed do not require an account — e.g., an endorsement.

algorithm

(**required**) This string indicates the algorithm used to sign the signature block. It may be **md5/rsa**[3] or **sha/dsa**[5] or **sha/rsa** or **sha/ecdsa**[6]. Note: Implementors of code that is used to sign FSML Electronic Documents may choose to support only one of the above three possible signing algorithms. Implementors of code that is used to verify FSML Electronic Documents must support all three algorithms. This ensures interoperablity. The use of md5 is deprecated.

timestamp

(optional) This field specifies the time that the document was signed. It must be in Universal time (i.e., GMT) specified as CCYYMMDDThhmmssZ, where the `T` and `Z` are literal characters, and where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month, “DD” is the day, “hh” is the hour, “mm” is the minute and ss is the second[12].

location

(optional) This field specifies location/country where the document was signed.

username — (optional) This is an identification string containing the account user’s name. It is optionally inserted into the check by the Electronic Checkbook hardware token.

This field, and the 5 following fields are considered the electronic equivalent of the data usually printed on a paper check by the Check Printing company. This data is supplied by the checkbook owner to the bank at the time the electronic checking account is established but it is not certified to be correct or accurate by the bank. The data is inserted into the Electronic Checkbook when the Checkbook is initialized at the bank, and may also be corrected or updated later by the bank using administrative checkbook functions and passwords.

This data is then inserted, under control of the user, into the check by the Electronic Checkbook, however the data cannot be changed or deleted by the user once the check is created. It therefore supplies a form of identification sometimes required by check guarantee organizations or merchants. The user may select, when writing a check, which of the 6 identification fields are to be inserted into the check, in any combination, or may select none of them.

useraddr — (optional) This is an identification string containing the account user’s address. It is optionally inserted into the check by the Electronic Checkbook hardware token.

userphone — (optional) This is an identification string containing the account user’s phone number. It is optionally inserted into the check by the Electronic Checkbook hardware token.

useremail — (optional) This is an identification string containing the account user’s email address. It is optionally inserted into the check by the Electronic Checkbook hardware token.

useridnum — (optional) This is an identification string containing the account user’s identification number. It is optionally inserted into the check by the Electronic Checkbook hardware token.

userotherid — (optional) This is an identification string containing any user identification the user wishes (e.g., company name). It is optionally inserted into the check by the Electronic Checkbook hardware token.

sig — (**required**) This is a hexadecimal encoding of the actual signature data. For certain algorithms, the field is split into two portions using a colon `":"`. For DSA or ECDSA, the field contains the two portions of the signature as r:s, where r and s are long hexdecimal ASCII strings. For RSA, only a single string is specified, with no colon separator. All signatures are encoded in “network byte order”, which means that the most significant bytes are leftmost (first).

Signature Calculation

The calculation of the Signature is performed as follows...

1. The <nonce> value is created (by the electronic checkbook) as a random number. The nonce value can be any string of random ASCII characters from within the set of allowed FSML characters (see Character Encoding above) not including whitespace.

   Note to Implementors:

   Although any FSML character except whitespace is allowed in the <nonce> value, it is permitted, and implementors may find it convenient, to generate a random number and include it in the <nonce> field represented as a decimal integer, a floating-point number, a hexadecimal-encoded octet string, or as a base64-encoded octet string. Note that the use of this string in the hash is purely as a sequence of ASCII octets. The fact that it may have been created as an ASCII representation of a floating point number or integer, or hexadecimal number is irrelevant to its use in the hash data.

2. The <nonce> value is logically prepended to the subject block contents before hashing. This includes the tag string "<nonce>" — e.g., if the nonce value is 12345, the characters `<nonce>12345` are logically prepended to the subject block before hashing.

3. The hash is calculated using the contents of the subject block, (with the <nonce> prepended) excluding the block start tag and block end tag, with the exception of all carriage returns, line feeds, and trailing spaces on a line. Leading and embedded spaces in a line are included in the hash. SGML entities (i.e., character names enclosed between an ampersand and a semicolon) are left untranslated when hashing.

4. The resulting hash value is inserted into the <hash> entry (encoded as hexadecimal ASCII) in the signature block.

5. Steps 2 through 4 are repeated for each block to be signed.

6. A second hash calculation is performed on the contents of the <sigdata> sub-block, which contains the previously calculated hashes, their block references, and the <nonce>. This should include all characters between the <sigdata> tag and the </sigdata> tag, not including the tags themselves, again omitting all carriage returns, line feeds, and trailing spaces. This second hash is then signed using the private key in the electronic checkbook. The result is the signature which is inserted (encoded as hexadecimal ASCII) into the signature block as the value for the <sig> tag.

Signature Verification

The verification of the Signature is performed as follows...

1. The following steps are repeated for each block referenced by a <blockref> tag in the signature. If the referenced block is not present, and `<blockref req="false">` was specified, the block is assumed to have been detached. The following steps are not performed for this block, and this block's absence is not considered to invalidate the document.

    (a) The <nonce> value from the signature block is logically prepended to the referenced blocks contents before hashing. This includes the tag string "<nonce>" — e.g., if the nonce value is 12345, the characters `<nonce>12345` are logically prepended to the referenced blocks contents before hashing.

    (b) A hash is calculated using the contents of the referenced block, (with the <nonce> prepended) excluding the block start tag and block end tag, with all characters in between, with the exception of all carriage returns, line feeds, and trailing spaces on a line. Leading and embedded spaces in a line are included in the hash. SGML entities (i.e., character names enclosed between an ampersand and a semicolon) are left untranslated when hashing. The hash algorithm to be used is specified in the `hash=` attribute in the <hash> tag for the referenced block.

    (c) The resulting hash value is compared to the <hash> entry in the signature block.

    (d) If the hashes do not match exactly, the signature fails verification.

2. The contents of the <sig> field are processed using the public key found by following the <sigref> tag. This tag will either point to an <account> block, or a <cert> block. If the <sigref> tag points (by name) to a <cert> block, the public key will be found in the <certdata> field in that block. (Parsing of

the <certdata> field may be required to extract the public key — e.g., an X.509 certificate parser may be required). If the <sigref> field points to an account block, the account block will, in turn, point to a <cert> block via the <certissuer> and <certserial> fields. The <cert> block whose <certissuer> and <certserial> fields match those in the <account> block contains the public key. The signature algorithm to be used is specified in the <algorithm> field.

3. A second hash calculation is performed on the contents of the <sigdata> sub-block, which contains the previously calculated hashes, their block references, and the <nonce>. This should include all characters between the <sigdata> tag and the </sigdata> tag, not including the tags themselves, again omitting all carriage returns, line feeds, and trailing spaces. The hash algorithm to be used is specified in the <algorithm> field.

4. The processed <sig> field is compared to the hash calculated in the previous step. If this comparison fails, the signature fails verification. If the comparison succeeds, the signature has verified successfully.

D.2.2 Version 1.0 Check Block Definition

This block contains the key data for an FSML Electronic Check.

Multiple signers/certificates may be required, as determined by the restrictions field in the signer's account block.

```
<check>
<blkname>namestring
<crit>true
<vers>1.0
<checkdata>
<checknum>numstring
<dateissued>valuestring
<datevalid>valuestring
<country>namestring
<amount>amountstring
<currency>valuestring
<payto>valuestring
<paytobank>valuestring
<paytoacct>valuestring
<paytocustno>valuestring
</checkdata>
<checkbook>numstring
<restrictions>valuestring
<payeracct>valuestring
<memo>valuestring
<conditions>valuestring
<legalnotice>valuestring
</check>
```

Figure D.24: Version 1.0 Check block element definition

Version 1.0 Check Block Field Definitions

checkdata

(**required**) This is an enclosing sub-block. It is used to contain all of the check fields that will be interpreted and/or logged by the Electronic Checkbook hardware token. To simplify parsing by this token, the <checkdata> sub-block contents must be in the order specified.

checknum

(**required**) This is the unique check number created by the Electronic Checkbook hardware token.

dateissued

(**required**) This is the effective date of the check, supplied by the check issuer. It is not necessarily the date the check was written. The date must be specified in the ISO standard[12] format CCYYMMDD, where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month and "DD" is the day. Document recipients are free to process or ignore this date as they choose.

datevalid

(**required**) This is the effective date of validity for the check, supplied by the check issuer. It is not necessarily the date the check was written. Currently, it should always be the same date as the <dateissued>. Other uses and values for this field will be described in a later edition of this specification. The date must be specified in the ISO standard[12] format CCYYMMDD, where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month and "DD" is the day.

country

(optional) This is the 2 letter ISO country code[17] of the location where the check is to be considered written.

amount

(**required**) A decimal number containing the amount of the check.

currency

(**required**) A 3 letter ISO currency code[18].

payto

(**required**) This is a string which is the name or other check-issuer specified identification of the payee. This field is used for informational purposes only — i.e., creation of statement information. It is not verified against other data.

This field, and the following three fields form a subunit which identifies one of the possible payees for the check. If multiple payees are being specified, then the subunit may be repeated, with the fields in the same order for each payee (excluding optional fields).

paytobank

(optional) This is a field which if specified must be accompanied by either the <paytoacct> field, or the <paytocustno> field, and which contains the bank code of the payee.

paytoacct

(optional) This is a field which if specified must be accompanied by the <paytobank> field, and which contains the account number of the payee.

paytocustno

(optional) This is a field which if specified must be accompanied by the <paytobank> field. It contains the customer number of the payee at the payees bank. Some banks may use this in lieu of an account number.

checkbook

(**required**) This is an integer, supplied by the Electronic Checkbook hardware token, which is the bank-unique serial number of the checkbook.

restrictions

(optional) This is a string containing restriction information about the specific check. The field may be repeated. It must be one of the following character strings.

- **duration p**nyn**m**n**d**

- **for deposit only**
- **all payees must endorse**

The **p**n**y**n**m**n**d** is an ISO standard[12] method of representing duration, where each "n" is a one or two digit number, and the **p** character is required. The numbers before **y**, **m**, and **d**, represent years, months and days, respectively. The duration (valid lifetime of a check) defaults to 60 days if not otherwise specified here.

payeracct (optional) This is a field containing a character string which is the account information of the payer at the payees business — i.e., the number that the payee uses to determine who is paying, or why it is being paid. This is not a bank account number. As an example, this is the payer's account number at the electric utility, on a check used to pay an electricity bill.

memo (optional) A character string field, used for any purpose the check issuer wishes. It is not processed by the bank.

conditions (optional) A character string field, used to specify any conditions between the check issuer and endorser. Not processed by the bank.

legalnotice (**optional**) If present, this field must be inserted by any software that creates a new <check> block as containing one of the following two character strings.... **This instrument subject to check law** for normal echecks, or **This instrument subject to U.S Treasury check law** for Treasury echecks. Software that receives and processes echecks may check that the field is non-blank if present, but must not check that the strings contain the above values, as other values may be possible in future. This field is for legal notification purposes.

D.2.3 Version 1.0 Deposit Block Definition

This block contains a electronic deposit slip, which is bound (via a signature block) to one or more endorsement blocks before being sent to a bank or other financial institution for deposit. The associated endorsement blocks must also have check blocks bound to them.

```
<deposit>
<blkname>namestring
<crit>true
<vers>1.0
<amount>amountstring
<currency>valuestring
<date>valuestring
<country>valuestring
<items>number
<bankacct>valuestring
</deposit>
```

Figure D.25: Version 1.0 Deposit block element definition

Version 1.0 Deposit Block Field Definitions

date (**required**) This is the effective date of the deposit slip, supplied by the depositor. It is not necessarily the date the deposit slip was created. The date must be specified in the ISO standard[12] format CCYYMMDD, where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month and “DD” is the day.

amount (**required**) A decimal number containing the total amount of the deposit.

currency (**required**) A 3 letter ISO currency code[18].

items (**required**) An integer specifying the total number of checks or other items being deposited.

country (optional) A 2 letter ISO Country code[17].

bankacct (**required**) This is a string containing the account number of this account in the issuing bank. This indicates the account that the funds are being deposited into.

D.2.4 Version 1.0 Endorsement Block Definition

This block contains a digital endorsement of a financial document, usually a check. It must be bound (via a signature block) to the check it endorses.

```
<endorsement>
<blkname>namestring
<crit>true
<vers>1.0
<endorsedata>
<date>valuestring
<country>valuestring
<payto>valuestring
<paytobank>valuestring
<paytoacct>valuestring
<paytocustno>valuestring
</endorsedata>
<checkbook>numstring
<restrictions>valuestring
<memo>valuestring
</endorsement>
```

Figure D.26: Version 1.0 Endorsement block element definition

Version 1.0 Endorsement Block Field Definitions

endorsedata (**required**) This is an enclosing sub-block. It is used to contain all of the endorsement fields that will be interpreted and/or logged by the Electronic Checkbook hardware token. To simplify parsing by this token, the <endorsedata> sub-block contents must be in the order specified. Since all of the fields enclosed in the <endorsedata> sub-block are optional, the sub-block may be empty. It is required to have the <endorsedata> and </endorsedata> tags in any case.

date (optional) This is the effective date of the endorsement, supplied by the endorser. It is not necessarily the date the endorsement was created. The date must be specified in the ISO standard[12] format CCYYMMDD, where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month and "DD" is the day.

country (optional) This is the 2 letter ISO country code of the location where the endorsement is to be considered written[17].

payto (optional) This is a string which is the name or other endorser identification of the ultimate payee or next holder in due course. This field is used for informational purposes only — i.e., creation of statement information. It is not verified against other data.

This field, and the following 3 fields form a subunit which identifies one of the possible payees for the check. If multiple payees are being specified, then the subunit may be repeated, with the 4 fields in the same order for each payee (excluding optional fields).

paytobank (optional) This field if specified must be accompanied by either the <paytoacct> field, or the <paytocustno> field, and which contains the bank code of the ultimate payee.

paytoacct (optional) This field if specified must be accompanied by the <paytobank> field, and which contains the account number of the ultimate payee.

paytocustno (optional) This field if specified must be accompanied by the <paytobank> field. It contains the customer number of the ultimate payee at the their bank. Some banks may use this in lieu of an account number.

checkbook (**required**) This is an integer, supplied by the Electronic Checkbook hardware token, which is the bank-unique serial number of the checkbook.

restrictions (optional) This is a string containing restriction information about the specific check being endorsed. The field may be repeated. It must be one of the following character strings.

- **for deposit only**

memo (optional) A character string field, used for any purpose the endorsement issuer wishes.

D.2.5 Version 1.0 Account Block Definition

This block contains information about the account of the check issuer, or endorser. It is always used in combination with a certificate block.

```
<account>
<blkname>namestring
<crit>true
<vers>1.0
<bankcode>valuestring
<bankacct>valuestring
<bankser>numstring
<expdate>valuestring
<accttitle>valuestring
<accttype>valuestring
<bankname>valuestring
<bankaddr>valuestring
<bankphone>valuestring
<bankfax>valuestring
<bankemail>valuestring
<acctrest>valuestring
<sigrest>valuestring
<certissuer>valuestring
<certserial>number
<micrmaskc1>valuestring
<micrmaskc2>valuestring
<micrmaskc3>valuestring
<micrmaskd1>valuestring
<micrmaskd2>valuestring
<micrmaskd3>valuestring
</account>
```

Figure D.27: Version 1.0 Account block element definition

Version 1.0 Account Block Field Definitions

blkname (**required**) The <blkname> field in an <account> block is slightly different than the “generic” <blkname>. Since the <account> block is signed by the bank issuing the electronic token, and is stored in the token, it is not changeable at runtime by FSML generating software. Thus the <blkname> chosen must be guaranteed to be unique for all subsequent documents. It is recommended (but not required) that a block naming convention be used to allow this. The recommended convention is that the name be suffixed with information that is unique to the account block, so that the same name would never be used by other account blocks in the same FSML document. As an example, an account block issued by a bank whose Bank Routing Code is 123456789, for a customer whose account number is 987654321 might have a blockname of `acct-123456789-987654321`.

bankcode (**required**) This is a string containing the unique bank routing code of the issuing bank.

bankacct (**required**) This is a string containing the account number of this account in the issuing bank.

bankser (**required**) This is a number containing the account block serial number of this account in the issuing bank. This must be unique for all account blocks within an issuing bank code.

expdate (**required**) This is the expiration date of this account block. The date must be specified in the ISO standard[12] format CCYYMMDD, where “CC” is the century (currently 19, soon 20), “YY” is the year, “MM” is the month and “DD” is the day.

accttitle (optional) This is a string containing the account title.

accttype (optional) This is a string containing the account type.

bankname (optional) This is a string containing the bank’s name.

bankaddr (optional) This is a string containing the bank’s address.

bankphone (optional) This is a string containing the bank’s phone number.

bankfax (optional) This is a string containing the bank’s fax number.

bankemail (optional) This is a string containing the bank’s email address.

acctrest (optional) This is a string containing any restrictions on the account. It must be one of the following character strings. The field may be repeated.

- **minimum amount** nnnnnnn.nn ccc
- **maximum amount** nnnnnnn.nn ccc
- n **signatures required**
- n **signatures required above amount** nnnnn.nn ccc
- **special processing**
- **currency** ccc
- **duration p**n**y**n**m**n**d**

The “n” in the above restrictions represents a *number*. The “nnnnnnnn.nn” in the above restrictions represents an *amountstring*. The “ccc” is a 3 letter ISO currency code[18]. This indicates the currency being specified in the **amount**, or the currency that checks are being restricted to by the bank (in the **currency** ccc restriction). The **p**n**y**n**m**n**d** is an ISO standard[12] method of representing duration, where each “n” is a one or two digit number, and the **p** character is required. The Numbers before **y**, **m**, and **d**, represent years, months and days, respectively. The duration (valid lifetime of a check) defaults to 60 days if not otherwise specified here or in the <restrictions> field in the <check> block. If specified in both places, the shortest duration takes precedence.

certissuer (**required**) This field contains the unique distinguished name of the issuer of the certificate[7] which can be used to verify signatures on FSML documents containing this account block. See the description of the <certissuer> field in the <cert> block for the syntax used to specify this field.

certserial (**required**) This field contains the unique certificate serial number assigned by the issuer of the certificate which can be used to verify signatures on FSML documents containing this account block.

micrmaskc1 (optional) This field is used when creating the MICR-line data for an E-check. This field corresponds to the field known as the “aux on us” field, called Field 2 in ANSI X9.37-1994, record type 25, also known as Field 7 in ANSI X9.13-1990. The encoding string must specify the locations for 15 characters, which is the size of the field. The field is allowed to contain fixed characters from the set “0123456789/-b”, and may optionally contain the specifier (“n” or “nnn..”) for Check serial number.

micrmaskc2 (optional) This field is used when creating the MICR-line data for an E-check. This field corresponds to the field known as the “on us” field, Field 6 in ANSI X9.37-1994, record type 25, which corresponds to Fields 2, 3, and 4 of the MICR line, in ANSI X9.13-1990. The encoding string must specify the locations for 20 characters, which is the size of the field. The field is allowed to contain fixed characters from the set “0123456789/-b”, and may optionally contain the specifier (“n” or “nnn..”) for Check serial number, or the specifier (“a” or “aaa...”) for Account number, or both.

micrmaskc3 (optional) This field is used when creating the MICR-line data for an E-check. This field corresponds to the field known as the “external processing code” field, Field 3 in ANSI X9.37-1994, record type 25, also known as Optional Field 6 (and sometimes known as Position 44), in ANSI X9.13-1990. The field is encoded as a single character from the set “0123456789/-b”.

micrmaskd1 (optional) This field is used when creating the MICR-line data for an E-deposit. This field corresponds to the field known as the “aux on us” field, called Field 2 in ANSI X9.37-1994, record type 25, also known as Field 7 in ANSI X9.13-1990. The encoding string must specify the locations for 15 characters, which is the size of the field. The field is allowed to contain fixed characters from the set “0123456789/-b”.

micrmaskd2 (optional) This field is used when creating the MICR-line data for an E-deposit. This field corresponds to the field known as the “on us” field, Field 6 in ANSI X9.37-1994, record type 25, which corresponds to Fields 2, 3, and 4 of the MICR line, in ANSI X9.13-1990. The encoding string must specify the locations for 20 characters, which is the size of the field. The field is allowed to contain fixed characters from the set “0123456789/-b”, and may optionally contain the specifier (“a” or “aaa...”) for Account number.

micrmaskd3 (optional) This field is used when creating the MICR-line data for an E-deposit. This field corresponds to the field known as the “external processing code” field, Field 3 in ANSI X9.37-1994, record type 25, also known as Optional Field 6 (and sometimes known as Position 44), in ANSI X9.13-1990. The field is encoded as a single character from the set “0123456789/-b”.

D.2.6 Version 1.0 Certificate Block Definition

This block contains an encoded X.509 certificate[7].

```
<cert>
<blkname>namestring
<crit>true
<vers>1.0
<certtype>valuestring
<certissuer>valuestring
<certserial>number
<certdata>hexstring
</cert>
```

Figure D.28: Version 1.0 Certificate block element definition

Version 1.0 Certificate Block Field Definitions

blkname (**required**) The <blkname> field in a <cert> is slightly different from the "generic" <blkname>. Since the <cert> block is signed by the authority issuing the electronic token, and is probably stored in the token, it is not changeable at runtime by FSML generating software. Thus the <blkname> chosen must be guaranteed to be unique for all subsequent documents. It is recommended (but not required) that a block naming convention be used to allow this.

certtype (**required**) This field indicates the type of certificate contained in the block. The only supported value for version 1.0 <cert> blocks is **x509v1** This value must correspond to the data in the actual certificate contained in the <certdata> field.

certissuer (**required**) This field contains the unique distinguished name of the issuer of the certificate. The certificate issuer string uses the fields from the distinguished name in the ASN.1 X509 certificate, separated by slashes, and using a TAG= identification of the name field type. The different name fields use the following identification tags:

Country	C=
DMDName	DMD=
Commonname	CN=
Orgname	O=
Orgunit	OU=
Title	T=

Thus, an example of an issuer string would be...

```
/C=US/ST=New York/O=FIRSTBANK_ANYTOWN/OU=checking/
```

This value must correspond to the data in the actual certificate contained in the <certdata> field. Although X.509 distinguished names allow additional fields, FSML only supports the ones named above.

certserial (**required**) This field contains the unique certificate serial number assigned by the issuer of the certificate. This value must correspond to the data in the actual certificate contained in the <certdata> field.

certdata (**required**) This contains the hexadecimal ASCII-encoded binary value of the ASN.1 DER[14][15] encoded X.509 certificate.

D.2.7 Version 1.0 Invoice Block Definition

This block contains invoice information used by a merchant or other payee to request that the payer create a check using the information contained in the invoice. It is also used as remittance information by the payer to be attached to the check being used to pay the invoice.

```
<invoice>
<blkname>namestring
<crit>true
<vers>1.0
<custacct>valuestring
<amount>amountstring
<currency>valuestring
<payto>valuestring
<paytobank>valuestring
<paytoacct>valuestring
<paytocustno>valuestring
<date>valuestring
<remittance>valuestring
<invdata>
...
</invdata>
</invoice>
```

Figure D.29: Version 1.0 Invoice block element definition

Version 1.0 Invoice Block Field Definitions

custacct (optional) This field contains the customers account number or code in the merchants (payees) accounting system. It should be returned in the <payeracct> field in the check used to pay the invoice.

amount (**required**) A decimal number containing the amount being invoiced, or the amount due.

currency (**required**) A 3 letter ISO currency code[18].

payto (**required**) This is a string which is the name or other check-issuer specified identification of the payee. This field is used for informational purposes only — i.e., creation of statement information. It is not verified against other data.

This field, and the following 3 fields form a subunit which identifies one of the possible payees for the check. If multiple payees are being specified, then the subunit may be repeated, with the fields in the same order for each payee (excluding optional fields).

paytobank (optional) This is a field which if specified must be accompanied by either the <paytoacct> field, or the <paytocustno> field, and which contains the bank code of the payee.

paytoacct (optional) This is a field which if specified must be accompanied by the <paytobank> field, and which contains the account number of the payee.

paytocustno (optional) This is a field which if specified must be accompanied by the <paytobank> field. It contains the customer number of the payee at the payees bank. Some banks may use this in lieu of an account number.

date (optional) The date that the payment is due. The date must be specified in the ISO standard[12] format CCYYMMDD, where "CC" is the century (currently 19, soon 20), "YY" is the year, "MM" is the month and "DD" is the day.

remittance (optional) This field contains any remittance identification number or string that is being used to correlate this payment with other systems. It may contain any number or other identifier that would indicate to the recipient which invoice or which remittance item is associated with this FSML document.

invdata (optional) This field contains any other data that may be associated with the invoice — e.g., an purchase order or other purchase information.

E-Mail Transport Recommendations

This document describes the E-mail transport recommendations that should be used for transporting FSML documents via E-mail. These specifications are not required, but merely recommended.

These specifications are for automatically processed E-mail messages — e.g., FSML Documents (echecks, Deposits, E-mail Registration messages).

Human interaction E-mail messages, such as notices, queries, etc. may deviate from the specifications.

E.1 E-Mail Format

All E-Mail messages being transmitted should adhere to the Internet E-mail specifications in RFC822 (Message Header Format), RFC2045[10], RFC2046[16], and RFC2047[25][26] (MIME).

All Messages will consist of single-part MIME messages (no multi-part MIME attachments should be used), and thus will contain the following headers (in addition to the required RFC822 headers)...

```
Mime-Version: 1.0
Content-Type: application/x-cccccc
Content-Transfer-Encoding: yyyyyy
```

The *cccccc* and *yyyyyy* in the above headers will contain various values.

The *yyyyy* value may be either **7bit** for E-Mail messages that contain normal, readable ASCII text, or **base64** for binary information (i.e., Encrypted data).

The *cccccc* value is discussed below.

If the Content-Type header contains `application/x-fsml` or `application/x-fsml-`*xxxx* then the E-mail message is considered to be an FSML Document which can be processed automatically by FSML processing software. Any other Content-Type will be considered “normal” E-mail, and will be handled accordingly. The *”xxxx”* suffix is used to specify mail format information, such as an encryption algorithm.

E.2 E-Mail Acknowledgments

In order to provide feedback to the Bank Customers that their E-mailed echecks have not been lost during transport over the Internet, the Bank Servers will issue acknowledgment E-mail messages whenever an appropriate FSML document is received by the Bank Servers.

These acknowledgment messages are intended solely to indicate to the customer that the E-mail that they sent has arrived at the bank. The acknowledgment messages do not indicate that the E-mail being acknowledged has passed thru any processing steps, or been verified as being correct syntactically, semantically, or cryptographically.

The acknowledgment will consist of an E-mail message being returned to the sender, with a subject header consisting of `Re:` followed by the contents of the subject header from the E-mail message being acknowledged.

The body of the acknowledgment message may contain any fixed language or message desired by the bank, as long as it does not contain any variable information, or require privacy encryption.

The acknowledgment message will be sent to the originator, using the rules in Internet RFC822 to determine the originator (from among the Reply-to, From, and Sender headers).

E.3 E-Mail Encryption

The privacy encryption technique will be specified by appending the characters `-enc1` (for encryption method 1) or other values, to the `Content-Type` E-mail header. Thus, for such encrypted messages, using encryption method 1, the full MIME header set will read...

```
Mime-Version: 1.0
Content-Type: application/x-fsml-enc1
Content-Transfer-Encoding: base64
```

Alternatively, a non-encrypted message will contain the headers...

```
Mime-Version: 1.0
Content-Type: application/x-fsml
Content-Transfer-Encoding: 7bit
```

E.4 E-Mail Registration

In order to simplify the process of registering new E-mail users, an automatic registration mechanism will be used.

The format of the E-mail registration message is an FSML document, as follows....

```
<fsml-doc docname="dnamestring" type="x:emailreg">
<action>
<function>register
<reason>process
</action>
<x:emailreg>
<crit>true
<vers>1.0
<publickey type="namestring">valuestring
```

```
<idnumber>valuestring
<bankacct>valuestring
<emailaddr>valuestring
<replaces>valuestring
</x:emailreg>
<sig>
...
</sig>
<cert>
... signer’s certificate goes here
</cert>
<cert>
... bank’s certificate goes here
</cert>
</fsml-doc>
```

The <publickey> tag contains a value that is the base64 encoded Encryption public key, which is to be used when sending encrypted information to the e-mail address specified. The tag attribute `type="name"` is to specify the type of public key being registered. Values of this attribute are determined by the E-mail encryption method used (e.g., pgp, s-mime, etc).

The <emailaddr> tag specifies the new E-mail address being registered. This address is to be considered a valid echeck customer

The tag <idnumber> will contain an optional identification number. The tag <bankacct> will be used when communicating the payees public key to the bank. It contains the Electronic Account Number at the bank.

The <replaces> tag is optional, and is used during re-registration. It contains the original E-mail address from the previous registration. If the tag is present, it indicates that the new registration information is a replacement for the original, If omitted, the new registration is a first-time registration or is an addition to the first one, not a replacement (some Customers may require multiple E-mail addresses and public keys). Note: a deletion function may be required later.

The <x:emailreg> block is signed using the Generic signing function of the Electronic Checkbook. The resulting <sig> and <cert> blocks are added to the document. The bank’s certificate needed to verify the signature in the X509 portion of the signer’s certificate should also be included in the document. The receiving system should verify the signatures before processing the registration.

E.4.1 Verification Requirements for x:emailreg document

The x:emailreg document must have the following blocks and fields...

Verification Table - Document Verification Rules

Document	<Block>/Document	Required	Notes
x:emailreg	<action>	x	Function must be `register`. Reason must be `process`, `test`, `resend`, or `info`. If reason is not `process` or `resend`, don't process document.
	<x:emailreg>	x	
	<signature>	x	User's signature.
	<cert>	x	User's certificate.
	<cert>	x	Bank's certificate.

Verification Table - Block Verification Rules

Block	Field	Type	Notes
action	function	string	Note: <action> block must be the first block in the document.
	reason	string	
x:emailreg	publickey	string	Parameter `type="namestring"` required
	idnumber	string	Optional
	bankacct	integer	Required for emailreg to bank
	emailaddr	string	
	replaces	string	Required for replacement registration

Signature Rules Table

Document	Signer	Sigref	sigtype	Blocks hashed
x:emailreg	user	cert	generic	<action> <x:emailreg>

E.5 Example E-mail Message

```
From: morbius@altair5.watson.ibm.com
Date: Fri, 16 May 1997 09:54:19 -0400
Message-Id: <9705161354.AA22234@altair5.watson.ibm.com>
Mime-Version: 1.0
To: echeck1@watson.ibm.com
Subject:  Sample Encoded E-Mail Message
Content-Type:  application/x-fsml-enc1
Content-Transfer-Encoding:  base64
```

```
SGVsbG8uICBUaGlzIGlzIGEgdGVzdCBtZXNzYWdlLgoKIEplZmYgS3Jhdml0eiAgICAgICAg
ICAgICAgICAgICAgICAiSWYgSSBjb3VsZCB0ZWxsIHRoZSBzdG9yeSBpbiB3b3JkcywKIElC
TSBULkouIFdhdHNvbiBSZXNlYXJjaCBDZW50ZXIgICAgSSB3b3VsZG4ndCBuZWVkIHRvIGx1
ZyBhIGNhbWVyYS4iCiBJbnRlcm5ldDptb3JiaXVzQHdhdHNvbi5pYm0uY29tICAgICAgICAg
ICAtIExld2lzIEhpbmUK
```

Base64 Encoding

The Base64 encoding and decoding process is described elsewhere [10], however a brief description follows...

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first byte, and the eighth bit will be the low-order bit in the first byte, and so on.

Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string. These characters, identified in the table below, are selected so as to be universally representable, and the set excludes characters with particular significance to SMTP (e.g., `"."`, `CR, LF`) and to the encapsulation boundaries defined in this document (e.g.,`"-"`).

Base64 Alphabet							
Val	Encode	Value	Encode	Value	Encode	Value	Encode
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Special processing is performed if fewer than 24 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the `'='` character. Since all base64 input is an integral number of octets, only the following cases can arise:

1. the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no `"="` padding,

2. the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two `"="` padding characters, or
3. the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one `"="` padding character.

Because it is used only for padding at the end of the data, the occurrence of any `'='` characters may be taken as evidence that the end of the data has been reached (without truncation in transit). No such assurance is possible, however, when the number of octets transmitted was a multiple of three.

Any characters outside of the base64 alphabet are to be ignored in base64-encoded data. The same applies to any illegal sequence of characters in the base64 encoding, such as `"====="`.

Acknowledgements

The creation and continued enhancement of this document was greatly assisted by many people, including the following contributors from the FSTC echeck Technical Team:

Jim Akister	RDM
Milt Anderson	FSTC
Sheueling Chang	Sun Microsystems
Greg Dunne	Telequip
Norb Dusyn	Bank of America
Mark Feldman	CommerceNet
Nikki Fischer	Huntington Banks
John Fricke	Chase Manhattan Bank
Michael Halperin	GTE Internetworking
Chris Hibbert	Agorics
Eric Hill	Agorics
Frank Jaffe	BankBoston
David Lant	RDM
An Le	National Semiconductor
Stuart Marks	Sun Microsystems
Cyndi Mills	GTE Internetworking
Elaine Palmer	IBM Research
Craig Rector	IBM
Brian Risman	Bank of Montreal
Robert Rocchetti	Sun Microsystems
Jim Seck	Unisys
Mark Smith	Oak Ridge National Lab
Sean Smith	IBM Research
Tony Smith	IntraNet
Dave Solo	GTE Internetworking
Andrew Sutton	IBM
Kurt Thams	Agorics
Gene Tsudik	USC-ISI
Jennifer Vancini	Certicom
Paridhi Verma	IBM Research
Mike Versace	Federal Reserve Bank
Jyri Virkki	Sun Microsystems
Chuck Wade	GTE Internetworking
Gary Werner	Unisys