\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Standard RBE

Część I: Specyfikacja warstwy komunikacyjnej ebXML

wersja 1.0

Bydgoszcz, marzec 2003

**Spis treści**

[Słownik 6](#_Toc36366770)

[1. Wprowadzenie 9](#_Toc36366771)

[2. Wymiana danych w architekturze ebXML 9](#_Toc36366772)

[2.1. Podstawowe informacje 9](#_Toc36366773)

[2.2. Modele wymiany danych 10](#_Toc36366774)

[2.2.1. Single-hop 11](#_Toc36366775)

[2.2.2. Multi-hop 11](#_Toc36366776)

[2.3. Struktura przesyłki ebXML 12](#_Toc36366777)

[2.4. Funkcjonalność MSH 13](#_Toc36366778)

[2.4.1. Schemat funkcjonalny MSH 13](#_Toc36366779)

[2.4.2. Podstawowa funkcjonalność MSH 15](#_Toc36366780)

[2.4.3. Opcjonalna funkcjonalność MSH 16](#_Toc36366781)

[2.4.4. Minimalne wymagania na implementację MSH zgodną z ebXML 16](#_Toc36366782)

[2.5. CPA – umowa o współpracy partnerów 17](#_Toc36366783)

[3. Wymiana danych zgodna z *reliable messaging* 17](#_Toc36366784)

[*3.1.* *Persistent storage* 18](#_Toc36366785)

[3.2. Alternatywne sposoby implementacji protokołu gwarantującego niezawodność dostarczania przesyłek 19](#_Toc36366786)

[3.3. Zasady wymiany danych w protokole *ebXML reliable messaging* 19](#_Toc36366787)

[3.3.1. Generowanie potwierdzeń 20](#_Toc36366788)

[3.3.1.1. Komunikat potwierdzający 21](#_Toc36366789)

[3.3.1.2. Wiadomość potwierdzająca 22](#_Toc36366790)

[3.3.2. Wykrywanie i obsługa duplikatów 22](#_Toc36366791)

[3.4. Szczegółowy opis zasad wymiany danych w protokole *ebXML reliable messaging* 24](#_Toc36366792)

[3.4.1. Wysyłanie przesyłki 24](#_Toc36366793)

[3.4.2. Odbiór przesyłki 25](#_Toc36366794)

[3.4.3. Ponowne przesłanie przesyłki 30](#_Toc36366795)

[3.4.4. Ponowne przesłanie potwierdzenia 31](#_Toc36366796)

[3.4.5. Postępowanie w przypadku niepowodzenia dostarczenia przesyłki do Adresata 32](#_Toc36366797)

[3.5. Tryby przekazywania przesyłek w *reliable messaging* 33](#_Toc36366798)

[3.6. Parametry protokołu *reliable messaging* 34](#_Toc36366799)

[3.6.1. DuplicateElimination 34](#_Toc36366800)

[3.6.2. AckRequested 35](#_Toc36366801)

[3.6.3. Retries 35](#_Toc36366802)

[3.6.4. RetryInterval 36](#_Toc36366803)

[3.6.5. TimeToLive 36](#_Toc36366804)

[3.6.6. PersistDuration 36](#_Toc36366805)

[3.6.7. syncReplyMode 37](#_Toc36366806)

[4. Dodatkowe moduły funkcjonalne MSH 38](#_Toc36366807)

[4.1. Moduł obsługi błędów 38](#_Toc36366808)

[4.1.1. Definicje 38](#_Toc36366809)

[4.1.2. Typy błędów 38](#_Toc36366810)

[4.1.3. Implementacja obsługi błędów 39](#_Toc36366811)

[4.1.3.1. Zasady generowania komunikatów o błędach 39](#_Toc36366812)

[4.1.3.2. Ustalenie *Error Reporting Location* 39](#_Toc36366813)

[4.1.3.3. Przekazywanie informacji o błędzie 40](#_Toc36366814)

[4.1.4. Struktura Komunikatu o błędzie 40](#_Toc36366815)

[4.2. Moduł obsługi sekwencji przesyłek 41](#_Toc36366816)

[4.3. Moduł obsługi Komunikatów diagnostycznych 42](#_Toc36366817)

[4.3.1. Komunikat diagnostyczny MSH Ping Message 42](#_Toc36366818)

[4.3.2. Komunikat diagnostyczny MSH Pong Message 43](#_Toc36366819)

[4.4. Moduł obsługi Komunikatów statusowych 44](#_Toc36366820)

[4.4.1. Komunikat Message Status Request 45](#_Toc36366821)

[4.4.2. Komunikat Message Status Response 46](#_Toc36366822)

[5. Warstwa zabezpieczeń w ebXML 48](#_Toc36366823)

[5.1. Wprowadzenie 48](#_Toc36366824)

[5.2. Parametry CPA 48](#_Toc36366825)

[5.3. Zabezpieczenia na poziomie warstwy transportowej 48](#_Toc36366826)

[5.4. Szyfrowanie w przesyłce ebXML 49](#_Toc36366827)

[5.4.1. Szyfrowanie elementów nagłówka koperty SOAP/ebXML 49](#_Toc36366828)

[5.4.2. Szyfrowanie dokumentów biznesowych 49](#_Toc36366829)

[5.5. Podpis elektroniczny w przesyłce ebXML 49](#_Toc36366830)

[5.5.1. XML Signature 49](#_Toc36366831)

[5.5.2. XML Signature w nagłówku SOAP:Header 50](#_Toc36366832)

[5.5.3. Podpis w Komunikacie potwierdzającym 51](#_Toc36366833)

[5.5.4. Podpis w modelu wymiany Multi-hop 52](#_Toc36366834)

[5.6. Bezpieczny podpis elektroniczny w przesyłce ebXML 52](#_Toc36366835)

[5.6.1. XML Advanced Electronic Signature (XAdES) 52](#_Toc36366836)

[6. Specyfikacja elementów koperty ebXML 55](#_Toc36366837)

[6.1. Informacje wprowadzające 55](#_Toc36366838)

[6.1.1. Stosowane w dokumencie przestrzenie nazw: 55](#_Toc36366839)

[6.1.2. Sposób przygotowania definicji elementów 55](#_Toc36366840)

[6.2. Struktura koperty ebXML 57](#_Toc36366841)

[6.3. Atrybuty o zasięgu publicznym 59](#_Toc36366842)

[6.3.1. Atrybut    id 59](#_Toc36366843)

[6.3.2. Atrybut version 59](#_Toc36366844)

[6.3.3. SOAP:mustUnderstand 59](#_Toc36366845)

[6.3.4. SOAP:actor 60](#_Toc36366846)

[6.3.5. Grupy atrybutów 60](#_Toc36366847)

[6.3.5.1. Grupa atrybutów "Header" 61](#_Toc36366848)

[6.3.5.2. Grupa atrybutów „Body*”* 61](#_Toc36366849)

[6.4. Elementy występujące w SOAP:Header 61](#_Toc36366850)

[6.4.1. MessageHeader 62](#_Toc36366851)

[6.4.2. From / To 65](#_Toc36366852)

[6.4.3. MessageData 66](#_Toc36366853)

[6.4.4. SyncReply 67](#_Toc36366854)

[6.4.5. MessageOrder 67](#_Toc36366855)

[6.4.6. AckRequested 69](#_Toc36366856)

[6.4.7. Acknowledgment 71](#_Toc36366857)

[6.4.8. ErrorList 72](#_Toc36366858)

[6.4.9. Error 73](#_Toc36366859)

[6.4.10. ds:Signature 75](#_Toc36366860)

[6.5. Elementy występujące w SOAP:Body 76](#_Toc36366861)

[6.5.1. Manifest 77](#_Toc36366862)

[6.5.2. Reference 77](#_Toc36366863)

[6.5.3. StatusRequest 78](#_Toc36366864)

[6.5.4. StatusResponse 79](#_Toc36366865)

[6.6. Elementy występujące w ds:Signature 80](#_Toc36366866)

[6.6.1. ds:SignedInfo 80](#_Toc36366867)

[6.6.2. ds:SignatureMethod 81](#_Toc36366868)

[6.6.3. ds:Reference 82](#_Toc36366869)

[6.6.4. ds:Transform 83](#_Toc36366870)

[6.6.5. ds:KeyInfo 84](#_Toc36366871)

[6.6.6. ds:KeyValue 85](#_Toc36366872)

[6.6.7. ds:DSAKeyValue 86](#_Toc36366873)

[6.6.8. ds:RSAKeyValue 86](#_Toc36366874)

[6.6.9. ds:RetrievalMethod 87](#_Toc36366875)

[6.6.10. ds:X509Data 87](#_Toc36366876)

[6.6.11. ds:X509IssuerSerial 88](#_Toc36366877)

[6.6.12. ds:PGPData 89](#_Toc36366878)

[6.6.13. ds:SPKIData 89](#_Toc36366879)

[6.6.14. ds:Object 89](#_Toc36366880)

[6.6.15. ds:Manifest 90](#_Toc36366881)

[6.6.16. ds:SignatureProperties 90](#_Toc36366882)

[7. Zasady działania i funkcjonalność aplikacji Standardu RBE 92](#_Toc36366883)

[7.1. Diagram przepływu danych pomiędzy Aplikacjami biznesowymi partnerów za pośrednictwem MSH 92](#_Toc36366884)

[7.2. Model obsługi podpisu 93](#_Toc36366885)

[7.2.1. Generowanie sygnatury XML 93](#_Toc36366886)

[7.2.2. Weryfikacja sygnatury XML 95](#_Toc36366887)

[7.3. Wymiana przesyłek ebXML przez HTTP 97](#_Toc36366888)

[7.3.1. Wysyłanie przesyłek ebXML przez HTTP 97](#_Toc36366889)

[7.3.2. Odbiór przesyłek ebXML przez HTTP 99](#_Toc36366890)

[7.3.3. Kody statusów odpowiedzi HTTP w obsłudze błędów SOAP/ebXML 100](#_Toc36366891)

[7.3.4. Kontrola dostępu na poziomie HTTP 101](#_Toc36366892)

[7.3.5. Poufność na poziomie protokołu komunikacyjnego 101](#_Toc36366893)

[7.3.6. Wymiana przesyłek ebXML przez SMTP 101](#_Toc36366894)

[7.3.7. Wysyłanie przesyłek ebXML przez SMTP 102](#_Toc36366895)

[7.3.8. Pocztowy Komunikat odbioru 104](#_Toc36366896)

[7.3.9. Kontrola dostępu 104](#_Toc36366897)

[7.3.10. Poufność na poziomie protokołu komunikacyjnego 104](#_Toc36366898)

[7.4. Parametry do zdefiniowania w CPA 104](#_Toc36366899)

[Bibliografia 105](#_Toc36366900)

# Słownik

|  |  |  |
| --- | --- | --- |
| Acknowledgement Message | – | *Komunikat potwierdzający* otrzymanie przesyłki przekazany przez *ReceivingMSH*. |
| ASN.1  *(Abstract Syntax Notation One)* | – | Formalny język abstrakcyjnego opisu komunikatów wymienianych pomiędzy systemami informatycznymi.  <http://www.oss.com/asn1/> |
| Atrybut XML | – | Nazwany parametr elementu XML. |
| Autoryzacja | – | Nadane uprawnienia, w tym zezwolenie na dostęp do konkretnych funkcji lub zasobów. |
| CA  *(Certification Authority)* | – | Podmiot świadczący usługi wydawania, zawieszania i unieważniania certyfikatów |
| Certyfikat | – | Sekwencja danych zawierająca: informacje pozwalające na identyfikację posiadacza, klucz publiczny posiadacza, numer seryjny, okres ważności. Certyfikat może być podpisany przez CA. |
| Dowolny atrybut | – | Niezdefiniowany w XML Schema atrybut stanowiący zawartość elementów rozszerzalnych. |
| Dowolny element | – | Niezdefiniowany w XML Schema element stanowiący zawartość elementów rozszerzalnych. |
| CPA  *(Colaboration Partner Agreement)* | – | Umowa o współpracy pomiędzy dwoma MSH, określająca parametry wymiany i obsługi przesyłek. |
| DAS  *(Digital Signature Algorithm)* | – | System kryptografii klucza publicznego przeznaczony specjalnie do tworzenia podpisów cyfrowych. |
| EbXML MSS | – | skrót od ebXML Message Service Specification |
| Element | – | Węzeł struktury (drzewa) XML posiadający wartość tekstową oraz opcjonalną sekwencję atrybutów. |
| Element główny | – | Pierwszy, główny węzeł struktury (drzewa) XML obejmujący pozostałe elementy XML. |
| Element potomny | – | Dowolny element XML stanowiący zawartość elementu złożonego. |
| Element pusty | – | Węzeł struktury (drzewa) XML nie posiadający żadnych elementów potomnych, atrybutów ani wartości tekstowej. |
| Element rozszerzalny | – | Węzeł struktury XML posiadający otwartą definicje zawartości w XML Schema. |
| Element złożony | – | Węzeł struktury (drzewa) XML zawierający co najmniej jeden element XML. |
| Encja XML | – | Odwołanie do dowolnego zasobu w dokumencie XML. |
| FromPartyMSH | – | Identyfikator MSH źródła przesyłki ebXML. |
| Funkcja skrótu | – | Algorytm, który pozwala na jednoznaczne i nieodwracalne odwzorowanie znacznej ilości danych na krótką sekwencję znaków. |
| HMAC  *(Hashing for Message Authentication Codes)* | – | Algorytm podpisywania danych w oparciu o funkcję skrótu oraz współdzielony klucz sesyjny.  <http://www.ietf.org/rfc/rfc2104.txt> |
| HTTP  *(Hypertext Transfer Protocol)* | – | Protokół wymiany danych.  <http://www.ietf.org/rfc/rfc2616.txt> |
| Implementacja MSH | – | Oprogramowanie realizujące co najmniej podstawową funkcjonalność MSH. |
| IntermediateMSH | – | Identyfikator pośredniczącego MSH w modelu wymiany Multi-hop. |
| IPSEC  *(IP Security Protocol)* | – | Protokół sieciowy zapewniający poufność danych, autoryzacje i uwierzytelnienie, stosowany w rozwiązaniach VPN (*Virtual Privte Network*).  <http://www.ietf.org/html.charters/ipsec-charter.html> |
| Komunikat o błędzie | – | Przesyłka ebXML zawierająca kopertę ebXML z elementem <*ErrorList*>. |
| Komunikat potwierdzający | – | Przesyłka ebXML zawierająca kopertę ebXML z elementem <*Acknowledgment*>. |
| Koperta SOAP / ebXML | – | Dokument XML zawierający elementy SOAP i ebXML, pierwsza część struktury MIMIE/Multipart. |
| Message Service Handler (MSH) | – | Logiczny podsystem w infrastrukturze informatycznej strony wymieniającej dane w standardzie ebXML, odpowiedzialny za wysyłanie i odbieranie przesyłek ebXML. |
| Message Status Request | – | *Komunikat diagnostyczny ebXML* zawierający zapytanie o status wcześniej przesłanej przesyłki ebXML. |
| Message Status Response | – | *Komunikat diagnostyczny ebXML* z odpowiedzią zawierającą status wcześniej przesłanej przesyłki ebXML. |
| MIME  *(Multipurpose Internet Mail Extensions)* | – | Standard formatowania komunikatów pocztowych.  <http://www.ietf.org/rfc/rfc2045.txt> |
| Multi-hop | – | Model wymiany przesyłek ebXML z pośredniczącym MSH. |
| NextMSH | – | Identyfikator określający MSH odbiorcy przesyłki ebXML. |
| Open PGP | – | Standard implementacji usług kryptograficznych (podpis elektroniczny, szyfrowanie, zarządzanie kluczami i certyfikatami) bazujących na PGP.  <http://www.ietf.org/rfc/rfc2440.txt> |
| Parsowanie XML | – | Proces analizy syntaktycznej (składniowej) dokumentu XML obejmujący sprawdzenie poprawności struktury i udostępnienie zawartości w interfejsie DOM, SAX. |
| PersistDuration | – | Parametr wymiany danych definiowany w CPA, określa czas przechowywania komunikatu w *persistent storage*. |
| Persistent storage | – | Moduł MSH umożliwiający przechowywanie wychodzących i przychodzących przesyłek ebXML w ramach obsługi *reliable messaging.* |
| PGP  *(Pretty Good Privacy)* | – | System kryptograficzny autorstwa Phila Zimmermanna wykorzystujący ideę klucza publicznego. |
| PKI  *(Public Key Infrastructure)* | – | Architektura, organizacja, techniki, zasady i procedury, które wspólnie umożliwiają stworzenie i funkcjonowanie systemu kryptograficznego klucza publicznego opartego na certyfikatach. |
| Przesyłka ebXML | – | Zbiór danych przesyłanych pomiędzy aplikacjami nadawcy i odbiorcy w formacie zgodnym ze specyfikacją *ebXML Message Service Specification.* |
| ReceivingMSH | – | Identyfikator MSH odbiorcy przesyłki ebXML. |
| Reliable messaging | – | Zasady wymiany przesyłek ebXML pomiędzy komunikującymi się MSH (wiarygodna, pewna komunikacja, niezawodne dostarczenie). |
| RSA | – | System kryptografii klucza publicznego opracowany przez Rivesta, Shamira i Adlemana. |
| Retries | – | Parametr wymiany danych definiowany w CPA, określa maksymalną liczbę kolejnych powtórzeń wysłania przesyłek ebXML, którego odbiór nie został potwierdzony. |
| RetryInterval | – | Parametr wymiany danych definiowany w CPA, określa maksymalny przedział czasu pomiędzy powtórnie wysłanymi przesyłkami ebXML. |
| SendingMSH | – | Identyfikator MSH nadawcy przesyłki ebXML. |
| SHA-1 | – | Algorytm funkcji skrótu. |
| S/MIME  *(Secure Multipurpose Internet Mail Extensions)* | – | Standard zabezpieczania sformatowanych komunikatów pocztowych.  <http://www.ietf.org/rfc/rfc2206.txt> |
| SOAP  *(Simple object Access Protocol)* | – | Protokół wymiany danych w standardzie XML.  <http://www.w3.org/TR/SOAP> |
| SPKI  *(Simple Public Key Infrastructure)* | – | Specyfikacja opracowana przez IETF, obejmująca formaty i protokoły stosowane w ramach uproszczonej infrastruktury klucza publicznego.  <http://www.ietf.org/html.charters/spki-charter.html> |
| SSL  *(Secure Socket Layer)* | – | Protokół warstwy transportowej sieci, zapewniający poufność danych autoryzacje i uwierzytelnienie.  <http://wp.netscape.com/eng/ssl3/draft302.txt> |
| SyncReplyMode | – | Parametr CPA określający rodzaj przesyłek ebXML przekazywanych w synchronicznej wymianie danych. |
| TimeToLive | – | Element XML w strukturze koperty ebXML określający maksymalny czas przesłania przesyłki ebXML. |
| TLS  *(Transport Layer Security)* | – | Protokół warstwy transportowej sieci, zapewniający autoryzacje i poufność danych, wstecznie kompatybilny z SSL.  <http://www.ietf.org/rfc/rfc2246.txt> |
| ToPartyMSH | – | Identyfikator MSH końcowego odbiorcy przesyłki ebXML. |
| URI  *(Uniform Resource Identifiers)* | – | Standard formatowania identyfikatorów zasobów.  <http://www.ietf.org/rfc/rfc2396.txt> |
| UTC  *(Universal Coordinated Time)* | – | Czas uniwersalny, który minął od północy (00:00) 1 stycznia 1970 roku czasu GMT. |
| UTF-8 | – | Międzynarodowy standard kodowanie znaków.  <http://www.ietf.org/rfc/rfc2279.txt> |
| Uwierzytelnienie  *(Authentication)* | – | Proces potwierdzenia tożsamości podmiotu lub integralności danych. Uwierzytelnienie wiadomości obejmuje ustalenie jej nadawcy i sprawdzenie, czy wiadomość nie została zmodyfikowana lub zastąpiona inną w czasie przesyłania. |
| Walidacja XML | – | Proces sprawdzenia zgodności struktury i zawartości dokumentu XML z przypisaną XML Schema lub DTD. |
| X509v3 | – | Standard opisujący strukturę certyfikatu. |
| XML  *(Extensible Markup Language)* | – | Rozszerzalny język znaczników. |
| XML Schema | – | Język opisu struktury i zawartości dokumentu XML.  <http://www.w3.org/XML/Schema> |
| Xpath | – | Język wyrażeń do wyszukiwania i wskazywania elementów (węzłów) w dokumentach XML.  <http://www.w3.org/XML/xpath> |
| Xlink | – | Hipertekstowe, zewnętrzne odsyłacze łączące wiele zasobów z dokumentem XML.  <http://www.w3.org/XML/Linking> |

# Wprowadzenie

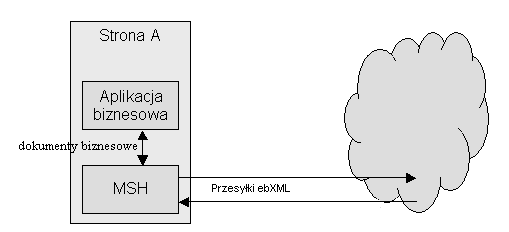
Dokument ten, nazywany dalej Specyfikacją, stanowi pierwszą część dokumentacji Standardu RBE, w której jest opisana warstwa komunikacyjna tego standardu. Opracowany został w oparciu o specyfikację *The ebXML Message Service Specification v. 2.0* (w dalszej części określanej w skrócie jako ebXML MSS).

1. W Specyfikacji, przedstawione są: ogólny model wymiany danych w architekturze ebXML,
2. komponenty systemu informatycznego strony zaangażowanej w wymianę danych,
3. struktura przesyłek ebXML, które są przesyłane pomiędzy współpracującymi ze sobą stronami,
4. zasady obsługi przesyłek i wymiany danych pomiędzy stronami, w tym:
   1. kontrole składni i zawartości merytorycznej przesyłek,
   2. sposoby raportowania o wykrytych błędach,
   3. zasady niezawodnego dostarczania przesyłek ebXML do adresata (*reliable messaging)*,
   4. rozwiązania przyjęte w ebXML związane z ochroną danych wymienianych pomiędzy partnerami, w tym podpis elektroniczny przekazywany w przesyłce ebXML.

# Wymiana danych w architekturze ebXML

## Podstawowe informacje

Strony, współpracujące ze sobą w architekturze ebXML, są wyposażone w systemy o strukturze pokazanej na poniższym rysunku.



Podstawowe komponenty systemu to:

1. *Aplikacja biznesowa* - oprogramowanie, które obsługuje dokumenty biznesowe i wykonuje funkcje związane z przekazaniem ich do i odbiorem z MSH,
2. MSH (*Message Service Handler*) - oprogramowanie realizujące następujące funkcje:
   1. przygotowywanie przesyłek ebXML zawierających przekazane z *Aplikacji biznesowej* dokumenty biznesowe,
   2. przesyłanie przesyłek ebXML do MSH partnerów,
   3. odbiór przesyłek ebXML przesłanych przez MSH partnerów.

W dalszej części Specyfikacji zostanie szczegółowo opisana funkcjonalność MSH. Opis *Aplikacji biznesowej* jest poza zakresem tego dokumentu i ogranicza się tylko do interakcji tej aplikacji z MSH.

W dalszych rozdziałach Specyfikacji w opisach wymiany danych stosowane są następujące oznaczenia:

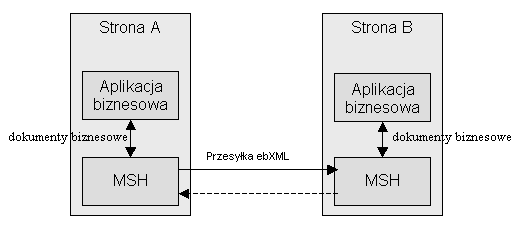
1. partnerzy biznesowi wymieniający ze sobą dane określani są jako:
   1. *FromParty* (nadawca przesyłki zawierającej dokument biznesowy),
   2. *ToParty –* (odbiorca przesyłki zawierającej dokument biznesowy),
2. elementy Systemu każdego z partnerów określane są odpowiednio:
   1. *Aplikacja biznesowa FromParty*,
   2. *Aplikacja biznesowa ToParty*,
   3. *FromPartyMSH*,
   4. ToPartyMSH,
3. elementy pośredniczące w wymianie pomiędzy partnerami (patrz w kolejnym pkt. opis *Multi-hop*) określane są jako *IntermediateMSH,*
4. role realizowane przez MSH w ramach danej wymiany są określane jako:
   1. *ReceivingMSH* (otrzymujące MSH),
   2. *SendingMSH* (wysyłające MSH),
   3. *NextMSH (*kolejne MSH w wymianie *Multi-hop*).

## Modele wymiany danych

Specyfikacja ebXML MSS określa zasady współpracy partnerów w zakresie wymiany danych, która może być realizowana w modelu:

1. *Single-hop*, w ramach którego przesyłki są przekazywane bezpośrednio pomiędzy partnerami biznesowymi,
2. *Multi-hop,* w którym w wymianę przesyłek są zaangażowane strony pośredniczące.

### Single-hop



Na rysunku powyżej jest przedstawiony model wymiany, w którym dane są przekazywane bezpośrednio pomiędzy partnerami biznesowymi. W modelu tym:

1. MSH nadawcy przesyłki (na rysunku jest nim MSH Strony A) pełni rolę *FromPartyMSH* i *SendingMSH*,
2. MSH odbiorcy przesyłki (na rysunku jest nim MSH Strony B) pełni rolę *ToPartyMSH* i *ReceivingMSH.*

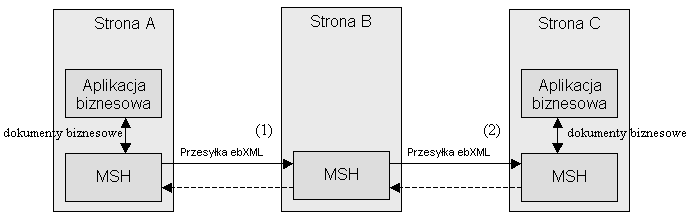
W architekturze ebXML każdy z partnerów biznesowych może być nadawcą i odbiorcą przesyłek.

### Multi-hop

W modelu *Multi-hop* w wymianę przesyłek, oprócz współpracujących ze sobą partnerów biznesowych, są zaangażowane strony pośredniczące, określane jako *IntermediateMSH*.

Model ten pozwala na wymianę danych pomiędzy stronami, które z pewnych względów nie mogą przekazywać przesyłek bezpośrednio np. w przypadku gdy obsługują inne protokoły transportowe. Ponadto w pewnych rozwiązaniach, nie objętych zakresem Specyfikacji, pośrednicy mogą pełnić rolę zaufanej trzeciej strony (np. świadczyć usługę znakowania czasem). Rola pośredników w modelu *Multi-hop* ogranicza się do przekazania przesyłki do kolejnego MSH (*NextMSH*) bez odnoszenia się do dokumentów biznesowych przekazywanych w przesyłce.

Dodatkowe funkcje, które mogą być realizowane przez pośredniczące MSH, związane np. z przekazywaniem komunikatów serwisowych, są opisane w kolejnych rozdziałach Specyfikacji.



Na powyższym diagramie:

* 1. MSH Strony A jest *FromPartyMSH* i pełni rolę *SendingMSH*,
  2. MSH Strony B jest *IntermediateMSH* i pełni rolę:
     1. *ReceivingMSH* i *NextMSH* w przepływie *(1)*,
     2. *SendingMSH* w przepływie *(2)*,
  3. MSH Strony C, który jest *ToPartyMSH*, pełni rolę *ReceivingMSH* i *NextMSH* w przepływie *(2)*

## Struktura przesyłki ebXML

Ogólna budowa przesyłki ebXML przedstawiona została na poniższym rysunku.

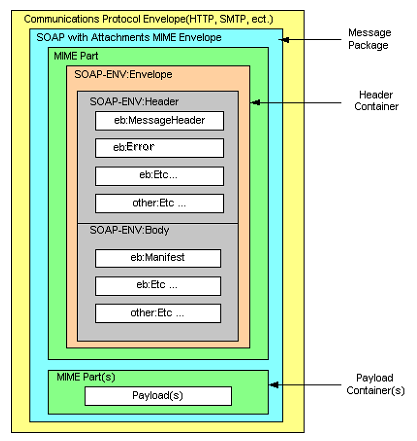
Struktura przesyłki ebXML jest zgodna ze specyfikacją *SOAP Messages with Attachments*.

Przesyłka ebXML ma format MIME/multipart i składa się z dwóch części MIME:

1. pierwszej, stanowiącej część nagłówkową przesyłki, określanej w Specyfikacji jako koperta ebXML,
2. drugiej, zawierającej części określane jako *payload*-y, w których są przekazywane dokumenty z *Aplikacji biznesowej* (zero lub więcej).

Podstawowym elementem koperty ebXML jest *<SOAP:Envelope>*, który w swoich podelementach *<SOAP:Header>* i *<SOAP:Body>* zawiera elementy rozszerzeń ebXML-owych zdefiniowanych w Specyfikacji (na rysunku pokazane są tylko podstawowe elementy tych rozszerzeń: *<MessageHeader>* i *<Manifest>*). Struktura elementów rozszerzeń ebXML-owych jest szczegółowo opisana w pkt *Specyfikacja elementów koperty ebXML.* W elementach tych przekazywane są dane:

1. identyfikujące przesyłkę, nadawcę, odbiorcę i załączone dokumenty,
2. określające parametry i wymagania dotyczące przetworzenia przesyłki np. żądanie dotyczące przesłania potwierdzenia otrzymania przesyłki, żądanie wykrywania duplikatów,
3. identyfikujące funkcję biznesową realizowaną przez dokument biznesowy przekazywany w przesyłce (informacje umożliwiające routing dokumentu do odpowiedniej aplikacji),
4. opcjonalnie zawierające podpis elektroniczny, zgodny ze specyfikacją W3C XML Signature.

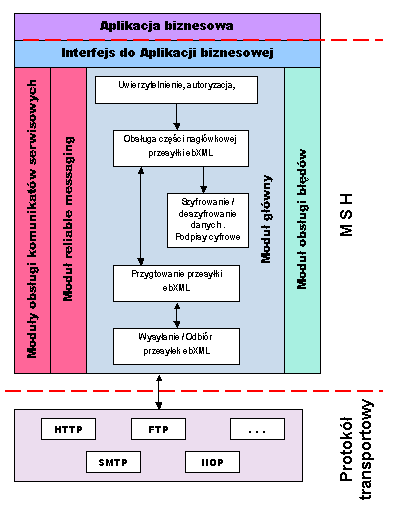


## Funkcjonalność MSH

### Schemat funkcjonalny MSH

Na poniższym rysunku pokazane są moduły funkcjonalne MSH, realizujące określone akcje związane z przygotowaniem i wymianą danych ebXML.

Schemat przedstawia zależności pomiędzy modułami i ich powiązania z zewnętrznymi modułami odpowiedzialnymi za obsługę dokumentów biznesowych lub obsługującymi protokoły transportowe (np. HTTP lub SMTP).



Na schemacie, w *Module głównym*, przedstawione są podstawowe funkcje realizowane przez MSH w ramach przygotowania lub odbioru przesyłek ebXML. Funkcje te obejmują:

1. obsługę części nagłówkowej przesyłki (koperty ebXML):
   1. w przypadku nowych przesyłek tworzenie ebXML-owych rozszerzeń koperty SOAP na bazie:
      1. danych biznesowych otrzymanych za pośrednictwem *Interfejsu do Aplikacji biznesowej*,
      2. parametrów zdefiniowanych w CPA,
      3. informacji generowanych przez moduł główny MSH takich jak np. podpis cyfrowy, data i czas przesłania czy unikalne identyfikatory,
   2. w przypadku odebranych przesyłek parsowanie części nagłówkowej przesyłki (koperty ebXML) polegające na pobieraniu i transformacji danych przekazanych w kopercie do postaci wymaganej przez daną implementację MSH,
2. usługi ochrony danych:
   1. uwierzytelnienie i autoryzacja stron wymieniających przesyłki (podpis cyfrowy może stanowić element koperty ebXML),
   2. szyfrowanie przekazywanych danych,
3. przygotowywanie przesyłek ebXML (umieszczenie przygotowanej koperty ebXML i dokumentów biznesowych w strukturze *SOAP Messages with Attachments*)*,*
4. wysyłanie/odbiór przesyłek (współpraca z protokołem transportowym).

Oprócz *Modułu* głównego MSH jest wyposażone w następujące moduły:

1. moduł *Interfejs do Aplikacji biznesowej;*

zakres funkcji realizowanych przez ten moduł obejmuje m.in. pobieranie dokumentów do wysłania, przekazywanie dokumentów otrzymanych, informowanie o niepowodzeniu dostarczenia dokumentów do adresata (*ToParty*),

1. *Moduł obsługi błędów*,
2. *Moduł reliable messaging* (realizujący m.in. funkcje ponownego wysyłania przesyłek ebXML i wysyłania potwierdzeń odbioru),
3. *Moduł obsługi komunikatów serwisowych* (takich jak np. *Komunikaty statusowe* lub *Komunikaty diagnostyczne*),
4. *Moduł obsługi sekwencji przesyłek.*

W Specyfikacji zdefiniowane są wszystkie funkcje, jakie MSH może realizować w ramach obsługi wymiany przesyłek w architekturze ebXML. Nie wszystkie z opisanych funkcji składają się na podstawową funkcjonalność, jaką musi realizować każdy system MSH. Funkcje zaliczone w Specyfikacji do opcjonalnych nie będą musiały być implementowane w MSH. W przypadku, gdy grupa partnerów handlowych nie wymaga niezawodności dostarczania przesyłek, implementacja w ich MSH *Modułu reliable messaging* jest zbędna.

### Podstawowa funkcjonalność MSH

Podstawowa funkcjonalność, wymagana w każdej implementacji MSH, obejmuje:

1. przygotowywanie przesyłki ebXML zgodnie z zasadami określającymi sposób budowania przesyłki (m.in. sposób umieszczania dokumentów biznesowych w strukturze zgodnej ze specyfikacją *SOAP with Attachments*, patrz opis *Struktura przesyłki ebXML*),
2. obsługę informacji przekazywanych w ebXML-owych rozszerzeniach koperty SOAP wymaganych w przesyłkach ebXML (patrz pkt *Specyfikacja elementów koperty* → *Struktura koperty ebXML* w którym podane są elementy, których obsługa jest wymagana w ramach podstawowej funkcjonalności MSH),
3. obsługę błędów (patrz pkt *Moduł obsługi błędów*),
4. obsługę podpisów cyfrowych zgodnych z XML *Signature* (patrz pkt *Podpis elektroniczny w przesyłce ebXML*),
5. obsługę żądania synchronicznej odpowiedzi (element *SyncReply*).

### Opcjonalna funkcjonalność MSH

Zgodnie z ebXML nie wszystkie funkcje opisane w Specyfikacji muszą być obsługiwane przez każde MSH. Obowiązuje jednak wymóg, aby MSH potrafiło zareagować na żądanie, wykraczające poza zaimplementowaną funkcjonalność, przekazując w odpowiedzi *Komunikat o błędzie* informujący o braku obsługi danego żądania ("*NotSupported*").

Opcjonalna funkcjonalność MSH obejmuje obsługę:

1. protokołu *reliable messaging* niezawodnego dostarczania przesyłek, (patrz pkt *Wymiana danych zgodna z reliable messaging),*
2. *Komunikatów statusowych*, pozwalających sprawdzić status przesyłki u odbiorcy (*ReceivingMSH)* (patrz pkt *Moduł obsługi komunikatów statusowych*),
3. *Komunikatów diagnostycznych* (patrz pkt *Moduł obsługi komunikatów diagnostycznych*),
4. sekwencji przesyłek *(patrz pkt Moduł obsługi sekwencji przesyłek)*,
5. przesyłek w modelu *Multi-hop*.

*Uwaga:*

*Implementacja w MSH wymienionej powyżej opcjonalnej funkcjonalności wiąże się z koniecznością obsługi dodatkowych, opcjonalnych elementów koperty ebXML.*

*W punkcie "Struktura koperty ebXML" przedstawiającym strukturę przesyłki ebXML wskazane zostały elementy, których obsługa jest wymagana przez poszczególne funkcje dodatkowe.*

### Minimalne wymagania na implementację MSH zgodną z ebXML

Implementacja MSH, która ma być zgodna z ebXML MSS, powinna spełniać następujące warunki:

1. obsługiwać wszystkie zalecenia wymienione w opisach elementów i funkcji składających się na podstawową funkcjonalność MSH *(patrz Specyfikacja elementów koperty ebXML);* zalecenia te zawierają w opisach słowa kluczowe: musi, nie musi (nie wolno), wymagane, powinien, nie powinien,
2. spełniać wymagania co do składni i zasad działania dla wszystkich zaimplementowanych przez MSH funkcji dodatkowych *(patrz pkt Opcjonalna funkcjonalność MSH),*
3. stosować następującą interpretację słów kluczowych *opcjonalny* oraz może; jeżeli słowa te dotyczą elementów przesyłki odnoszących się do modułów zaimplementowanych w MSH, poprawna implementacja tych modułów musi potrafić przetworzyć opcjonalne zawartości przesyłek zgodnie z opisaną semantyką ebXML,
4. w przypadku zaimplementowania opcjonalnej, dodatkowej funkcjonalności zdefiniowanej w Specyfikacji należy dodatkowo umożliwić współdziałanie z innymi MSH, w których ta funkcjonalność nie została zaimplementowana; współdziałanie to powinno dotyczyć obsługi błędów raportujących brak żądanej funkcjonalności w MSH partnera,
5. reagować na żądania wykraczające poza zaimplementowaną funkcjonalność przekazując w odpowiedzi właściwy *Komunikat o błędzie* raportujący brak obsługi danego żądania.

## CPA – umowa o współpracy partnerów

CPA (*Collaboration Protocol Agreement*) jest dokumentem, w którym są zapisywane ustalenia partnerów dotyczące zakresu i sposobu wzajemnej współpracy określające m.in.:

1. listę przekazywanych dokumentów biznesowych,
2. wymogi i możliwości w zakresie dodatkowej funkcjonalności MSH,
3. stosowane zabezpieczenia,
4. parametry wymiany przesyłek,
5. możliwe sposoby transportu przesyłek ebXML.

Zakres parametrów, które powinny się znaleźć w CPA, jest podany w pkt. *Parametry do zdefiniowania w CPA*.

Zakłada się, że docelowo w architekturze ebXML uzgodnienia dotyczące warunków współpracy będą prowadzone w sposób elektroniczny.

W specyfikacji *ebXML* *Collaboration Protocol Profile and Agreement Specification* przedstawione zostały struktury XML-owych dokumentów wymienianych w ramach prowadzenia dialogu zmierzającego do ustalenia CPA. Jeden z tych dokumentów, CPP (*Collaboration Protocol Profile*), opisuje profil partnera, w tym ofertę biznesową oraz obsługiwane przez MSH sposoby współpracy. Uzgodniony w ramach negocjacji dokument CPA jest tworzony na podstawie wybranych z CPP elementów wspólnych dla partnerów.

Możliwe jest ustalenie wielu wariantów CPA pomiędzy daną stroną a różnymi partnerami biznesowymi.

Każde z ustalonych CPA musi otrzymać unikalny identyfikator, który przekazywany w kopercie przesyłki ebXML jednoznacznie wskaże otrzymującemu MSH, które CPA powinno zostać wykorzystane przy obsłudze przesyłki.

W obecnej wersji Standardu RBE przyjmuje się, że uzgadnianie dokumentu CPA będzie prowadzone dowolnymi środkami, których opis wykracza poza zakres tej Specyfikacji (jest to zgodne z założeniami ebXML, w myśl których dozwolone jest osobne traktowanie każdej ze specyfikacji ebXML i sukcesywne implementowanie kolejnych części zbioru tych specyfikacji).

# Wymiana danych zgodna z *reliable messaging*

Protokół wymiany danych określany jako *reliable messaging* definiuje szczegółowe zasady wymiany przesyłek pomiędzy MSH (*Message Service Handler*), zapewniające niezawodne dostarczenie przesyłki do adresata i gwarantujące, że do dalszej obsługi zostanie ona przekazana tylko jeden raz.

W modelu tym zaimplementowane zostały rozwiązania oparte o:

1. przekazywanie potwierdzeń o odbiorze przesyłki,
2. zapisywanie otrzymanych i przekazanych przesyłek w zasobach określanych jako *persistent storage*
3. możliwość wielokrotnego przesyłania przesyłek, które mogły nie dotrzeć do adresata (w przypadku, jeśli *SendingMSH* nie otrzymał potwierdzenia otrzymania przesyłki od *ReceivingMSH)*,
4. mechanizmy wykrywania duplikatów wcześniej otrzymanych przesyłek i ustalone zasady reakcji na ich odebranie (oparte o unikalne identyfikatory zapisywane w *persistent storage*).

Obsługa przesyłek zgodnie z protokołem *reliable messaging* jest opcjonalna (należy do dodatkowej funkcjonalności MSH).

Kwestie dotyczące możliwości wymiany danych zgodnie z tym protokołem, a także jego parametry są uzgadniane pomiędzy współpracującymi MSH i zapisywane w umowie o współpracy CPA (*Collaboration Protocol Agreement –*patrz pkt *CPA – umowa o współpracy partnerów*).

## *Persistent storage*

*Persistent storage,* które jest wymagane w protokole *reliable messaging*, rozumiane jest jako sposób magazynowania danych gwarantujący, że nie zostaną one utracone w wyniku awarii systemu.

W przypadku ewentualnej awarii wymagane jest, aby MSH zapewniło przetworzenie zapisanych w *persistent storage* przesyłek w taki sam sposób jakby awaria nie wystąpiła (sposób rozwiązania zależny jest od implementacji).

MSH obsługujące *reliable messaging* musi przechowywać w *persistent storage* dane przesyłek odebranych i wysłanych (z uwzględnieniem przekazywanych odpowiedzi). Przechowywanie przesyłek (całych lub tylko informacji o nich) jest niezbędne ze względu na potencjalną konieczność ponownego przesłania kopii przesyłki i filtrowania duplikatów przez *ReceivingMSH.*

Zakres danych, które będą zapisywane przez *ReceivingMSH* w *persistent storage* jest zależny od implementacji, niemniej jednak wymagane jest stosowanie następujących zasad:

1. konieczne jest, aby *ToPartyMSH* zachowało m.in. *MessageId* otrzymanej przesyłki (wymagane ze względu na konieczność filtrowania duplikatów),
2. zalecane jest, aby *ReceivingMSH* zachowało:
   1. całą otrzymaną przesyłkę, przynajmniej dopóki informacje z niej nie zostaną pobrane przez *Aplikację biznesową* lub inny proces odpowiedzialny za ich dalsze przetwarzanie,
   2. czas otrzymania komunikatu (informacja ta jest wykorzystywana w odpowiedzi na *Message Status Response –* patrzpkt *Komunikat Message Status Response*),
   3. kompletną przesyłkę przekazaną w odpowiedzi (przechowywanie przesłanych odpowiedzi pozwala na przekazanie ich kopii w odpowiedzi na duplikat zapytania, co zabezpiecza system przed ponownym przetwarzaniem przesyłki na poziomie merytorycznym).

## Alternatywne sposoby implementacji protokołu gwarantującego niezawodność dostarczania przesyłek

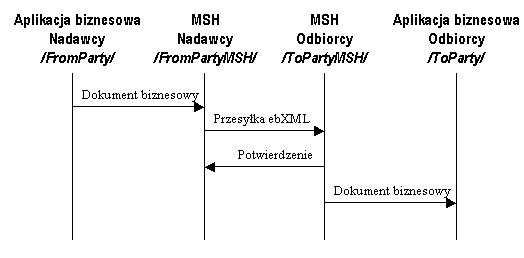
Zgodnie ze specyfikacją ebXML MSS współpracujące MSH mogą realizować niezawodne dostarczanie w oparciu o jeden z następujących sposobów, polegających na:

1. zastosowaniu protokołu *reliable messaging*,
2. wykorzystaniu ebXML-owych elementów przekazywanych w strukturze koperty SOAP w komercyjnym oprogramowaniu, zapewniającym niezawodne dostarczanie w ramach alternatywnych protokołów,
3. przeniesieniu pewnych zasad (np. dotyczących wykrywania duplikatów) do *Aplikacji biznesowej* użytkownika,
4. zastosowaniu kombinacji powyższych rozwiązań.

Sposób implementacji niezawodnego dostarczania musi być ustalony pomiędzy współpracującymi MSH i zapisany w CPA.

## Zasady wymiany danych w protokole *ebXML reliable messaging*

Poniższy rysunek przedstawia podstawowy schemat wymiany przesyłki zgodnie z protokołem *reliable messaging*:



Zaimplementowane w tym protokole zasady zapewniające niezawodne dostarczenie przesyłki do adresata i gwarantujące, że *Aplikacja biznesowa* przetworzy dostarczony w przesyłce dokument tylko raz, oparte są o dwa podstawowe mechanizmy:

1. przekazywanie potwierdzeń o otrzymaniu przesyłki (niezawodność dostarczenia);

otrzymanie potwierdzenie oznacza, że przesyłka została przez *ReceivingMSH* odebrana i przetworzona albo umieszczona w *persistent storage*,

1. wykrywanie duplikatów wcześniej przekazanych przesyłek (gwarancja jednokrotnego przetworzenia przez *Aplikację biznesową*).

Wymogi co do zastosowania powyższych mechanizmów przez MSH otrzymujące przesyłkę są podawane w kopercie przesyłki lub ustalane pomiędzy współpracującymi MSH i zapisywane w CPA.

Pozwala to na odmienne traktowanie przesyłek w zależności od charakteru i ważności przekazywanych w nich dokumentów biznesowych, np.:

1. jeśli dokumenty biznesowe przekazują ważne dane np. zlecenia płatnicze, przy obsłudze przesyłki powinny być wykorzystane oba mechanizmy,
2. jeśli dokumenty biznesowe stanowią zapytania o informacje to np. wykrywanie duplikatów można pominąć.

*Uwaga: Różne kombinacje ustawień wymuszających stosowanie powyższych i związane z nimi tryby dostarczenia przesyłki (np. once-and-only-once, at-least-once) są opisane w rozdziale "Tryby przekazywania przesyłek w reliable messaging".*

### Generowanie potwierdzeń

Zgodnie z *reliable messaging* *ReceivingMSH* musi wygenerować potwierdzenie otrzymania przesyłki, jeśli występuje w niej element *<AckRequested>* z wartością atrybutu *SOAP:actor* wskazującą na dane MSH, które w danej wymianie może pełnić rolę:

1. ostatecznego odbiorcy (*ToPartyMSH*),
2. kolejnego MSH (*NextMSH*) w modelu *Multi-hop.*

Szczegółowy opis zawarty w pkt. *Specyfikacja elementów koperty ebXML – AckRequested.*

Potwierdzenie jest przekazywane w postaci ebXML-owej przesyłki, która w kopercie jako minimum musi zawierać element *<Acknowledgement>* z podelementem *<RefToMessageId>*, zawierającym tę samą wartość co element *<MessageId>* przesyłki potwierdzanej.

W ebXML dopuszcza się możliwość przekazywania potwierdzeń razem z odpowiedzią z *Aplikacji biznesowej* dotyczącą dokumentu otrzymanego jako *payload* potwierdzanej przesyłki.

W związku z tym w ebXML stosowane są dwa rodzaje potwierdzeń:

1. techniczne: realizujące tylko funkcję potwierdzenia odbioru przesyłki przez MSH,
2. biznesowe: przekazujące odpowiedź z aplikacji biznesowej.

W tej dokumentacji dla rozróżnienia, potwierdzenia te są odpowiednio określane jako:

1. *Komunikat potwierdzający* (techniczne potwierdzenie),
2. *Wiadomość potwierdzająca* (biznesowe potwierdzenie).

*Uwaga:*

*Dla uproszczenia, w tej Specyfikacji, jeśli rodzaj potwierdzenia nie będzie istotny, będzie używane określenie Komunikat potwierdzający (lub przesyłka potwierdzająca).*

Rodzaj potwierdzenia jest uzależniony od parametrów w CPA.

Przesyłka potwierdzająca musi być zapisana w *persistent storage* i przechowywana przez okres identyczny jak przesyłka potwierdzana (czas przechowywania jest określony parametrem *PersistDuration).*

#### Komunikat potwierdzający

*Komunikat* potwierdzający jest ebXML-ową przesyłką nie zawierającą *paylaod*-ów, realizującą tylko funkcję potwierdzenia otrzymania przesyłki przez MSH (nie przekazuje żadnych informacji z *Aplikacji biznesowej*). Wartości podelementów elementu *<MessageHeader>* tego komunikatu są ściśle określone przez specyfikację ebXML:

* 1. *<Service>*: musi zawierać "*urn:oasis:names:tc: ebxml-msg:service*",
  2. *<Action>* musi zawierać wartość "*Acknowledgement*",
  3. wartości elementu *<From>* mogą być pobrane z elementu *<To>* otrzymanej przesyłki z uwzględnieniem wszystkich podelementów podanych w *<To>*,
  4. *<RefToMessageId>* musi zawierać *MessageId* potwierdzanej przesyłki.

#### Wiadomość potwierdzająca

*Wiadomość potwierdzająca* realizuje funkcję potwierdzenia, ale jednocześnie przekazuje odpowiedź z *Aplikacji biznesowej* wygenerowaną w ramach obsługi dokumentów biznesowych przesłanych jako *payload*-y potwierdzanej przesyłki.

*Wiadomość potwierdzająca* jest ebXML-ową przesyłką z *paylaod*-em, w którym jest przekazywana odpowiedź biznesowa:

1. w tym przypadku wartości podelementów *<MessageHeader>* są uzależnione od ustawień *<Service>* i *<Action>* powiązanych z biznesową odpowiedzią,
   1. element *<RefToMessageId>,* analogicznie jak w poprzednim przypadku, zawiera *MessageId* potwierdzanej przesyłki.

*Wiadomość potwierdzająca* może być przekazywana, jeśli wartość parametru *syncReplyMode* w CPA nie jest równa "*none*" i wskazuje, że w potwierdzeniu ma być uwzględniona odpowiedź z *Aplikacji biznesowej*.

Przy zachowaniu tych warunków przesłanie *Wiadomości potwierdzającej* jest wymuszone wystąpieniem w przesyłce elementu *<SyncReply>.*

*Uwaga:*

*Jeśli odpowiedź z Aplikacji biznesowej nie ma być przesłana w trybie synchronicznym, to jest ona przekazywana jako payload nowej przesyłki, a nie potwierdzenia. Ewentualne jej powiązanie z pierwotnym dokumentem odbywa się na poziomie Aplikacji biznesowej. Możliwe jest wykorzystanie elementów <RefToMessageId> w <MessageData> i <ConversationId>* .

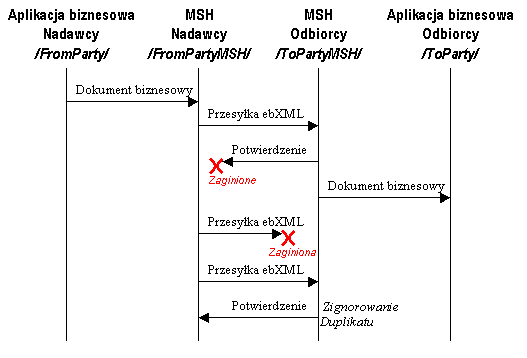
### Wykrywanie i obsługa duplikatów

W punkcie tym przedstawione są ogólne zasady obsługi przesyłki, która zawiera informacje z żądaniem:

1. przekazania potwierdzenia otrzymania przesyłki (występuje element *<AckRequested>)*,
2. wykrywania duplikatów (występuje element *<DuplicateElimination>)*.

*Uwaga:*

*Kombinacja takich ustawień jest najbardziej typowa dla reliable messaging, chociaż w ebXML dopuszcza się również zastosowanie mechanizmu wykrywania duplikatów niezależnie od ebXML-owych potwierdzeń w wypadku, jeśli niezawodność dostarczenia jest realizowana na innym poziomie (np. protokołu transportu).*



Powyższy diagram przedstawia przepływ danych pomiędzy MSH wysyłającym i MSH otrzymującym przesyłkę w ramach wymiany realizowanej zgodnie z następującymi zasadami *reliable messaging:*

1. *FromPartyMSH* (MSH nadawcy) musi powtórnie przesłać "identyczną przesyłkę", jeśli nie otrzymał *Komunikatu potwierdzającego*.
2. Kiedy *ToPartyMSH* (MSH odbiorcy) otrzymuje "duplikat przesyłki", musi ponownie przesłać do *FromPartyMSH* (MSH nadawcy) "pierwszy komunikat odpowiedzi".
3. *ToPartyMSH* (MSH odbiorcy) nie wolno  ponownie przekazać duplikatu przesyłki do *Aplikacji biznesowej*.

W powyższym opisie użyte zostały następujące określenia:

1. "identyczna przesyłka" - przesyłka ebXML, która zawiera taki sam *<SOAP:Header>*, *<SOAP:Body>* i *payload(y)* jak przesyłka wcześniej przesłana,
2. "duplikat przesyłki" *-* przesyłka, która ma ten sam *<MessageId>* jak przesyłka wcześniej otrzymana,
3. "pierwszy komunikat odpowiedzi" *-*  przesyłka z *Komunikatem potwierdzającym,* która ma najwcześniejszy *<Timestamp>* w elemencie *<MessageData>* i która posiada ten sam *<RefToMessageId>* jak "duplikat przesyłki".

## Szczegółowy opis zasad wymiany danych w protokole *ebXML reliable messaging*

W poniższych punktach opisany jest szczegółowy sposób realizacji podstawowych funkcji MSH związanych z wymianą danych takich jak:

1. wysyłanie przesyłki,
2. odbiór przesyłki z uwzględnieniem przekazania w odpowiedzi *Komunikatu potwierdzającego*,
3. ponowne wysłanie przesyłki,
4. ponowne wysłanie potwierdzenia.

Akcje podejmowane przez MSH w ramach obsługi tych funkcji są przedstawione na diagramach i opisane w postaci scenariusza.

### Wysyłanie przesyłki

MSH po otrzymaniu z *Aplikacji biznesowej* danych, które mają być przesłane zgodnie z protokołem *reliable messaging,* przygotowuje przesyłkę zawierającą przekazany dokument biznesowy i wysyła ją do adresata.



Procedura wysłania przesyłki obejmuje następujące kroki:

1. Utworzenie przesyłki składającej się z komponentów otrzymanych z *Aplikacji biznesowej* nadawcy (*FromParty*).
2. Dołożenie elementu *<AckRequested>*, jeżeli odbiór przesyłki ma być potwierdzony przez odbiorcę przesyłki (*ToPartyMSH / ReceivingMSH*) *–* patrz szczegółowy opis *elementu <AckRequested>* w *Specyfikacji elementów koperty ebXML*.
3. Zapisanie przesyłki w *persistent storage*.
4. Przesłanie przesyłki do adresata (*ReceivingMSH  / ToPartyMSH*)*.*
5. Jeżeli żądane jest potwierdzenie otrzymania przesyłki przez *ReceivingMSH*:
6. oczekiwanie aż *ReceivingMSH* prześle *Komunikat potwierdzający*:
   1. jeśli potwierdzenie nie zostanie odebrane przed upływem czasu określonego parametrem *RetryInterval* lub, jeśli zgłoszony zostanie błąd przez protokół komunikacyjny, wysyłające MSH (*FromPartyMSH / SendingMSH)* podejmuje akcję opisaną w pkt. *Ponowne przesłanie przesyłki,*
   2. jeżeli potwierdzenie zostanie odebrane, przesyłka oznaczana jest w *persistent storage* jako *delivered*.

### Odbiór przesyłki

MSH po otrzymaniu przesyłki podejmuje akcje zgodnie z procedurą uzależnioną od rodzaju otrzymanej przesyłki.

Rodzaj jest określany na podstawie zawartości koperty otrzymanej przesyłki, określającej funkcję przesyłki i wymogi dotyczące jej przetworzenia.

Otrzymana przesyłka może:

1. być *Komunikatem potwierdzającym* lub *Wiadomością potwierdzającą* (w przesyłce występuje element <*Acknowledgement*> stanowiący podelement w *<SOAP:Header>* - szczegółowy opis dotyczący elementów występujących w obu potwierdzeniach znajduje się w pkt. *Generowanie potwierdzeń*),
2. nie być *Komunikatem (Wiadomością) potwierdzającym(ą)* (w przesyłce nie występuje element *<Ackowledgement>*) i stanowić:
   1. przesyłkę, która nie podlega kontroli duplikatów,
      1. w *<SOAP:Header>* koperty nie występuje element *<DuplicateElimination>*,
   2. nową przesyłkę,
      1. w *persistent storage* nie zostaje znaleziony *MessageId* zgodny z wartością *<MessageId>* otrzymanej przesyłki,
   3. duplikat wcześniejszej przesyłki,
      1. w *persistent storage* zostaje znaleziony wcześniej zapisany *MessageId* zgodny z wartością *<MessageId>* otrzymanej przesyłki,
      2. *uwaga:* sprawdzanie duplikatów jest wykonywane tylko, jeśli w przesyłce występuje element <*DuplicateElimination*>.

Akcje podejmowane przez MSH po odbiorze przesyłki określonego rodzaju są przedstawione na poniższym schemacie. Diagram przedstawia wszystkie kroki MSH z uwzględnieniem akcji związanych z określeniem rodzaju przesyłki (są one przedstawione na diagramie w postaci wyróżnionych kolorem bloków decyzyjnych).

Związek Banków Polskich Bazy i Systemy Bankowe Sp. z o.o.

Rada Bankowości Elektronicznej

Standard RBE strona - 27 / 106

Część I: Specyfikacja warstwy komunikacyjnej – wersja 1.0



Scenariusz opisujący przedstawiony powyżej diagram obsługi odbioru przesyłki został podzielony na osobne części obejmujące akcje podejmowane przez MSH po rozpoznaniu rodzaju przesyłki.

Jeśli otrzymana przesyłka jest *Komunikatem potwierdzającym*

(w przesyłce występuje element *<Acknowledgement >)*

1. MSH wyszukuje w *persistent storage* przesyłkę, której dotyczy otrzymany *Komunikat potwierdzający* (szukany jest zapisany *MessageId* wysłanej wcześniej przesyłki zgodny z wartością <*RefToMessageId>* otrzymanego *Komunikatu potwierdzającego*).
2. Jeśli przesyłka zostanie odszukana w *persistent storage*, wówczas jest ona zaznaczana jako *delivered*. Jeśli nie, oznacza to że otrzymane zostało nieoczekiwane potwierdzenie, które jest przez MSH ignorowane (MSH nie podejmuje żadnych akcji, nie jest też raportowany żaden błąd do nadawcy potwierdzenia).

Przesyłka, która nie podlega kontroli duplikatów

(w przesyłce nie występuje element *<DuplicateElimination>)*

1. MSH przekazuje dane do *Aplikacji biznesowej* odbiorcy (*ToParty*). Zakłada się, że w przypadku braku *<DuplicateElimination>* *ReceivingMSH* przekazuje dane do *Aplikacji biznesowej*, która może realizować funkcje wykrywania duplikatów na poziomie biznesowym. Wykrywanie duplikatów na poziomie MSH nie jest w tym wypadku obowiązkowe, ale jest dozwolone.
2. Jeśli w przesyłce jest zawarte żądanie przesłania potwierdzenia (występuje element *<AckRequested>), ReceivingMSH*:
3. generuje *Komunikat potwierdzający* zgodnie z zasadami opisanymi w pkt. *Generowanie potwierdzenia*,
4. wysyła *Komunikat potwierdzający* do *SendingMSH*.

Nowa przesyłka

(w *persistent storage* nie został znaleziony *MessageId*)

1. MSH zapisuje *MessageId* otrzymanej przesyłki w *persistent storage*. W *persistent storage* może, ale nie musi, zostać zapisany cały komunikat.
2. MSH przekazuje przesyłkę do dalszego przetwarzania:
3. do *Aplikacji biznesowej*, jeśli *ReceivingMSH* jest ostatecznym odbiorcą (*ToPartyMSH*),
4. do kolejnego MSH (*NextMSH)*, jeśli *ReceivingMSH* jest pośredniczącym MSH przy przekazywaniu przesyłki w modelu *Multi-hop*.
5. Jeśli w przesyłce jest zawarte żądanie przesłania potwierdzenia (występuje element *<AckRequested>), ReceivingMSH*:
6. generuje *Komunikat potwierdzający* zgodnie z zasadami opisanymi w pkt *Generowanie potwierdzeń*,
7. zapisuje *Komunikat potwierdzający* w *persistent storage*,
8. wysyła potwierdzenie do *SendingMSH*.

Duplikat wcześniejszej przesyłki

(w *persistent storage* zostałznaleziony *MessageId*)

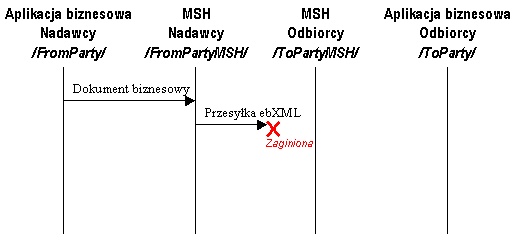
1. MSH sprawdza czy w przesyłce jest zawarte żądanie przesłania potwierdzenia (czy występuje element *<AckRequested>*)
2. jeśli nie, MSH nie podejmuje żadnych akcji,
3. jeśli tak, MSH w odpowiedzi powinien ponownie przesłać *Komunikat potwierdzający* wygenerowany w odpowiedzi na wcześniejszą przesyłkę.
4. Jeśli w przesyłce jest element *<AckRequested>* (żądanie potwierdzenia)
5. MSH szuka w *persistent storage* odpowiedzi na wcześniej otrzymaną przesyłkę (według <*RefToMessageId>*, które jest zgodne z *<MessageId>* otrzymanego komunikatu),
   1. jeśli komunikat został znaleziony wówczas jest on przesyłany ponownie do *SendingMSH,*  które przesłało otrzymaną przesyłkę,
   2. jeśli komunikat nie został znaleziony, możliwe są przypadki:
      1. zamiast technicznego *Komunikatu potwierdzającego* w odpowiedzi oczekiwana jest *Wiadomość potwierdzająca* (patrz w pkt *Wiadomość potwierdzająca*)*,* która jeszcze nie została wygenerowana przez *Aplikację biznesową*; w takim wypadku MSH:
         1. czeka na odpowiedź z aplikacji,
         2. po jej otrzymaniu generuje *Wiadomość potwierdzającą*,
      2. oczekiwane było tylko potwierdzenie techniczne, którego nie ma w *persistent storage*; w takim wypadku MSH:
         1. generuje *Komunikat potwierdzający*,
6. po wygenerowaniu potwierdzenia (*Komunikatu* lub *Wiadomości potwierdzającej*, zgodnie z opisem powyżej), MSH:
   1. zapisuje przesyłkę potwierdzającą w *persistent storage*,
   2. wysyła przesyłkę potwierdzającą do *SendingMSH*.

### Ponowne przesłanie przesyłki

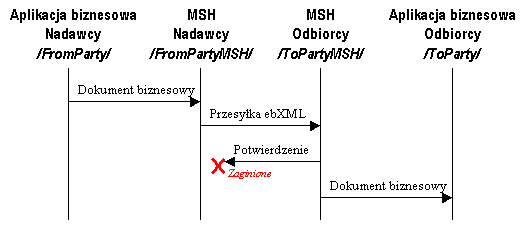
W punkcie tym opisane jest postępowanie *SendingMSH* w przypadku, jeśli nie otrzyma informacji o tym, że przesyłka dotarła do adresata.

Sytuacja taka może mieć miejsce, jeśli:

1. przesłana przesyłka nie dotrze do *ReceivingMSH*,



1. potwierdzenie wysłane przez *ReceivingMSH* nie dotrze do *SendingMSH*.



1. wysłanie przesyłki przez *SendingMSH* zakończy się nierozpoznawalnym błędem na poziomie protokołu transportowego (np. HTTP, SMTP).

Jeżeli w wysłanej przesyłce zawarte było żądanie przesłania potwierdzenia (występował element *<AckRequested>*), *SendingMSH* w sytuacjach wymienionych powyżej podejmuje akcję ponownego przesłania przesyłki realizowaną zgodnie z procedurą przedstawioną na poniższym diagramie.



1. *SendingMSH* ponownie wysyła oryginalną przesyłkę zgodnie z parametrami wymiany, tzn. jeśli:
2. minął już wymagany czas określony parametrem *RetryInterval* (czas przerwy pomiędzy kolejnymi próbami) od momentu poprzedniego przesłania przesyłki,
3. przesyłka została wcześniej przesłana mniej razy niż liczba dopuszczalnych prób ponownego przesłania, która jest określona w parametrze *Retries*.
4. Jeśli *SendingMSH* nie otrzymał potwierdzenia po maksymalnej liczbie prób określonej w parametrze *Retries*, powinien powiadomić *Aplikację biznesową* o braku potwierdzenia odbioru przesyłki przez adresata.

*Uwaga:*

*Stosowanie procedury uwzględniającej ilość możliwych prób ponownego przesłania i czas przerwy pomiędzy nimi jest konieczne również w przypadku, jeśli na etapie wysyłania SendingMSH otrzyma informacje o nierozpoznawalnym błędzie na poziomie transportu wiadomości (np. HTTP, SMTP).*

### Ponowne przesłanie potwierdzenia

Ponowne przesłanie potwierdzenia w odpowiedzi na otrzymaną przesyłkę zostało uwzględnione na diagramie przedstawionym w punkcie *Odbiór przesyłki.*

MSH wysyła wcześniej wygenerowane potwierdzenie, jeśli stwierdzi, że otrzymana przesyłka jest duplikatem i w *persistent storage* znajdzie przesyłkę potwierdzającą, która była wysłana w odpowiedzi.

### Postępowanie w przypadku niepowodzenia dostarczenia przesyłki do Adresata

W poniższym punkcie opisane zostało postępowanie, które jest podejmowane przez MSH w przypadku, jeśli wystąpią błędy w trakcie przekazywania danych do kolejnego procesu odpowiedzialnego za dalsze przetwarzanie przesyłki tzn. do:

1. *Aplikacji biznesowej*, jeśli *ReceivingMSH* jest ostatecznym odbiorcą (*ToPartyMSH*),
2. *NextMSH*, jeśli *ReceivingMSH* jest ogniwem pośredniczącym (*IntermediateMSH*) w wymianie przesyłki w modelu *Multi-hop*.

Jeśli otrzymana przesyłka, która zawiera żądanie potwierdzenia (w przesyłce występuje element *<AckRequested>)*  nie może być dostarczona do kolejnego procesu, MSH powinien przesłać do nadawcy wiadomości (*FromParty*) zawiadomienie o niepowodzeniu dostarczenia.

Zawiadomienie o niepowodzeniu dostarczenia jest przekazywane w postaci *Komunikatu o błędzie* z następującymi wartościami atrybutów elementu *<Error>:*

1. *errorCode =* "*DeliveryFailure*",
2. *severity* =
   1. "*Error*" - jeśli strona, która wykryła problem nie mogła przekazać wiadomości (np. jeśli błędy wystąpiły na poziomie protokołu transportowego),
   2. "*Warning*" – jeśli wiadomość (przesyłka lub dokument biznesowy) została przekazana, ale nie zostało odebrane potwierdzenie odbioru od procesu otrzymującego (oznacza to, że prawdopodobnie wiadomość nie została dostarczona, chociaż istnieje prawdopodobieństwo, że kolejny proces odebrał wiadomość).

Informacja o niedostarczeniu przesyłki musi być przekazana do MSH, który jest nadawcą przesyłki *(FromPartyMSH)*.

Jeśli *Komunikat o błędzie* typu "*DeliveryFailure*" nie może być dostarczony pomyślnie z jakiegoś powodu, wówczas MSH musi wykorzystać inne sposoby przekazania informacji zgodnie z wcześniejszymi ustaleniami pomiędzy współpracującymi stronami (dodatkowe środki komunikowania się MSH są poza zakresem tej specyfikacji).

*Uwaga:*

*Jeśli FromPartyMSH otrzyma Komunikat potwierdzający od ToPartyMSH powinien zignorować wszystkie Komunikaty o błędach* "*DeliveryFailure*"*, które otrzymał.*

## Tryby przekazywania przesyłek w *reliable messaging*

Informacje wymuszające na *ReceivingMSH* zastosowanie mechanizmów *reliable messaging* w odniesieniu do otrzymanej przesyłki są zawarte w następujących elementach koperty przesyłki:

1. *<DuplicateElimination>* - wystąpienie tego elementu określa, że duplikaty przesyłek powinny być wykrywane przez *ToPartyMSH*; zakłada się, że w modelu *Multi-hop* tylko ostateczny odbiorca wykrywa duplikaty przesyłek, gdyż realizacja tej funkcji przez *IntermediateMSH* mogłyby kolidować z parametrami *reliable messaging* ustalonymi pomiędzy końcowymi MSH (*FromParty, ToParty*),
2. *<AckRequested>* - wystąpienie tego elementu określa, że oczekiwane jest potwierdzenie odbioru przesyłki od:
   1. ostatecznego odbiorcy przesyłki (*ToPartyMSH*),
   2. kolejnego MSH (*NextMSH*) pośredniczącego w wymianie w modelu *Multi-hop*.

Strony (lub strona) zobowiązane do przesłania potwierdzenia są określane w atrybucie *SOAP:actor –* bliższe informacje patrz pkt *Atrybuty o zasięgu publicznym*. W poniższej tabeli są przedstawione różne kombinacje ustawień *<DuplicateElimination>* i *<AckRequested>* (wskazujące tylko na *ToPartyMSH* , tylko na *NextMSH* lub na oba), które powodują, że przesyłka jest dostarczana w trybie:

1. *Once-And-Only-Once* - przesyłka dociera do adresata jeden i tylko jeden raz,
2. *At-Least-Once* - przesyłka dociera do adresata przynajmniej raz,
3. *At-Most-Once* - przesyłka dociera do adresata co najwyżej raz,
4. *Best-Effort* - przesyłka jest przekazywana do adresata z pominięciem mechanizmów *reliable messaging*.

Zastosowane w tabeli określenie *End-to-End* oznacza wymianę pomiędzy końcowymi ogniwami modelu wymiany (tutaj pomiędzy *FromPartyMSH* i *ToPartyMSH*).

|  | Duplicate  Elimination | AckRequested  ToPartyMSH | AckRequested  NextMSH | Tryb dostarczenia przesyłki  Komentarz |
| --- | --- | --- | --- | --- |
| 1 | Y | Y | Y | **Once-And-Only-Once** *reliable messaging* na poziomie End-to-End  At-Least-Once na poziomie Pośrednika  Pośrednik i *ToPartyMSH* mogą generować zawiadomienie o niepowodzeniu dostarczenia (*DeliveryFailure)* |
| 2 | Y | Y | N | **Once-And-Only-Once** *reliable messaging* na poziomie End-to-End oparte tylko na retransmisji przesyłek na poziomie End-to-End |
| 3 | Y | N | Y | **At-Least-Once** *reliable messaging na poziomie IntermediateMSH*  W tym trybie Once-And-Only-Once dla End-to-End jest realizowane pośrednio  Brak potwierdzeń przekazywanych pomiędzy End-to-End |
| 4 | Y | N | N | **At-Most-Once**  Tylko eliminacja duplikatów poprzez *ToParty*  Brak retransmisji przesyłki przez Pośredników i *FromPartyMSH* |
| 5 | N | Y | Y | **At-Least-Once** *reliable messaging* z możliwościąduplikatów na poziomie Pośredników i *ToParty* |
| 6 | N | Y | N | **At-Least-Once** *reliable messaging* z możliwościąduplikatów na poziomie Pośredników i *ToParty* |
| 7 | N | N | Y | **At-Least-Once** *reliable messaging* na poziomie *NextMSH*  Brak potwierdzeń przekazywanych pomiędzy End-to-End |
| 8 | N | N | N | **Best Effort** |

## Parametry protokołu *reliable messaging*

W poniższych punktach opisane zostały parametry określające zachowania MSH związane z obsługą przesyłek zgodnie z protokołem *reliable messaging*. Większość z tych parametrów może być określona w dokumencie CPA, w którym są zapisywane ustalone pomiędzy partnerami warunki współpracy MSH.

### DuplicateElimination

Eliminacja duplikatów przesyłki przez MSH jest uzależniona od:

1. wartości parametru *duplicateElimination* zapisanego w CPA,
2. wystąpienia elementu *<DuplicateElimination>* w przekazywanej przesyłce.

Wystąpienie elementu *<DuplicateElimination>* w przesyłce jest uzależnione od wartości *duplicateElimination w CPA:*

1. jeśli jest ona równa "*never*"*, <DuplicateElimination>* nie może wystąpić w przesyłce,
2. w przeciwnym wypadku, *<DuplicateElimination>* musi wystąpić w przesyłce zawsze wtedy, gdy *FromPartyMSH* żąda eliminowania duplikatów przez *ToPartyMSH*.

Jeśli *<DuplicateElimination>* występuje, *ToPartyMSH* musi działać zgodnie z zasadami *reliable messaging* powodującymi ignorowanie duplikatów wiadomości:

* 1. musi zachować wiadomość w *persistent storage*,
  2. wykryć duplikat,
  3. zapewnić, że wiadomość będzie przekazana do *Aplikacji biznesowej* co najwyżej raz (*At-Most-Once*).

Jeśli *<DuplicateElimination>* nie występuje, nie wymaga się (chociaż jest to dozwolone), aby *ToParty* sprawdzało czy otrzymana wiadomość jest duplikatem:

* 1. zapis w *persistent storage* przez *ToPartyMSH,* umożliwiający wykrycie duplikatu, nie jest wymagany,
  2. duplikaty mogą być przekazywane do *Aplikacji biznesowej*.

*ToPartyMSH* powinien raportować błąd do *FromPartyMSH* w przypadkach jeśli:

* 1. nie jest w stanie obsłużyć żądanej funkcjonalności,
  2. wartość *duplicationElimination* w CPA nie jest zgodna z wystąpieniem/brakiem elementu *<DuplicateElimination>* w przesyłce.

W tych sytuacjach w odpowiedzi jest przekazywany *Komunikat o błędzie* zawierający element *<ErrorList>* z podelementem *<Error>* o następujących wartościach atrybutów:

* 1. *errorCode =* "*Inconsistent*"
  2. *severity =* "*Error*".

### AckRequested

Powyższy parametr przekazywany w przesyłce określa, czy *SendingMSH* żąda, aby dany *ReceivingMSH* (występujący w roli zgodnej z wartością parametru *SOAP:actor*) przesłał przesyłkę zawierającą element *<Acknowledgement>.*

W zależności od wystąpienia elementu *<AckRequested>* i wartości w nim przekazanych przesyłka potwierdzająca może:

1. nie być przekazywana,
2. być przekazywana z podpisem cyfrowym *ReceivingMSH*,
3. być przekazywana i nie zawierać podpisu *ReceivingMSH*.

Więcej informacji na temat potwierdzeń w punktach:

1. *Komunikat* (*Wiadomość) potwierdzająca*,
2. opis <*AckRequested>* i <*Acknowledgement>* w *Specyfikacji elementów koperty ebXML*.

### Retries

Parametr *Retries*, podawany w CPA, jest wartością typu *integer*, określającą maksymalną liczbę prób, jakie powinno wykonać *SendingMSH*, aby przy użyciu tego samego protokołu komunikacyjnego dostarczyć przesyłkę, której odbiór nie został potwierdzony przez *ReceivingMSH.*

### RetryInterval

Parametr *RetryInterval*, podawany w CPA, jest wartością typu *duration* z XML Schema. Wartość ta określa minimalny odstęp czasu pomiędzy kolejnymi próbami wysłania przesyłki przez *SendingMSH*, jeśli nie otrzymał on przesyłki potwierdzającej lub jeśli stwierdzono błędy komunikacyjne w czasie próby przesłania przesyłki.

### TimeToLive

Wartość podawana w elemencie *<TimeToLive>* wskazuje na czas, wyrażony jako UTC, do którego przesyłka powinna być dostarczona do ostatecznego odbiorcy *ToPartyMSH*. Wartość ta musi być zgodna z typem XML Schema *dateTime*.

*TimeToLive* mija, jeśli czas wewnętrznego zegara u *ToPartyMSH*, odniesiony do UTC, przekracza wartość elementu *<TimeToLive>* podaną w przesyłce.

Jeśli *ToPartyMSH* otrzymuje komunikat, dla którego minął *TimeToLive*, powinien przesłać do *FromParty* *Komunikat o błędzie* informujący o tym fakcie. Komunikat ten powinien zawierać element *<ErrorList>* z podelementem *<Error>* o następujących wartościach atrybutów:

1. *errorCode =*"*TimeToLiveExpired*",
2. *severity =* "*Error*"*.*

W *reliable messaging* musi być spełniona zależność:

*TimeToLive* > *Timestamp* + ((*Retries* +1) \* *RetryInterval*)

gdzie:

1. *Retries* i *RetryInterval* są parametrami *reliable messaging*,
2. wartość *<Timestamp*> jest pobierana z *<MessageData>*.

### PersistDuration

Parametr *PersistDuration*, podawany w CPA, jest minimalną długością przedziału czasu wyrażoną jako *duration* (typ *XML Schema*), przez który dane z otrzymanego komunikatu są przechowywane w *persistent storage* przez *ToPartyMSH*.

Po upływie czasu *PersistDuration* od momentu, kiedy przesyłka została przekazana po raz pierwszy, *SendingMSH* nie może przesyłać ponownie wiadomości z tym samym *MessageId*.

Jeśli przesyłka nie może być pomyślnie dostarczona przed upływem czasu określonego w *PersistDuration*, wówczas *SendingMSH* powinno zgłosić raport o awarii dostarczenia (*patrz opis w pkt. Postępowanie w przypadku niepowodzenia dostarczenia przesyłki*).

W *reliable messaging* musi być spełniona zależność:

*TimeStamp + PersistDuration > TimeToLive*

gdzie:

1. *TimeStamp i TimeToLive* są wartościami elementów przekazywanych w *<MessageHeader>*,
2. *PersistDuration* jest parametrem podawanym w CPA.

### syncReplyMode

Parametr *syncReplyMode*, podawany w CPA, jest wykorzystywany tylko, jeśli używany jest synchroniczny protokół transportu (np. HTTP). W innych wypadkach wartość *syncReplyMode* jest ignorowana.

Parametr ten określa typ odpowiedzi, jaka musi zostać synchronicznie przesłana w przypadku wystąpienia elementu *<SyncReply>* w strukturze *<SOAP:Header>*.

Dopuszczalne wartości *syncReplyMode*:

1. "*mshSignalsOnly*" – oczekiwane jest techniczne potwierdzenie generowane przez MSH, tzw. *Komunikat potwierdzający* (patrz pkt *Komunikat potwierdzający*),
2. "*signalsOnly*" - oczekiwane jest biznesowe potwierdzenie zawierające informacje o obecnym stanie transakcji realizowanej w *Aplikacji biznesowej*,
3. "*responseOnly*" – oczekiwana jest odpowiedź biznesowa zawierająca dane uzyskane po zakończeniu przetwarzania transakcji w *Aplikacji biznesowej*,
4. "*signalsAndResponse*" – oczekiwane jest biznesowe potwierdzenie lub odpowiedź biznesowa,
5. "*none*" - żadne informacje z poziomu MSH lub *Aplikacji biznesowej* nie będą przekazywane synchronicznie (przyjmowana jako wartość domyślna, jeśli *syncReplyMode* nie występuje).

Jeśli wartość parametru jest różna od "*none*" w przesyłce musi występować element *<SyncReply>*.

*ToPartyMSH* powinno raportować błąd, jeśli parametr CPA *syncReplyMode*="*none*" i w otrzymanej przesyłce występuje element <*SyncReply>*. *Komunikat o błędzie* przekazywany w odpowiedzi zawiera element *<ErrorList>* z podelementem *<Error>* o następujących wartościach atrybutów:

1. *errorCode =* "*Inconsistent*",
2. *severity =* "*Error*"*.*

# Dodatkowe moduły funkcjonalne MSH

## Moduł obsługi błędów

Obsługa błędów należy do podstawowej funkcjonalności MSH.

W poniższym punkcie opisane są rodzaje błędów i zasady ich raportowania pomiędzy współpracującymi MSH.

Przesyłki ebXML stanowiące dokumenty SOAP, zawierające w podelementach *<SOAP:Body>* i *<SOAP:Header>* ebXML-owe rozszerzenia, są obsługiwane przez procesory SOAP, które w warstwie aplikacyjnej obsługują dodatkowe elementy rozszerzeń ebXML.

Procesory te mogą zgłaszać błędy z poziomu przetwarzania:

1. w warstwie SOAP
2. w warstwie ebXML, która stanowi warstwę aplikacyjną dla procesora SOAP

Błędy z poziomu warstwy SOAP, raportowane jako komunikaty *SOAP Fault,* są zgłaszane w przypadku jeśli przesyłka nie może być obsłużona w warstwie aplikacyjnej ebXML (zawiera błędy uniemożliwiające interpretację danych), a tym samym nie może zostać wygenerowany ebXML-owy *Komunikat o błędzie*.

Moduł obsługi błędów w *SendingMSH* musi być przygotowany do odbioru i przetworzenia zarówno ebXML-owych *Komunikatów o błędzie* jak i wartości błędów zgłaszanych w komunikatach *SOAP Fault.*

### Definicje

W punkcie *Moduł obsługi błędów* wykorzystywane są dwa określenia:

1. *Przesyłka z błędem* – przesyłka zawierająca lub powodująca błąd lub ostrzeżenie, nazywana również *przesyłką błędną* (ang. *message in error*)
2. *Komunikat o błędzie* – przesyłka zawierająca element <*ErrorList>* opisujący ostrzeżenia lub błędy znalezione w *przesyłce z błędem* (w ebXML MSS określany jako *Error Message*).

### Typy błędów

Otrzymanie przesyłki może się wiązać z wygenerowaniem błędu na poziomie:

1. protokołu transportowego
2. warstwy SOAP
3. warstwy ebXML

Błędy na poziomie protokołu transportowego są wykrywane i raportowane przy użyciu standardowych mechanizmów wspieranych przez te protokoły i ich obsługa nie jest implementowana w opisywanym tu *Module obsługi błędów*.

Błędy zgłaszane na poziomie SOAP i struktura komunikatu *SOAP Fault* są zdefiniowane w [SOAP].

Błędy zgłaszane przez MSH na poziomie ebXML są zdefiniowane w bieżącej Specyfikacji.

Obejmują one :

1. błędy dotyczące elementów rozszerzeń ebXML w kopercie SOAP,
2. błędy protokołu *reliable messaging*,
3. błędy zabezpieczeń.

Szczegółowa lista zgłaszanych błędów znajduje się w pkt. *Specyfikacja elementów koperty ebXML* (*patrz opis elementu <ErrorList>*).

### Implementacja obsługi błędów

#### Zasady generowania komunikatów o błędach

Jeśli MSH wykryje w przesyłce błąd, wówczas powinno zgłosić to do MSH, które wygenerowało *Przesyłkę z błędem*. Jest to możliwe wtedy, gdy:

1. znane jest *Error Reporting Location (patrz opis w kolejnym punkcie)*, do którego powinien być przesłany *Komunikat o błędzie*,
2. *Przesyłka z błędem* nie posiada elementu <*ErrorList>* z wartością atrybutu *highestSeverity* ustawioną na "*Error*" (oznacza to, że *Komunikaty o błędzie* nie są generowane w odpowiedzi na *Komunikaty o błędzie*).

Jeśli *Error Reporting Location* nie jest znane lub *Przesyłka z błędem* ma element <*ErrorList>* z atrybutem *highestSeverity* ustawionym na "*Error*", wówczas zalecane jest, aby:

1. informacja o błędzie była wprowadzona do logu,
2. problem został rozwiązany za pomocą innych środków,
3. żadna dalsza akcja nie była podejmowana przez MSH.

#### Ustalenie *Error Reporting Location*

*Error Reporting Location* jest URI określonym przez nadawcę *Przesyłki z błędem*, które wskazuje, dokąd przesłać *Komunikat o błędzie*.

*Error Reporting Location* powinno być określone poprzez parametr *ErrorURI* określony w CPA, które jest identyfikowane przez wartość elementu <*CPAId*> *Przesyłki z błędem*. Jeśli nie jest to możliwe, odbiorca może ustalić *ErrorURI* w oparciu o element <*From*> *Przesyłki z błędem*. Dopuszcza się, by w sytuacjach, gdy *Przesyłka z błędem* nie może zostać poprawnie przeanalizowana, MSH próbowało ustalić *Error Reporting Location* innymi sposobami, zależnymi od implementacji MSH. Jeśli to również nie jest możliwe, *Komunikat o błędzie* nie zostanie wysłany.

#### Przekazywanie informacji o błędzie

Informacje o błędzie lub uwagi dotyczące *Przesyłki z błędem* są przekazywane w elemencie *<ErrorList>*, stanowiącym podelement *<SOAP:Header>.*

Element ten może być:

1. dołączony do typowej przesyłki, która jest generowana w odpowiedzi na otrzymaną przesyłkę (np. zawierającej w *payload*-zie dokument biznesowy typowy dla funkcji realizowanej przez przesyłkę)
   1. sposób ten nie może być stosowany jeśli jest przekazywana informacja o znaczeniu *"Error"* wskazanym poprzez wartość atrybutu *highestSeverity* w *<ErrorList>*
2. zawarty w osobnej przesyłce określonej wcześniej jako *Komunikat o błędzie* (komunikat techniczny bez payload-ów)

Wartości elementów *<Service>* i *<Action>* przesyłek zawierających *<ErrorList>* są różne dla obu przypadków. W pierwszym są typowe dla odpowiedzi biznesowej, w której *<ErrorList>* jest dodatkowym elementem, w którym są przekazywane uwagi (*highestSeverity* =*"Warning"*) do otrzymanej przesyłki.

W drugim mają wartości wskazujące na techniczny *Komunikat o błędzie*.

### Struktura Komunikatu o błędzie

*Komunikat o błędzie* jest przesyłką ebXML, nie zawierającą *payload*-ów.

W strukturze komunikatu są wykorzystane następujące elementy koperty ebXML:

1. w *<SOAP:Header>*:
   1. *<MessageHeader>* z podelementami:
      1. *<From>* - identyfikacja strony, która wysłała *Komunikat o błędzie*,
      2. *<To>* - identyfikacja strony, do której przesyłany jest *Komunikat o błędzie*,
      3. *<Service>*, który musi mieć wartość *"urn:oasis:names:ebxml-msg:service”*,
      4. *<Action>*, który musi mieć wartość *"MessageError”*,
      5. *<MessageData>*, w którym jest podawany identyfikator przesyłki której dotyczy *Komunikat o błędzie*,
   2. *<ErrorList>,* w którym jest podawana lista błędów wykrytych w *Przesyłce z błędem*
   3. *<ds:Signature>* (opcjonalnie, wykorzystywane do uwierzytelnienia nadawcy komunikatu)

Pełna definicja podanych elementów jest podana w pkt. *Specyfikacja elementów koperty ebXML.*

## Moduł obsługi sekwencji przesyłek

Obsługa sekwencji przesyłek należy do opcjonalnej funkcjonalności MSH.

Implementacja tego modułu pozwala na przekazywanie *Aplikacji biznesowej* strony *ToParty* danych z otrzymanych przesyłek w kolejności określonej przez sekwencje powiązanych przesyłek.

Przesyłki, które mają być obsłużone jako sekwencja, zawierają w kopercie element *<MessageOrder>* i muszą być zawsze przekazywane zgodnie z protokołem *reliable messaging* w trybie *once-and-only-once* (patrz pkt *Tryb przekazywania przesyłek w reliable messaging*).

W elemencie *<MessageOrder>* znajduje się podelement *<SequenceNumber>,* w którym są przekazywane numery przesyłek w danej serii. Jeśli jedna z przesyłek nie dotrze do *ToPartyMSH*, to dokumenty z ewentualnych późniejszych przesyłek nie są przekazywane do *Aplikacji biznesowej* *ToParty.*

Stosowanie elementu *<MessageOrder>,* wskazującego, że przesyłka ma być przekazana sekwencyjnie, razem z elementem *<SyncReply>* nie jest dopuszczalne.

Jednoczesne wystąpienie obu elementów musi być raportowane do *SendingMSH* za pomocą *Komunikatu o błędzie* zawierającym w elemencie *<ErrorList>* podelement *<Error>* o wartościach atrybutów:

1. *errorCode =* " *Inconsistent*",
2. *severity =* "*Error*".

Przesyłki sekwencyjne, przekazywane zgodnie z *reliable messaging*, są przechowywane w *persistent storage*. Konieczność przechowywania mija po upływie czasu, który co do wartości jest niemniejszy niż *TimeToLive* + *persistDuration* oraz *retryInterval \* retries*.

Identyfikatorem serii przesyłek, jest wartość elementu <*ConversationId>*. Identyfikator ten jest nadawany przez stronę inicjującą wysyłanie danych (*FromPartyMSH*), jest powtarzany przy wysyłaniu każdej przesyłki z serii. Jest on unikalny dla danej serii w kontekście *FromPartyMSH* i *ToPartyMSH* (*CPAId*).

Informacje dotyczące zasad nadawania numerów sekwencyjnych przesyłkom przekazywanym w ramach danej konwersacji są podane w opisie elementu *<StatusNumber>* w pkt. *Struktura przesyłki ebXML.*

Kolejność przekazywania przesyłek przez *ToPartyMSH* do przetworzenia do *Aplikacji biznesowej* musi być zachowana. MSH nie może przekazać danej przesyłki do przetworzenia, jeśli posiada nie przekazane przesyłki o mniejszej wartości *<SequenceNumber>*.

## Moduł obsługi Komunikatów diagnostycznych

Protokół ebXML dostarcza gotowe rozwiązania dla realizacji usług pozwalających na uzyskiwanie informacji na temat aktualnego stanu infrastruktury ebXML - węzłów MSH. Usługi te są oparte o wymianę *Komunikatów diagnostycznych* *Ping i Pong* realizowaną w modelu zapytanie – odpowiedź.

Wymiana komunikatów *Ping* i *Pong* pozwala na:

1. stwierdzenie czy *ReceivingMSH* jest aktualnie czynne i gotowe do wymiany danych,
2. określenie średniego czasu wymiany przesyłek w podstawowym modelu zapytanie-odpowiedź.

*Komunikaty diagnostyczne* stanowią uproszczone przesyłki ebXML złożone tylko z koperty ebXML zawierającej <*MessageHeader>,* w którym jest identyfikowana funkcja komunikatu ("*Ping*" lub "*Pong*").Dodatkowo koperta może zawierać podpis elektroniczny uwierzytelniający strony wymieniające komunikaty.

Obsługa *Komunikatów diagnostycznych* jest opcjonalna. Jeżeli MSH ich nie obsługuje w reakcji na otrzymanie komunikatu *Ping* powinien przesłać *Komunikat o błędzie* z następującymi wartościami atrybutów elementu *<Error>*:

1. *errorCode=*"*NotSupported*"*,*
2. *severity=*"*Error*"*.*

Zakłada się, że strona otrzymująca komunikat *Ping* powinna zawsze przesłać odpowiedź *Pong.*

Istnieje jednak ryzyko wykorzystania *Ping Message* do ataku typu *denial of service* na MSH.

Jeśli MSH stwierdzi, że żądanie pochodzi od nieautoryzowanego nadawcy, może zrezygnować z odpowiedzi (rozwiązania dotyczące zasad podejmowania decyzji typowe dla implementacji).

### Komunikat diagnostyczny MSH Ping Message

Komunikat *MSH Ping Message* jest przesyłką ebXML, nie zawierającą *payload*-ów.

W strukturze komunikatu są wykorzystane następujące elementy koperty ebXML:

1. w *<SOAP:Header>*:
2. *<MessageHeader>* z podelementami:
3. *<From>* - identyfikacja strony, która wysłała *MSH Ping Message*,
4. *<To>* - identyfikacja strony, do której przesyłana jest *MSH Ping Message*,
5. *<CPAId>*,
6. *<ConversationId>*,
7. *<Service>* z wartością *"urn:oasis:names:ebxml-msg:service”*,
8. *<Action>* z wartością "*Ping*”,
9. *<MessageData>*,
10. *<ds:Signature>* *(opcjonalnie, wykorzystywane do uwierzytelnienia nadawcy komunikatu*.

Pełna definicja podanych elementów jest podana w pkt. *Specyfikacja elementów koperty ebXML.*

Przykładowy komunikat *Ping*

|  |
| --- |
| <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <SOAP:Header>  <eb:MessageHeadereb:version=”2.0” SOAP:mustUnderstand=”1”>  <eb:From><eb:PartyId>urn:send-MSH</eb:PartyId></eb:From>  <eb:To><eb:PartyId>urn:receive-MSH</eb:PartyId></eb:To>  <eb:CPAId> ... </eb:CPAId>  <eb:ConversationId> ... </eb:ConversationId>  <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>  <eb:Action>Ping</eb:Action>  <eb:MessageData>  <eb:MessageId> ... </eb:MessageId>  <eb:Timestamp>2002-07-24T11:12:34<eb:Timestamp>  </eb:MessageData>  </eb:MessageHeader>  </SOAP:Header>  <SOAP:Body/>  </SOAP:Envelope> |

### Komunikat diagnostyczny MSH Pong Message

Komunikat *MSH Pong Message*, przekazywany w odpowiedzi *MSH Ping Message* jest przesyłką ebXMLnie zawierającą *payload*-ów.

W strukturze komunikatu są wykorzystane następujące elementy koperty ebXML:

1. w *<SOAP:Header>*:
2. *<MessageHeader>* z podelementami:
3. *<From>* - identyfikacja strony, która przekazuje *MSH Pong Message*,
4. *<To>* - identyfikacja strony, do której przesyłany jest *MSH Pong Message*,
5. *<CPAId>*,
6. *<ConversationId>*,
7. *<Service>* z wartością "*urn:oasis:names:ebxml-msg:service*",
8. *<Action>* z wartością "*Pong*",
9. *<MessageData>* zawierający m.in.
   * 1. *<RefToMessageId>* identyfikujący *MSH Ping Message*, na który *Pong* odpowiada,
10. *<ds:Signature>* *(opcjonalnie, wykorzystywane do uwierzytelnienia nadawcy komunikatu)*.

Pełna definicja podanych elementów jest podana w pkt. *Specyfikacja elementów koperty ebXML.*

Przykładowy komunikat *Pong*

|  |
| --- |
| <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <SOAP:Header>  <eb:MessageHeader eb:version=”2.0” SOAP:mustUnderstand=”1”>  <eb:From><eb:PartyId>urn:receive-MSH</eb:PartyId></eb:From>  <eb:To><eb:PartyId>urn:send-MSH</eb:PartyId></eb:To>  <eb:CPAId> ... </eb:CPAId>  <eb:ConversationId> ... </eb:ConversationId>  <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>  <eb:Action>Pong</eb:Action>  <eb:MessageData>  <eb:MessageId> ... </eb:MessageId>  <eb:Timestamp>2002-07-24T11:12:34<eb:Timestamp>  <eb:RefToMessageId> ... </eb:RefToMessageId>  </eb:MessageData>  </eb:MessageHeader>  </SOAP:Header>  <SOAP:Body/>  </SOAP:Envelope> |

## Moduł obsługi Komunikatów statusowych

*Komunikaty statusowe* *MessageStatus* pozwalają wysyłającemu MSH monitorować status przesyłek ebXML uprzednio przekazanych do innego MSH.

Usługa ta jest obsługiwana przez parę komunikatów *Message Status Request* i *Message Status Response.*

MSH po otrzymaniu komunikatu *Message Status Request:*

1. powinien przesłać *Message Status Response* jeśli zapytanie dotyczy przesyłki, przekazanej zgodnie z protokołem *reliable messaging*,
2. zalecenie to nie obowiązuje, jeśli zapytanie o status jest przesłane po czasie określonym poprzez parametr *PersistDuration* (po tym czasie *ReceivingMSH* nie jest zobowiązane do przechowywania przesyłki w *persistent storage*),
3. może (ale nie musi, zależy to od rozwiązań przyjętych w implementacji MSH) przesłać odpowiedź na zapytanie dotyczące pozostałych przesyłek (nie przesłanych zgodnie z *reliable messaging*).

*Komunikaty statusowe* nie powinny być wykorzystywane do zapewnienia niezawodności przekazania przesyłki (funkcja ta jest realizowana w oparciu o zasady *reliable messaging* oparte o przekazywanie potwierdzeń).

Obsługa *Komunikatów statusowych* jest opcjonalna. Jeżeli MSH ich nie obsługuje w reakcji na otrzymanie *Message Status Request* powinien przesłać *Komunikat o błędzie* z następującymi wartościami atrybutów elementu *<Error>*:

1. errorCode="*NotSupported*",
2. severity="*Error*".

### Komunikat Message Status Request

Komunikat *Message Status Request,* realizujący funkcję żądania przesłania informacji o statusie przekazanej wcześniej przesyłki, jest przesyłką ebXML nie zawierającą *payload*-ów.

W strukturze komunikatu są wykorzystane następujące elementy koperty ebXML:

1. w *<SOAP:Header>*:
2. *<MessageHeader>* z podelementami:
3. *<From>* - identyfikacja strony, która wysłała *Message Status Request*,
4. *<To>* - identyfikacja strony, która powinna przesłać odpowiedź dotyczącą statusu,
5. *<Service>* z wartością „urn:oasis:names:ebxml-msg:service”,
6. *<Action>* z wartością "*StatusRequest*",
7. *<Message Data>*, w którym nie jest wykorzystany podelement *<RefToMessageId>*, pozostałe wg. Specyfikacji,
8. *<ds:Signature>* *(opcjonalnie, wykorzystywane do uwierzytelnienia nadawcy komunikatu)*,
9. w *<SOAP:Body>*,
10. *<StatusRequest>* z podelementem *<RefToMessageId>* identyfikującym przesyłkę, której dotyczy zapytanie

Pełna definicja podanych elementów jest podana w pkt. *Specyfikacja elementów koperty ebXML*.

Przykładowy komunikat *Message Status Request*

|  |
| --- |
| <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <SOAP:Header>  <eb:MessageHeader eb:version=”2.0” SOAP:mustUnderstand=”1”>  <eb:From><eb:PartyId>urn:send-MSH</eb:PartyId></eb:From>  <eb:To><eb:PartyId>urn:receive-MSH</eb:PartyId></eb:To>  <eb:CPAId> ... </eb:CPAId>  <eb:ConversationId> ... </eb:ConversationId>  <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>  <eb:Action>StatusRequest</eb:Action>  **<eb:MessageData>**  **<eb:MessageId> status\_reguest\_message\_id </eb:MessageId>**  <eb:Timestamp>2002-07-24T11:12:34<eb:Timestamp>  </eb:MessageData>  </eb:MessageHeader>  </SOAP:Header>  <SOAP:Body>  **<eb:StatusRequest eb:version=”2.0”>**  **<eb:RefToMessageId> requested\_status\_message\_id </eb:RefToMessageId>**  </eb:StatusRequest>  </SOAP:Body>  </SOAP:Envelope> |

### Komunikat Message Status Response

Komunikat *Message Status Response,* generowany przez MSH w odpowiedzi na *Message Status Request,* jest przesyłką ebXML, nie zawierającą *payload*-ów.

W strukturze komunikatu są wykorzystane następujące elementy koperty ebXML:

1. w *<SOAP:Header>*:
2. *<MessageHeader>* z podelementami:
3. *<From>* - identyfikacja strony, która wysłała *Message Status Response*,
4. *<To>* - identyfikacja strony, do której przesyłana jest odpowiedź,
5. *<Service>* z wartością "*urn:oasis:names:ebxml-msg:service*",
6. *<Action>* z wartością "*StatusResponse*",
7. *<MessageData>*, z podelementem *<RefToMessageId>* zawierającym wartość *<MessageId>* z komunikatu *Message Status Request* (patrz: przykłady),
8. *<ds:Signature>* (*opcjonalnie, wykorzystywane do uwierzytelnienia nadawcy komunikatu)*,
9. w *<SOAP:Body>*,
10. *<StatusResponse>* z atrybutem *messageStatus* i podelementami:
11. *<RefToMessageId>* - identyfikacja przesyłki, której dotyczy zapytanie (patrz: przykłady),
12. *<TimeStamp>* - czas odebrania przesyłki, której status jest raportowany.

Pełna definicja podanych elementów jest podana w pkt. *Specyfikacja elementów koperty ebXML*

W atrybucie *messageStatus* jest przekazywana jedna z poniższych wartości:

1. "*NotRecognized*" jeśli przesyłka nie została znaleziona przez MSH,
2. "*Received*" jeśli przesyłka została otrzymana przez MSH,
3. "*Processed*" jeśli przesyłka jest przetwarzana w MSH,
4. "*Forwarded*" jeśli przesyłka została przesłana do *NextMSH* w ramach wymiany *Multi-Hop*,
5. "*UnAuthorized*" jeśli komunikat *Message Status Request* nie został poprawnie zautoryzowany.

Zakłada się, że strona otrzymująca żądanie przesłania informacji o statusie powinna zawsze przesłać odpowiedź. Jeśli jednak żądanie pochodzi od nieautoryzowanego nadawcy, może ona zrezygnować z odpowiedzi, zamiast przesyłać *Message Status Response* ze statusem "*UnAuthorized*" (rozwiązania dotyczące zasad podejmowania decyzji typowe dla implementacji).

Przykładowy komunikat *Message Status Response*

|  |
| --- |
| <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <SOAP:Header>  <eb:MessageHeader eb:version=”2.0” SOAP:mustUnderstand=”1”>  <eb:From><eb:PartyId>urn:receive-MSH</eb:PartyId></eb:From>  <eb:To><eb:PartyId>urn:send-MSH</eb:PartyId></eb:To>  <eb:CPAId> ... </eb:CPAId>  <eb:ConversationId> ... </eb:ConversationId>  <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>  <eb:Action>StatusResponse</eb:Action>  <eb:**MessageData**>  **<eb:MessageId> status\_response\_message\_id </eb:MessageId>**  <eb:Timestamp>2002-07-24T11:12:34<eb:Timestamp>  **<eb:RefMessageId> status\_reguest\_message\_id </eb:RefMessageId>**  </eb:MessageData>  </eb:MessageHeader>  </SOAP:Header>  <SOAP:Body>  <eb:**StatusResponse** eb:version=”2.0” eb:messageStatus=”Received”>  **<eb: RefToMessageId> requested\_status\_message\_id </eb:RefToMessageId>**  <eb:Timestamp>2002-07-24T11:01:01<eb:Timestamp>  </eb:StatusResponse>  </SOAP:Body>  </SOAP:Envelope> |

# Warstwa zabezpieczeń w ebXML

## Wprowadzenie

EbXML jako protokół wymiany danych jest przystosowany do korzystania z szerokiego zakresu usług kryptograficznych zapewniających:

1. integralność przesyłanych komunikatów ebXML, w tym dołączonych dokumentów ebXML,
2. uwierzytelnienie i autoryzację MSH wymieniających dane,
3. poufność informacji wymienianych pomiędzy współpracującymi MSH,
4. znakowanie czasem, (ang.. *trusted timestamping),* przesyłanych informacji.

Zabezpieczenia są implementowane na dwóch poziomach:

1. na poziomie warstwy transportowej,
2. na poziomie przesyłki ebXML.

Weryfikacja zabezpieczeń na poziomie przesyłki ebXML może zostać przesunięta w czasie względem procesu wysłania, odbioru samej przesyłki przez MSH.

Zabezpieczenia na poziomie warstwy transportowej (np. IPSEC, TLS, SSL) są ściśle związane z procesem inicjowania połączenia sieciowego oraz transmisji danych.

## Parametry CPA

Zakres zabezpieczeń obsługiwanych lub wymaganych przez MSH jest parametryzowany i definiowany w CPA (patrz pkt *Parametry do zdefiniowania w CPA*). Parametryzowane są następujące elementy:

1. rodzaj zabezpieczeń na poziomie warstwy transportowej (np.: konkretne wersje SSL, IPSEC, TLS),
2. rodzaj algorytmów kryptograficznych oraz funkcji skrótu w przypadku stosowania podpisu cyfrowego,
3. metody szyfrowania dokumentów biznesowych w strukturze MIME (np.: S/MIME, PGP/MIME).

## Zabezpieczenia na poziomie warstwy transportowej

W celu zabezpieczenia warstwy transportowej dopuszczalne jest stosowanie bezpiecznych protokołów sieciowych np. TLS [TLS], SSL [SSLv3] lub IPSEC [IPSEC]. Wykorzystanie jednego z wyżej wymienionych protokołów gwarantuje:

* integralność pakietów danych transmitowanych przez połączenie sieciowe,
* przezroczyste dla MSH nadawcy i odbiorcy szyfrowanie danych podczas transportu przez sieć,
* obustronne uwierzytelnienia MSH nadawcy i odbiorcy z wykorzystaniem certyfikatów, przed rozpoczęciem sesji połączenia,
* autoryzacje MSH nadawcy przesyłek ebXML.

## Szyfrowanie w przesyłce ebXML

W celu zapewnienia poufności danych zawartych w przesyłce ebXML zalecane jest szyfrowanie elementów nagłówka koperty SOAP/ebXML oraz dokumentów biznesowych.

### Szyfrowanie elementów nagłówka koperty SOAP/ebXML

W przypadku szyfrowania elementów nagłówka koperty SOAP/ebXML zalecane jest implementowanie specyfikacji *XML Encryption*, opisującej metody selektywnego szyfrowania elementów dokumentów XML. Specyfikacja ta powstaje w wyniku wspólnej pracy W3C i IETF, przewiduje się, że prace zostaną ukończone w przyszłym tj. 2003 roku. Po uzyskaniu przez *XML Encryption* statusu rekomendacji W3C, standard *XML Encryption* stanie się jedynym, zgodnym z *ebXML MSS*, rozwiązaniem w zakresie szyfrowania elementów koperty SOAP/ebXML.

### Szyfrowanie dokumentów biznesowych

Poufność dokumentów biznesowych w formacie XML, zawartych w przesyłce ebXML,może być realizowana z wykorzystaniem metod selektywnego szyfrowania opisanych w specyfikacji *XML Encryption* [XMLENC].

Dodatkowo możliwe jest zaimplementowanie funkcjonalności związanej z szyfrowaniem dokumentów biznesowych, bazującej na wybranym standardzie S/MIME [S/MIME], [S/MIMECH] lub PGP/MIME [PGP/MIME]. Obsługa standardu S/MIME lub PGP/MIME musi być wzajemnie uzgodniona w ramach dokumentu CPA przez MSH stron wymieniających poufne dane.

Jeżeli MSH wymaga zarówno podpisu jak i szyfrowania, najpierw należy dokument podpisać, a następnie zaszyfrować.

## Podpis elektroniczny w przesyłce ebXML

Dane zawarte w przesyłce ebXML mogą zostać podpisane elektroniczne. Format struktury XML przenoszącej podpis musi być zgodny ze standardem XMLSignature opracowanym przez *IETF/W3C XML Signature Working Group* [XMLDSIG].

### XML Signature

W XML Signature podpisywany cyfrowo jest element <*ds:SignedInfo*> zawierający listę referencji do obiektów, których integralność i niezaprzeczalność pochodzenia ma być poświadczona.

Każda referencja (element <*ds:Reference*>) jest jednoznacznie przypisana do reprezentowanego obiektu, zawiera m.in. wyliczony skrót danych obiektu. Pojedynczy podpis cyfrowy zawarty w XML Signature może, zatem obejmować zarówno dołączone komunikaty biznesowe jak i poszczególne elementy koperty SOAP/ebXML.

W ramach struktur XML Signature można także przenosić dane kryptograficzne służące do sprawdzenia zawartego podpisu cyfrowego. Rodzaj przekazywanych danych (identyfikatory, klucze publiczne, certyfikaty) jest uzależniony od przyjętej przez strony wymieniające dane polityki podpisu.

### XML Signature w nagłówku SOAP:Header

Podpisana przesyłka ebXML musi zawierać przynajmniej jeden element <*ds:Signature>* - element główny struktury XML Signature - jako element potomny nagłówka <*SOAP:Header>*. Dodatkowo każdy element <*ds:Signature>* musi być zadeklarowany w przestrzeni nazw identyfikowanej przez URI http://ww.w3.org/222/09/xmldsig#*.*

Pierwsze wystąpienie elementu <*ds:Signature*> w przesyłce ebXML nie będącej *Komunikatem potwierdzającym* zawsze reprezentuje sygnaturę wystawioną przez stronę *FromPartyMSH* i obejmuje zarówno dołączone komunikaty biznesowe jak i wszystkie elementy koperty SOAP/ebXML, za wyjątkiem danych dla *NextMSH* w wymianie *Multi-hop.* Wygenerowany w ten sposób element <*ds:Signature*>, poświadcza m.in. integralność danych zawartych w kopercie ebXML oraz dołączonych dokumentów biznesowych. Stanowi także dowód uwierzytelniający nadawcę *FromPartyMSH* przesyłki ebXML.

Kolejne elementy <*ds:Signature*> dołączane przez *FromPartyMSH* nie są weryfikowane przez *ToPartyMSH*. Dodatkowe elementy <*ds:Signature*> mogą być zatem wykorzystane w celu przenoszenia podpisów cyfrowych generowanych i weryfikowana przez *Aplikacje biznesowe*. Możliwe jest zaimplementowanie usługi podpisu, w tym wielokrotnego, pod samodzielnym dokumentem biznesowym.

Fragment standardowej koperty SOAP/ebXML ze sygnaturą XML.

|  |
| --- |
| <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd"  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <SOAP:Header>  <eb:MessageHeader eb:id=”...” eb:version=”2.0” SOAP:mustUnderstand=”1”>  .....  </eb:MessageHeader>  **<ds:Signature ds:Id="sig1”>**  **<ds:SignedInfo>**  **<ds:CanonicalizationMethod ds:Algorithm="http://www.w3.org/TR/2001/**  **REC-xml-c14n-20010315"/>**  **<ds:SignatureMethod ds:Algorithm="http://www.w3.org/2000/09/**  **xmldsig#dsa-sha1"/>**  **<ds:Reference ds:URI="">**  <ds:Transforms>  ...  </Transforms>  <ds:DigestMethod ds:Algorithm="http://www.w3.org/2000/09/  xmldsig#sha1"/>  <ds:DigestValue>jKF4Ni01OvKtMuOQ87GFH4Vu8nk=</DigestValue>  **</ds:Reference>**  **<ds:Reference ds:URI="cid://fddd1fae-1dec-11d0-a715-00a0c91e6bf6">**  <ds:Transforms>  ...  </ds:Transforms>  <ds:DigestMethod ds:Algorithm="http://www.w3.org/2000/09/  xmldsig#sha1"/>  <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>  **</ds:Reference>**  **</ds:SignedInfo>**  **<ds:SignatureValue>MC0CFFrVLtRlk=...</ds:SignatureValue>**  **<ds:KeyInfo>**  <ds:KeyValue>  <ds:DSAKeyValue>  <ds:P>...</ds:P><ds:Q>...</ds:Q><ds:G>...</ds:G><ds:Y>...</ds:Y>  </ds:DSAKeyValue>  </ds:KeyValue>  **</ds:KeyInfo>**  **</ds:Signature>** |
| </SOAP:Header>  <SOAP:Body>  <eb:Manifest eb:id="Mani01">  <eb:Reference xlink:href="cid://fddd1fae-1dec-11d0-a715-00a0c91e6bf6">  </eb:Reference>  </eb:Manifest>  </SOAP:Body>  </SOAP:Envelope> |

### Podpis w Komunikacie potwierdzającym

*Komunikaty potwierdzające* są wykorzystywane do uzyskania niezaprzeczalności odbioru w wymianie przesyłek ebXML. *Komunikat potwierdzający* musi zostać podpisany przez *ReceivingMSH* w przypadku, kiedy potwierdzana przesyłka ebXML zawiera element <*AckRequested*> z atrybutem  *signed=”true”*.

Złożony pod *Komunikatem potwierdzającym* podpis cyfrowy stanowi dowód uwierzytelniający nadawcę komunikatu *(ReceivingMSH)* oraz m.in. poświadcza integralność zawartego w tym komunikacie elementu <*Acknowledgment*>. Zawartość elementu <*Acknowledgment*> w przypadku podpisanego *Komunikatu potwierdzającego* zawsze obejmuje listę referencji <*ds:Reference*> ze skrótami danych potwierdzanej przesyłki ebXML.

*SendingMSH* po sprawdzeniu listy skrótów w podpisanym *Komunikacie potwierdzającym* uzyskuje pewność, iż potwierdzana przesyłka ebXML została odebrana przez *ReceivingMSH* i dotarła tam w nienaruszonej postaci.

### Podpis w modelu wymiany Multi-hop

W modelu wymiany danych *Multi-hop* funkcjonalność pośredniczącego MSH - *IntermediateMSH - jest* ograniczona do przekazywania (odbioru i wysłania), przechowywania i potwierdzania przychodzących przesyłek ebXML.

Pośredniczące MSH nie weryfikuje poprawności podpisu elektronicznego w przekazywanych przesyłkach ebXML i nie raportuje błędu *SecurityFailure* wysyłającemu *MSH* (*SendingMSH*).

Pośredniczące MSH nie może dokonywać żadnych modyfikacji w przesyłce ebXML, za wyjątkiem elementów koperty SOAP/ebXML zawierających atrybut *SOAP:actor* wskazujący na *IntermediateMSH.*

Wszelkie zmiany w niedozwolonych elementach mogłyby naruszyć integralność podpisu cyfrowy wystawionego przez *FromPartyMSH*, który jest weryfikowany przez *ToPartyMSH*. W szczególności pośredniczące MSH nie dołącza własnej sygantury XML do nagłówka <*SOAP:Header*> przesyłki ebXML.

Jeżeli w przesyłce odebranej przez pośredniczące MSH pojawi się żądanie wygenerowania podpisanego *Komunikatu potwierdzającego*, to żądany komunikat musi być wysłany jako samodzielna przesyłka ebXML a nie np. dołączony do *Wiadomości potwierdzającej* odebranej z *ToPartyMSH*.

## Bezpieczny podpis elektroniczny w przesyłce ebXML

Wykorzystywany w ebXML format podpisu elektronicznego bazujący na specyfikacji XML Signature umożliwia także stosowanie bezpiecznego podpisu elektronicznego, o którym mowa w *Ustawie o podpisie elektronicznym* (Dz. U. z 2001 r. Nr 130 poz. 1450) oraz Rozporządzeniu Rady Ministrów z dnia 7 sierpnia 2002 (Dz. U. z 2002 r. Nr 128 poz. 1094). Przewiduje się, że ze względów oraganizacyjno-technicznych bezpieczne podpisy elektroniczne będą generowane i weryfikowane przez *Aplikacje biznesowe,* w związku z czym będą przenoszone w przesyłce ebXML przez elementy <*ds:Signature*> nie weryfikowane przez MSH.

Na mocy *Ustawy o podpisie* elektronicznym (Art. 5 ust. 2, Art. 6 ust. 1) dane przesyłane w przesyłce ebXML opatrzone bezpiecznym podpisem elektronicznym weryfikowanym przy pomocy ważnego kwalifikowanego certyfikatu będą równoważne pod względem skutków prawnych dokumentom opatrzonym podpisami własnoręcznymi oraz będą mogły stanowić dowód w postępowaniu sądowym.

### XML Advanced Electronic Signature (XAdES)

W Rozporządzeniu Rady Ministrów do *Ustawy o podpisie elektronicznym* wskazano na specyfikację techniczną ETSI TS 101 903 oraz opisany w niej standard XAdES jako obowiązujący XML-owy format bezpiecznego podpisu elektronicznego. XAdES bazuje ściśle na strukturach i typach danych opracowanych przez IETF / W3C w ramach specyfikacji XML Signature, może być zatem bez przeszkód stosowany w kopercie ebXML jako element *<ds:Signature>* w strukturze <*SOAP:Header*>.

W XAdES zostały zdefiniowane elementy i struktury umożliwiające m.in.:

1. jednoznaczne określenie referencji do kwalifikowanego certyfikatu,
2. jednoznaczny sposób identyfikacji polityki podpisu, zgodnie z którą podpis cyfrowy został wygenerowany,
3. określenie czasu wystawienia podpisu,
4. określenie miejsca (adres) wygenerowania podpisu cyfrowego,
5. określenie typu formatu i kodowania podpisanego obiektu,
6. określenie roli (np. dyrektor sprzedaży itp.) jaką podpisujący dokument pełni w procesie biznesowym związanym z wymianą podpisanych danych,
7. określenie zobowiązań podjętych przez podpisującego wobec podpisanego obiektu w kontekście przyjętej polityki podpisu ( np.: dowód utworzenia, dowód dostarczenia, dowód odbioru),
8. znakowanie czasem podpisywanych obiektów.

Dodatkowo, w ramach rozszerzeń standardu XAdES, umożliwiono:

1. znakowanie czasem samego podpisu cyfrowego (*XAdES-TimeStamping*),
2. dołączanie referencji do dodatkowych danych weryfikujących bezpieczny podpis cyfrowy np.: listy CRL, odpowiedzi OCSP itd. *(XAdES-Complete validation data*).

Wyspecyfikowane w standardzie XAdES elementy XML stanowią zawartość struktury <*qp:QualifyingProperties*>, która jest deklarowana w opcjonalnym podelemencie <*ds:Object>*  elementu <*ds:Signature*>*.*

Element <*qp:QualifyingProperties*> wraz z zawartością jest deklarowany w oddzielnej przestrzeni nazw identyfikowanej przez URI http://uri.etsi.org/01903/v1.1.1#.

Wymagane zabezpieczenie (podpis) elementu <*qp:SignedProperties>* wchodzącego w skład struktury <*qp:QualifyingProperties*> jest realizowane poprzez dodatkowy element <ds:*Reference>* ze skrótem <*qp:SignedProperties>,* w strukturze <*ds:SignedInfo*>, która jest podpisywana cyfrowo*.*

Przykład bezpiecznego podpisu elektroniczny w formacie XAdES.

|  |
| --- |
| <?xml version=”1.0” encoding=”utf-8” ?>  <Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  **xmlns:qp="http://uri.etsi.org/01903/v1.1.1#"**  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >  <ds:SignedInfo>  <ds:CanonicalizationMethod ds:Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>  <ds:SignatureMethod ds:Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>  <ds:Reference ds:URI="">  <ds:Transforms>  <ds:Transform ds:Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>  <ds:Transform ds:Algorithm="http://www.w3.org/TR/199/REC-xpath-19991116">  <ds:Xpath>  ....  </ds:Xpath>  </ds:Transform>  </ds:Transforms>  <ds:DigestMethod ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>  <ds:DigestValue>jKF4Ni01OvKtMuOQ87GFH4Vu8nk=</ds:DigestValue>  </ds:Reference>  <ds:Reference ds:URI="cid://fddd1fae-1dec-11d0-a715-00a0c91e6bf6">  <ds:Transforms>  <ds:Transform ds:Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>  </ds:Transforms>  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>  <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>  </ds:Reference>  **<ds:Reference ds:URI="#SignedProperties" ds:Type="http://uri.etsi.org/01903/v1.1.1#SignedProperties">**  **<ds:Transforms>**  **<ds:Transform ds:Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>**  **</ds:Transforms>**  **<ds:DigestMethod ds:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>**  **<ds:DigestValue>m87RK571mnTSt89ovKtMup4NbeVu8nk=</ds:DigestValue>**  **</ds:Reference>**  </ds:SignedInfo>  <ds:SignatureValue>MC0CFFrVLtRlk=...</ds:SignatureValue>  <ds:KeyInfo>  ...  </ds:KeyInfo>  **<ds:Object >**  **<qp:QualifyingProperties>**  **<qp:SignedProperties qp:Target="#SignatureWithSignedAndUnsignedProperties" qp:ID="SignedProperties">**  **<qp:SignedSignatureProperties>**  **<qp:SigningTime>2002-07-11T12:10:00Z</qp:SigningTime>**  **<qp:SigningCertificate> ... </qp:SigningCertificate>**  **<qp:SignaturePolicyIdentifier> ... </qp:SignaturePolicyIdentifier>**  **</qp:SignedSignatureProperties>**  **<qp:SigneddataObjectProperties>**  **<qp:DataObjectFormat> ... </qp:DataObjectFormat>**  **<qp:AllDataObjectTimeStamp> ... </qp:AllDataObjectTimeStamp>**  **</qp:SigneddataObjectProperties>**  **<qp:SignedProperties>**  **<qp:UnsignedProperties>**  **<qp:UnsignedSignatureProperties>**  **<qp:SignatureTimeStamp> ... </qp:SignatureTimeStamp>**  **<qp:CompleteCertificateRefs> ... </qp:CompleteCertificateRefs>**  **<qp:CompleteRevocationRefs> ... </qp:CompleteRevocationRefs>**  **<qp:SigAndRefsTimeStamp> ... </qp:SigAndRefsTimeStamp>**  **<qp:CertificateValues> ... </qp:CertificateValues>**  **<qp:RevocationValues> ... </qp:RevocationValues>**  **</qp:UnsignedSignatureProperties>**  **</qp:UnsignedProperties>**  **</qp:QualifyingProperties>**  **</ds:Object>**  </ds:Signature> |

# Specyfikacja elementów koperty ebXML

## Informacje wprowadzające

### Stosowane w dokumencie przestrzenie nazw:

Elementy i atrybuty składające się na strukturę koperty ebXML zostały zdefiniowane w następujących specyfikacjach:

* + - 1. *SOAP Messages with Attachments*,
      2. *Message Service Specification Version 2.0 (ebXML MSS)*,
      3. *XML-Signature Syntax and Processing*,
      4. *XML Advanced Electronic Signatures (XAdES)*.

Przestrzeń nazw identyfikująca, w jakiej specyfikacji dany element lub atrybut jest zdefiniowany określona jest poprzez prefiks, który poprzedza nazwę elementu i atrybutu.

W Specyfikacji występują następujące prefiksy:

SOAP*:* - wskazujący na *SOAP Messages with Attachments*,

*ds:* - wskazujący na *XML-Signature Syntax and Processing*,

*qp:* - wskazujący na specyfikację *XML Advanced Electronic Signature*,

*eb*: - wskazujący na ebXML MSS (w Specyfikacji elementy zdefiniowane w tej przestrzeni nazw nie są poprzedzane prefiksem).

*Uwaga:*

W dokumentach XML przy definicji prefiksu, podawany jest adres URL, pod którym można znaleźć specyfikacje lub XML Schema zawierające definicje struktury, np.

1. xmlns:SOAP= <http://schemas.xmlsoap.org/soap/envelope>,
2. xmlns:ds=[http://www.w3.org/2000/09/xmldsig#](http://www.w3.org/2000/09/xmldsig),
3. xmlns:qp=http://uri.etsi.org/01903/v1.1.1#,
4. xmlns:eb= http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd.

### Sposób przygotowania definicji elementów

Opis elementów koperty ebXML przedstawiony jest w tabelach, złożonych z kolumn, w których podawane są następujące dane:

1. ***Nazwa*** –nazwa elementu lub atrybutu w strukturze XML-owej koperty; wyjątkiem są następujące określenia przyjęte w Specyfikacji oznaczające odpowiednio:
   1. *gr. atryb. Header* - wskazanie na grupę atrybutów zdefiniowaną w Specyfikacji (pkt *Grupa atrybutów* "*Header*"),
   2. *gr. atryb. Body* - wskazanie na grupę atrybutów zdefiniowaną w Specyfikacji (pkt *Grupa atrybutów* "*Body*"),
   3. *Dowolny element* – wskazanie, że element opisany w tabeli jest *Elementem rozszerzalnym*,
   4. *Dowolny atrybut* – wskazanie, że zestaw atrybutów elementu może zostać rozszerzony.
2. ***Status/Ilość powt.***
   1. status elementu lub atrybutu przyjmujący wartości: *Opcjonalny*, *Wymagany*,
   2. ilość powtórzeń danego elementu podawana w formacie *[0..n]* lub *[1..n]* (jeśli n nie jest określone to ilość powtórzeń nie jest ograniczona); brak informacji o ilości wystąpień jest równoznaczna ze wskazaniem na maksymalne jednokrotne wystąpienie elementu (pomijane są zapisy *[0..1]* i *[1..1]*).
3. ***Rodzaj*** – wskazanie czym w strukturze XML jest element tabeli określony poprzez nazwę; w kolumnie są podawane określenia, odpowiednio wskazujące na:
   1. *Atrybut* – atrybut elementu XML,
   2. *Grupa atrybutów* – zdefiniowana w Specyfikacji grupa atrybutów patrz pkt *Grupa* atrybutów *Header*, *Grupa atrybutów Body*,
   3. *Element* – pojedynczy element zawierający wartość typu określonego w kolumnie ***Typ***,
   4. *Element pusty* – element, który nie posiada wartości,
   5. *Element złożony* – element zawierający inne elementy,
   6. *Element rozszerzalny* – element, którego strukturę można rozbudować o nowe elementy potomne, zgodnie z potrzebami implementacji MSH.
4. ***Typ*** – typ elementu zgodny z *XML Schema*,
5. ***Opis***– dodatkowy opis (np. przeznaczenie elementu, sposób wykorzystania),
6. ***Wartości dopuszczalne*** – zawiera wartości dopuszczalne, które mogą wystąpić w ramach danego elementu tabeli.

W poszczególnych tabelkach opisujących elementy koperty (podpunkty: *Elementy występujące w SOAP:Header* i *Elementy występujące w SOAP:Body)* przedstawiony jest tylko pierwszy poziom zagłębienia struktury. Elementy złożone stanowiące podelementy danego elementu zdefiniowane są w kolejnych tabelach, do których w wersji elektronicznej Specyfikacji znajdują się hiperłącza.

## Struktura koperty ebXML

|  |  |  |
| --- | --- | --- |
| SOAP:Envelope | Opis | Funkcjonalność |
|  |  |  |
| SOAP:Header |  |  |
| − MessageHeader | Nagłówek przesyłki ebXML |  |
| − − From | Nadawca przesyłki |  |
| − − − PartyId | Identyfikator nadawcy |  |
| − − − Role | Rola w jakiej występuje nadawca |  |
| − − To | Odbiorca przesyłki |  |
| − − − PartyId | Identyfikator odbiorcy |  |
| − − − Role | Rola w jakiej występuje odbiorca |  |
| − − CPAId | Identyfikator umowy CPA |  |
| − − ConversationId | Identyfikator wspólny dla przesyłek składających się na jeden dialog |  |
| − − Service | Identyfikacja usługi, jaką pełni przesyłka | Obsługa podstawowych elementów rozszerzeń ebXML |
| − − Action | Identyfikacja akcji w ramach usługi |  |
| − − MessageData | Dane niezbędne do identyfikacji przesyłki |  |
| − − − MessageId | Unikalny globalnie identyfikator przesyłki |  |
| − − − Timestamp | Czas wygenerowania nagłówka koperty ebXML |  |
| − − − RefToMessageId | Wartość *<MessageId>* wcześniej wysłanej przesyłki |  |
| − − − TimeToLive | Czas po upłynięciu którego przesyłka traci ważność |  |
| − − DuplicateElimination | Wskaźnik eliminacji duplikatów |  |
| − − Description | Tekst opisujący przesyłkę |  |
| − SyncReply | Element ustalający, w jakim trybie ma być przesłana przesyłka (synchronicznie czy asynchronicznie) | Obsługa żądań synchronicznych odpowiedzi |
| − MessageOrder | Element wymuszający zachowanie kolejności przesyłek | Obsługa sekwencji przesyłek |
| − − SequenceNumber | Numer kolejny |  |
| − AckRequested | Żądanie przesyłki potwierdzającej |  |
| − Acknowledgment | Potwierdzenie odbioru przesyłki |  |
| − − RefToMessageId | Identyfikator *<MessageId>* potwierdzanej przesyłki |  |
| − − From | Nadawca potwierdzanej przesyłki | Obsługa reliable messaging |
| − − − PartyId | Identyfikator nadawcy |  |
| − − − Role | Rola jaką pełni nadawca potwierdzanej przesyłki |  |
| − − ds:Reference | Referencja do elementów potwierdzanej przesyłki |  |
| − ErrorList | Lista błędów |  |
| − − **Error** | Informacje o błędzie | Obsługa błędów |
| − − − Description | Opis błędu |  |
| − ds:Signature |  |  |
| − − ds:SignedInfo | Struktura objęta podpisem cyfrowym. |  |
| − − − ds:CanonicalizationMethod | Identyfikator algorytmu kanonizacji XML. |  |
| − − − ds:SignatureMethod | Identyfikator algorytmu kryptograficznego i funkcji skrótu używanych w procesie generowania/weryfikacja podpisu cyfrowego. |  |
| − − − ds.:Reference | Referencja do zasobu objętego podpisem w ramach elementu <*ds:SignedInfo*>. |  |
| − − − − ds:Transforms | Lista przekształceń zasobu XML. |  |
| − − − − − ds:Transform | Pojedyncze przekształcenie XML. |  |
| − − − − ds:DigestMethod | Identyfikator funkcji skrótu dla dowiązanego obiektu. |  |
| − − − − ds:DigestValue | Wartość funkcji skrótu dla dowiązanego obiektu. | Podpis elektroniczny |
| − − ds:SignatureValue | Podpis cyfrowy elementem <*ds:SignedInfo*>. |  |
| − − ds:KeyInfo | Dane kryptograficzne pozwalające na weryfikację poprawności podpisu cyfrowego. |  |
| − − − ds:KeyName | Nazwa, identyfikator klucza. |  |
| − − − ds:KeyValue | Parametry klucza. |  |
| − − − − ds:DSAKeyValue | Parametry klucza publicznego DSA. |  |
| − − − − ds:RSAKeyValue | Parametry klucza publicznego RSA. |  |
| − − − ds:X509Data | Dane dotyczące certyfikatów w formacie X509v3. |  |
| − − − ds:SPKIData | Dane binarne udostępniane przez SPKI. |  |
| − − ds:Object | Opcjonalny element rozszerzalny. |  |
| − − − ds:Manifest | Element grupujący referencje z dowiązanymi obiektami. |  |
| − − − ds:SignaureProperties | Dodatkowe parametry sygnatury XML. |  |
| /SOAP:Header |  |  |
|  |  |  |
| SOAP:Body |  |  |
| − Manifest | Identyfikacja załączników przesyłki | Obsługa podstawowych |
| − − Reference | Referencje do załączników | elementów rozszerzeń ebXML |
| − StatusRequest | Żądanie statusu przesyłki |  |
| − − RefToMessageId | Identyfikator *<MessageId>* przesyłki, której status jest żądany | Obsługa |
| − StatusResponse | Informacje o statusie przesyłki | komunikatów statusowych |
| − − RefToMessageId | Identyfikator *<MessageId>* przesyłki, której dotyczy status |  |
| − − Timestamp | Czas otrzymania przesyłki, której dotyczy status |  |
| /SOAP:Body |  |  |
|  |  |  |
| /SOAP:Envelope |  |  |

## Atrybuty o zasięgu publicznym

### Atrybut    id

Każdy element będący ebXML-owym rozszerzeniem koperty SOAP może być zaopatrzony w atrybut *id*, który umożliwia jego jednoznaczną identyfikację w obrębie wiadomości SOAP.

Może być wykorzystany, jeśli w przesyłce ebXML jest zawarty podpis cyfrowy do identyfikacji dowolnego elementu koperty poprzez podanie wartości atrybutu id (patrz pkt *ds:Reference*).

### Atrybut version

Wskazuje wersję specyfikacji *Message Service Specification,* z którą dana definicja koperty ebXML jest zgodna.

Elementy koperty przesyłki ebXML, których struktura jest zgodna z bieżącą wersją specyfikacji Standardu RBE mają wartość atrybutu równą "*2.0*". Atrybut *version* musi posiadać przestrzeń nazw identyczną z przestrzenią nazw ebXML-owych rozszerzeń koperty SOAP.

Zakłada się, że wszystkie elementy koperty powinny być zgodne z tą sama wersją specyfikacji ebXML MSS.

Stosowanie elementów o innym atrybucie *version* wewnątrz jednej i tej samej koperty ebXML jest dopuszczalne tylko w wyjątkowych przypadkach, jeśli istnieje potrzeba zmiany semantyki elementu przed opublikowaniem nowej wersji specyfikacji ebXML MSS, która taką zmianę będzie uwzględniała.

Taka sytuacja jest dozwolona, pod warunkiem, że współpracujące ze sobą MSH będą rozpoznawać wersje wszystkich elementów koperty. Jeśli wersja elementu(ów) nie zostanie rozpoznana przez MSH musi zostać zgłoszony błąd.

### SOAP:mustUnderstand

Prefiks *SOAP* poprzedzający nazwę *mustUnderstand* oznacza, że atrybut pochodzi z przestrzeni nazw określonej w *XML Schema* definiującej kopertę SOAP.

Atrybut *SOAP:mustUnderstand* umieszczony w elemencie koperty ebXML określa sposób przetwarzania tego elementu, uzależniając sam proces przetwarzania od funkcjonalności zaimplementowanej w serwerze (procesie) odbierającym przesyłkę.

Atrybut może przybierać wartości:

1. "*1*" co oznacza, że proces odbierający przesyłkę musi być w stanie poprawnie zinterpretować dostarczone w elemencie XML dane oraz zareagować na nie zgodnie ze specyfikacją opisującą funkcjonalność elementu. W przeciwnym wypadku proces odbierający powinien odrzucić przesyłkę oraz zgłosić błąd,
2. "*0*" co oznacza, że element i informacja z nim związana może zostać zignorowana, jeżeli proces odbierający nie ma zaimplementowanej funkcjonalności pozwalającej na obsługę danego elementu.

### SOAP:actor

W atrybucie *SOAP:actor* występującym w niektórych elementach koperty ebXML wskazywane jest jakie MSH jest powiązane z danym elementem (kto jest adresatem informacji przekazywanych w elemencie lub od kogo one pochodzą – zależy to od rodzaju elementu).

W atrybucie mogą być przekazywane odpowiednio wartości URI:

1. "*urn:oasis:names:tc:ebxml-msg:actor:nextMSH*" *-* wskazująca, że adresatem/nadawcą informacji w elemencie jest kolejne MSH uczestniczące w wymianie przesyłki (*NextMSH*),
2. "*urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH*" (wartość domyślna atrybutu) - wskazująca, że adresatem/nadawcą informacji w elemencie jest ostateczny odbiorca przesyłki (*ToPartyMSH*).

W dalszych punktach Specyfikacji wartości atrybutu będzie dla uproszczenia podawana w skrócie odpowiednio jako "*NextMSH*", "*ToPartyMSH*"

### Grupy atrybutów

Ze względu na to, że określone sekwencje podanych wyżej atrybutów, występują w wielu elementach koperty ebXML dla potrzeb tego dokumentu zostały zdefiniowane dwie grupy atrybutów:

1. *gr.atr.* "*Header*"
   1. wykorzystywana w definicji elementów występujących w *SOAP:Header*,
2. *gr.atr.* "*Body*"
   1. wykorzystywana w definicji elementów występujących w *SOAP:Body*.

#### Grupa atrybutów "Header"

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| id | Opcjonalny | Atrybut | xsd:ID | Atrybut elementu koperty ebXML umożliwiający jego identyfikację w strukturze SOAP:Message | Zgodny z typem prostym ID W3C Schema |
| version | Wymagany | Atrybut | xsd:string | wersja specyfikacji *Message Service Specification*, z którą definicja danego elementu jest zgodna | "2.0" – jeśli struktura elementu zgodna z bieżącą specyfikacją Standardu RBE  *inna* – patrz opis |
| SOAP:mustUnderstand | Wymagany | Atrybut | xsd:boolean |  | 0 – fałsz  1 – prawda |
| Dowolny atrybut | *Opcjonalny*  *[0..n]* |  |  |  |  |

#### Grupa atrybutów „Body*”*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| Id | Opcjonalny | Atrybut | xsd:ID | Atrybut elementu koperty ebXML umożliwiający jego identyfikację w strukturze SOAP:Message | Zgodny z typem prostym ID W3C Schema |
| Version | Wymagany | Atrybut | xsd:string | wersja specyfikacji *Message Service Specification*, z którą definicja danego elementu jest zgodna | "2.0" – jeśli struktura elementu zgodna z bieżącą specyfikacją Standardu RBE  *inna* – patrz opis |
| Dowolny atrybut | Opcjonalny |  |  |  |  |

## 

## Elementy występujące w SOAP:Header

Element *<SOAP:Header>* jest pierwszym elementem potomnym elementu głównego *<SOAP:Envelope>* koperty ebXML.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| [MessageHeader](#MessageHeader) | Wymagany | Element złożony |  | Element zawiera podstawowe informacje o nadawcy i odbiorcy przesyłki, informacje techniczne identyfikujące przesyłkę ebXML oraz informacje routingowe. |  |
| SyncReply | Opcjonalny | Element rozszerzalny |  | Element umożliwia określenie czy odpowiedź ma być przesłana w trybie synchronicznym. |  |
| [MessageOrder](#MessageOrder) | Opcjonalny | Element złożony |  | Element wykorzystywany w przypadku obsługi sekwencji przesyłek |  |
| AckRequested | Opcjonalny  [0..2] | Element rozszerzalny |  | Element wskazuje, że oczekiwany jest *Komunikat potwierdzający* |  |
| [Acknowledgment](#Acknowledgment) | Opcjonalny  [0..2] | Element złożony |  | Element wykorzystywany w przesyłce potwierdzającej |  |
| [ErrorList](#ErrorList) | Opcjonalny | Element złożony |  | Element występuje w przesyłce zwrotnej informującej o błędzie (-ach) występujących w odebranej przesyłce. |  |
| [ds:Signature](#Signature) | Opcjonalny  [0..n] | Element złożony |  | Występuje w przesyłkach zawierających podpis wysyłającego MSH. |  |

### MessageHeader

Wymagany element złożony *<MessageHeader>* występuje w kopercie ebXML jako pierwszy element potomny elementu *<SOAP:Header>*.

Zawiera podstawowe dane identyfikujące przesyłkę, informacje routingowe dotyczące nadawcy i odbiorcy przesyłki, funkcji realizowanej przez przesyłkę oraz informacje techniczne określające wymogi dotyczące obsługi przesyłki.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| gr. atryb. Header |  | Grupa atrybutów |  |  | SOAPMustUnderstand ="1" |
| [From](#FromTo) | Wymagany | Element złożony |  | Informacje o nadawcy przesyłki. |  |
| [To](#FromTo) | Wymagany | Element złożony |  | Informacje o odbiorcy przesyłki. |  |
| CPAId | Wymagany | Element | xsd:string | Identyfikator umowy CPA |  |
| ConversationId | Wymagany | Element | xsd:string | Identyfikator grupujący przesyłki dotyczące jednego dialogu między stronami. Wartość elementu jest określana przez stronę inicjującą dialog. |  |
| Service | Wymagany | Element | xsd:string | Identyfikuje usługę, jaką pełni przesyłka. | patrz opis poniżej |
| type | Opcjonalny | Atrybut | xsd:string | Atrybut elementu Service |  |
| Action | Wymagany | Element | xsd:string | Identyfikuje akcję w ramach usługi. | patrz opis poniżej |
| [MessageData](#MessageData) | Wymagany | Element złożony |  | Zawiera informacje identyfikujące przesyłkę ebXML. |  |
| DuplicateElimination | Opcjonalny | Element pusty | Brak | Zapewnia sprawdzenie przesyłki pod kątem wystąpienia jej duplikatów. | Brak |
| Description | Opcjonalny  [0..n] | Element | xsd:string | Umożliwia dostarczenie opisu dotyczącego przesyłki. |  |
| xml:lang | Wymagany | Atrybut | xsd:language | Atrybut elementu Description |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

Wymagany element *<CPAId>* identyfikuje CPA, uzgodnione poprzez współpracujące ze sobą strony, określające zestaw parametrów dotyczących wymiany przesyłek.

Zaleca się, aby wartość elementu *<CPAId>* była typu URI. Musi ona być unikalna dla przestrzeni nazw uzgodnionej wzajemnie przez obie strony wymiany. Wartość *<CPAId>* może być:

1. połączeniem wartości *<PartyId>* z elementów *<From>* i *<To>*,
2. URI z prefiksem będącym nazwą domeny internetowej jednej ze stron,
3. przestrzenią nazw zaproponowaną i utrzymywaną przez jedną ze stron.

Jeśli odbiorca przesyłki stwierdzi, że nie spełnia ona parametrów uzgodnionych w CPA, wówczas powinien zgłosić ten fakt, wysyłając zwrotnie *Komunikat o błędzie* z wartościami atrybutów elementu *<Error>* ustawionymi jako:

1. *errorCode="Inconsistent",*
2. *severity="Error"*.

Przykład elementu *<CPAId>*:

<CPAId>http://przykład.com/cpa/moja\_i\_twoja\_umowa\_cpa.xml</CPAId>

Wymagany element *<ConversationId>* grupuje komunikaty wymieniane przez partnerów, składające się na jedną konwersację. Wartość elementu jest określana przez stronę inicjującą konwersację i powinna pozostać niezmieniona we wszystkich przesyłkach, składających się na daną konwersację. Element umożliwia tematyczne powiązanie przesyłek. Wartość elementu *<ConversationId>* jest zależna od implementacji. Nie jest ściśle określone, w jaki sposób w danej implementacji MSH powinna być prowadzona identyfikacja konwersacji umożliwiająca powiązanie przesyłek w ramach danego dialogu.

Wymagany element *<Service>* identyfikuje usługę, jaką pełni przesyłka.

Element <Service> może mieć wartości:

1. ustalone w bieżącej Specyfikacji jeśli przesyłka pełni funkcję komunikatu technicznego bez payload-ów (patrz opisy struktur komunikatów technicznych, np. o błędzie statusowy, potwierdzający itd.)
2. związane z przekazywanym w *payload*-zie przesyłki dokuemntem biznesowym – kody te są zdefiniowane w warstwie biznesowej Standardu RBE (część II dokuemntacji) lub w innych implementacjach wykorzystujących wartwę komunikacyjną Standardu RBE.

Element *<Service>* posiada atrybut *type*. Występuje, jeśli jest znany obu stronom, wysyłającej i odbierającej, i wskazuje jak zinterpretować kod elementu *<Service>.*

Brak atrybutu wymusza podanie wartości elementu *<Service>* jako URI. W przeciwnym wypadku musi być raportowany błąd z wartościami atrybutów elementu <Error>:

1. *errorCode="Inconsistent"*
2. *severity="Error"*

Wymagany element *<Action>* identyfikuje akcję realizowaną w ramach usługi określonej w elemencie *<Service>*. Element *<Action>* przybiera wartości zgodnie z zasadami opisanymi dla wartości elementu <Service> (powinny być unikalne w powiązaniu z elementem *<Service>)*.

Jeśli wartości obu elementów *<Service>* i *<Action>* nie są rozpoznawalne przez *ReceivingMSH* wówczas musi zostać zgłoszony błąd z następującymi wartościami atrybutów elementu *<Error>*:

1. *errorCode*="*NotRecognized*",
2. *severity*="*Error*".

Pusty, opcjonalny element *<DuplicateElimination>* sygnalizuje *ReceivingMSH* konieczność wykrywania duplikatów przesyłki. Element nie może wystąpić w przesyłce, jeżeli w CPA zadeklarowano parametr d*uplicateElimination* o wartości "*never*".

Więcej informacji w opisie parametrów protokołu *reliable messaging (*patrz *pkt Wymiana danych zgodna z protokołem reliable messaging → Parametry protokołu reliable messaging)*.

Opcjonalny element *<Description>* umożliwia dostarczenie tekstowego opisu dotyczącego przesyłki ebXML. Element posiada atrybut *xml:lang* w którym jest określony język opisu. Dopuszczalne są wielokrotne wystąpienia elementu *<Description>.* Każde z wystąpień elementu *<Description>* powinno posiadać inną wartość atrybutu *xml:lang.*

### From / To

Elementy złożone *<From>* i *<To>* posiadają identyczną strukturę. Elementy potomne elementu *<From>* identyfikują nadawcę przesyłki. Elementy potomne elementu *<To>* identyfikują odbiorcę przesyłki.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| PartyId | Wymagany  [1..n] | Element | xsd:string | Identyfikator strony. |  |
| type | Opcjonalny | Atrybut | xsd:string | Atrybut elementu PartyId. Wskazuje na domenę nazw, do której należy zawartość elementu PartyId. |  |
| Role | Opcjonalny | Element | xsd:string | Określa rolę w jakiej występuje nadawca lub odbiorca. |  |

Jeśli powtarzalny element *<PartyId>* występuje więcej niż jeden raz w którymkolwiek z elementów *<From>* lub *<To>* to wszystkie wystąpienia elementu muszą identyfikować tę samą jednostkę (dopuszcza się możliwość stosowania przez daną stronę wielu identyfikatorów).

Element *<PartyId>* posiada atrybut *type* określający, do jakiej domeny nazw należy wartość podawana w elemencie *<PartyId>*.

Jeżeli jakaś wartość atrybutu *type* odnosi się do jednostek, które posiadają wiele identyfikatorów, wartość atrybutu *type* musi być unikalna dla wszystkich wystąpień elementów *<PartyId>* zawartych w elementach *<From>* lub *<To>.*

Wartość atrybutu *type* musi być wzajemnie uzgodniona przez obie strony. Zaleca się, aby zarówno wartość elementu *<PartyId>* jak i jego atrybutu *type* były typu URI.

Ponadto zaleca się, aby wartości atrybutu *type* pochodziły z rejestrów EDIRA (ISO 6523), EDIFACT ISO 9735 lub ANSI ASC X12 I05.

Jeśli atrybut *type* elementu *<PartyId>* nie występuje, wówczas zawartość elementu *<PartyId>* musi być typu URI, w przeciwnym razie *ReceivingMSH* powinien zgłosić błąd podając w atrybutach elementu *<Error>* wartości:

1. *errorCode*="*Inconsistent*",
2. *severity*="*Error*".

Element *<Role>* określa rolę, w jakiej może wystąpić nadawca lub odbiorca przesyłki ebXML. Dopuszczalne wartości elementu, które zgodnie z zaleceniami powinny być typu URI, muszą być określone w CPA..

### MessageData

Element złożony *<MessageData>* zawiera elementy jednoznacznie identyfikujące przesyłkę ebXML.

Element *<MessageId>* zawiera unikalny identyfikator przesyłki.

Element *<Timestamp>* zawiera czas utworzenia nagłówka przesyłki ebXML. Czas musi być wyrażony jako UTC (w wartości elementu może być pominięte wskazanie na UTC).

Element *<RefToMessageId>* jeśli występuje, musi zawierać wartość elementu *<MessageId>* do której odnosi się ta przesyłka. Element nie może wystąpić, jeśli nie ma wcześniejszych przesyłek, do których można się odnieść. Dla przesyłek informujących o wykrytych błędach wystąpienie elementu *<RefToMessageId>* jest wymagane a jego wartość musi być wartością elementu *<MessageId>* z błędnej przesyłki.

Element *<TimeToLive>,* jeśli występuje wskazuje datę i czas, wyrażony jako UTC, do którego przesyłka powinna być dostarczona do *ToPartyMSH*.

Czas ważności przesyłki jest przekroczony, jeśli czas odbioru przesyłki przez *ReceivingMSH* zgodny z UTC jest większy od wartości elementu *<TimeToLive>* przesyłki.

Jeśli *ToPartyMSH* otrzyma przesyłkę po czasie ważności przesyłki powinien przesłać do *SendingMSH* *Komunikat o błędzie* z następującymi wartościami atrybutów elementu *<Error>:*

*errorCode*="*TimeToLiveExpired*",

*severity*="*Error*".

Więcej informacji na temat elementu *<TimeToLive>* w opisie parametrów protokołu *reliable messaging (*patrz *pkt Wymiana danych zgodna z protokołem reliable messaging → Parametry protokołu reliable messaging)* .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| MessageId | Wymagany | Element | xsd:string | Unikalny globalnie identyfikator przesyłki. |  |
| Timestamp | Wymagany | Element | xsd:dateTime | Czas wygenerowania nagłówka koperty ebXML. |  |
| RefToMessageId | Opcjonalny | Element | xsd:string | Wartość MessageId wcześniej wysłanej przesyłki, której dotyczy bieżąca przesyłka. |  |
| TimeToLive | Opcjonalny | Element | xsd:dateTime | Czas ważności przesyłki. |  |

### SyncReply

Element *<SyncReply>* może być użyty w każdej przesyłce wysyłanej przy użyciu synchronicznego protokołu transportowego np. HTTP, jeśli nadawca przesyłki żąda, aby odpowiedź była przesłana w ramach tego samego połączenia.

Wystąpienie tego elementu wskazuje wszystkim kolejnym węzłom MSH uczestniczącym w wymianie (*IntermediateMSH* i/lub *ToPartyMSH*), że oczekiwana jest synchroniczna odpowiedź i z tego względu konieczne jest utrzymanie otwartego połączenia (atrybut *SOAP:actor* ma wartość wskazującą na *NextMSH*).

Po odbiorze przesyłki z elementem *<SyncReply>* *ToPartyMSH* generuje odpowiedź, której rodzaj jest określony parametrem *syncReplyMode*  określonym w CPA.

Użycie elementu *<SyncReply>* nie jest dopuszczalne, jeśli wartość *syncReplyMode* = "*none*"

W tym wypadku *ReceivingMSH* powinno zgłosić błąd przekazując *Komunikat o błędzie* z następującymi wartościami atrybutów elementu *<Error>*:

1. *errorCode*="*Inconsistent*",
2. *severity*="*Error*".

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| gr. atryb. Header |  | Grupa atrybutów |  |  |  |
| SOAP:actor | Wymagany | Atrybut | xsd:anyURI |  | <http://schemas.xmlsoap.org/soap/actor/next> |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### MessageOrder

Element złożony *<MessageOrder>* jest wykorzystywany, jeśli w MSH jest zaimplementowany moduł obsługi sekwencji przesyłek (patrz pkt *Moduł obsługi sekwencji przesyłek*). Wystąpienie tego elementu w przesyłce wskazuje, że stanowi ona kolejną przesyłkę w serii składającej się na daną konwersację.

W podelemencie *<SequenceNumber>* przekazywany jest numer sekwencyjny przesyłki wskazujący na kolejność, w jakiej *ReceivingMSH* musi przetworzyć przesyłki.

W ramach danej konwersacji, określonej poprzez identyfikator przekazywany w elemencie *<ConversationId>*, każda ze stron zaangażowana w dialog stosuje unikalne numery sekwencyjne dla każdej z wysyłanych przez daną stronę przesyłek (chodzi tu o przesyłki z dokumentami biznesowymi, nie o *Komunikaty potwierdzające*). Oznacza to, że *<SequenceNumber>* jest unikalny w ramach *<ConversationId>* i *FromPartyMSH*.

Wartość elementu *<SequenceNumber>* mieści się w przedziale "*0-99999999*". W pierwszej przesyłce przesłanej w ramach danej konwersacji wartość jest ustawiona na zero, a następnie zwiększana o 1 dla każdej kolejnej przesyłki. Po osiągnięciu wartości równej "*99999999*" numerowanie zaczyna się od wartości "0".

W elemencie *<SequenceNumber>* przekazywana jest wartość równa "0" w przypadku:

1. pierwszej przesyłki wysyłanej przez *SendingMSH* w ramach konwersacji.
2. pierwszej przesyłki po wyzerowaniu numeru sekwencyjnego przez *SendingMSH*.
3. pierwszej przesyłki po przekroczeniu limitu (po osiągnięciu wartości "*99999999*").

Element *<SequenceNumber>* posiada atrybut *status* typu wyliczeniowego, który może przyjmować wartości:

1. "*Reset*" - jeśli wartość w elemencie jest równa 0 dla przypadków 1 lub 2,
2. "*Continue*" (wartość domyślna) – jeśli przekazywana przesyłka jest kolejna w danej sekwencji; wartość ta występuje, jeśli w wartość elementu jest równa 0 dla przypadku 3 (przekroczenie licznika).

*SendingMSH* zanim wyzeruje wartość elementu *<SequenceNumber>* w ramach danej konwersacji (po przepełnieniu licznika)musi poczekać aż otrzyma potwierdzenia dla wszystkich poprzednio wysłanych przesyłek.

Zalecane jest, aby pierwszy komunikat w nowej sekwencji z danym *ConversationId* zawierał element <*SequenceNumber>* (o wartości "0") z atrybutem *status* o wartości *Reset.*

Jeśli został osiągnięty limit ilości przesyłek, wówczas *ReceivingMSH* musi zasygnalizować *SendingMSH* błąd dostarczenia poprzez ustawienie wartości atrybutów elementu *<Error>* w sposób następujący:

1. *errorCode=*"*DeliveryFailure*",
2. *severity*="*Error*".

Element *<MessageOrder>:*

1. musi wystąpić razem z elementem *<DuplicateElimination>* (obsługa sekwencji przesyłek może być stosowana tylko wtedy, jeśli są one przekazywane w trybie *reliable messaging*),
2. nie może wystąpić z elementem *<SyncReply>* wskazującym na żądanie przekazania synchronicznej odpowiedzi.

W przypadku, jeśli jeden z tych warunków nie zostanie zachowany *ReceivingMSH* musi zgłosić błąd przesyłając *Komunikat o błędzie* z następującymi wartościami atrybutów elementu *<Error>:*

1. *errorCode =* "*Inconsistent*"*,*
2. *severity =* "*Error*"*.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| gr. atryb. Header |  | Grupa atrybutów |  |  |  |
| SequenceNumber | Wymagany | Element | xsd:nonNegativeInteger | Numer kolejny |  |
| status | Opcjonalny | Atrybut | NMTOKEN | Typ wyliczeniowy na bazie NMTOKEN | "Reset", "Continue" (wartość domyślna) |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### AckRequested

Element *<AckRequested>* wskazuje *ReceivingMSH* występującemu w roli określonej przez atrybut *SOAP:actor*, że oczekiwany jest *Komunikat potwierdzający*. Element informuje także za pomocą atrybutu *signed* czy przesyłka ta ma być podpisana przez *ReceivingMSH*.

Element *<AckRequested>* nie może wystąpić w przesyłce:

1. zawierającej tylko element *<Acknowledgment>* (brak *payload-ów*) – oznacza to, że w odpowiedzi na *Komunikat potwierdzający* odbiór wcześniejszej przesyłki nie jest przekazywane potwierdzenie,
2. stanowiącej *Komunikat o błędzie*.

Przesyłka ebXML może posiadać 0, 1 lub 2 wystąpienia elementu *<AckRequested>.*

Pozwala to na przekazanie informacji o żądaniu potwierdzenia zarówno do:

1. kolejnego *NextMSH* uczestniczącego w przekazaniu przesyłki do ostatecznego odbiorcy *ToPartyMSH*,
2. MSH ostatecznego odbiorcy (*ToPartyMSH*).

Zakłada się, że informacja identyfikująca MSH występujące w danej roli występuje co najwyżej raz.

Adresat informacji, dotyczącej żądania potwierdzenia, jest wskazywany w atrybucie *SOAP:actor*, zawierającym w kolejnych wystąpieniach <*AckRequested*> odpowiednio wartości URI oznaczające:

1. *NextMSH*,
2. *ToPartyMSH* .

Atrybut *SOAP:actor* jest opcjonalny, w przypadku jego braku przyjmowana jest wartość domyślna wskazująca na *ToPartyMSH*.

W przypadku wymiany w modelu *Single-hop* powyższe wartości można traktować jako zamiennie identyfikujące ostatecznego odbiorcę. Jeżeli jednak *ReceivingMSH* odbierze przesyłkę zawierającą dwa elementy *<AckRequested>* (jeden wskazujący na *NextMSH*, drugi na *ToPartyMSH)* zobowiązane jest przekazać w odpowiedzi dwie przesyłki potwierdzające lub jedną przesyłkę z dwoma elementami *<Acknowledgement>.*

Różne wskazania na określone MSH jako zobowiązane do przekazywania potwierdzenia wpływa na różne warianty implementacji *reliable messaging* – patrz tabela w pkt. *Kombinacje reliable messaging*

W zależności od wartości obowiązkowego atrybutu *signed (*"*true*" lub "*false*"*) Komunikat potwierdzający* powinien zawierać element *<Signature>* z podpisem *ReceivingMSH*.

Przed ustaleniem wartości atrybutu *signed* *SendingMSH* powinno sprawdzić czy *ReceivingMSH* obsługuje *Komunikaty potwierdzające* żądanego typu.

Kiedy *ReceivingMSH* odbiera przesyłkę z atrybutem *signed* ustawionym na wartość "*true*" lub "*false*" sprawdza czy jest w stanie obsłużyć żądany typ komunikatu:

1. jeśli tak, musi przesłać do *SendingMSH* przesyłkę zawierająca element *<Acknowledgment>*,
2. jeśli nie, musi zgłosić *SendingMSH* błąd za pomocą *Komunikatu o błędzie* z następującymi wartościami atrybutów elementu *<Error>*:
   1. *errorCode="Inconsistent",*
   2. *severity="Error".*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| gr. atryb. Header |  | Grupa atrybutów |  |  |  |
| SOAP:actor | Opcjonalny | Atrybut | xsd:anyURI | Wskazanie na adresata informacji, dotyczącej żądania przesłania potwierdzenia | urn:oasis:names:tc:ebxml-msg:actor:nextMSH  urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH |
| Signed | Wymagany | Atrybut | xsd:boolean | Informuje czy przesyłka potwierdzająca ma być podpisana. | true, false |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### Acknowledgment

Element *<Acknowledgment>* występujący w przesyłce wskazuje danemu MSH, ze inne MSH odebrało przesyłkę, której *MessageId* jest podany w podelemencie *<RefToMessageld>*.

Przesyłka zawierająca element *<Acknowledgment>* realizuje funkcje *Komunikatu potwierdzającego* używanego w celu poinformowania *SendingMSH* przez *ReceivingMSH* o fakcie otrzymania przesyłki i przekazaniu jej do dalszego przetwarzania:

1. przez ostatecznego odbiorcę (*ToPartyMSH*) – do aplikacji,
2. przez pośrednika (*IntermediateMSH*) w modelu *Multi-hop* – do *NextMSH.*

Przesyłka potwierdzająca może posiadać 0, 1 lub 2 wystąpienia elementu *<AckRequested>,* w zależności od ilości wystąpień *<AckRequsted>* w przesyłce potwierdzanej.

Element nie może występować w *Komunikacie o błędzie* (zawierającym element *<Error>*).

Powtarzalny element *<ds:Reference>* jest wykorzystywany, jeśli w odpowiedzi jest przekazywany *Komunikat potwierdzający* z podpisem cyfrowy *ReceivingMSH* (dołączanym jeśli potwierdzana przesyłka zawiera element *<AckRequested>* z wartością atrybutu *signed* równą "*true*").

Przekazywane są w nim elementy *<ds:Reference>* pobrane z przesyłki potwierdzanej. Referencje te zawierają skróty elementów tej przesyłki, które były objęte podpisem *SendingMSH*.

Zamieszczenie skrótów z otrzymanej przesyłki w *Komunikacie potwierdzającym* pozwala na uwzględnienie ich w podpisie *Komunikatu potwierdzającego*, który w takim wypadku realizuje funkcje niezaprzeczalności odbioru (patrz opis w pkt. *Podpis w komunikacie potwierdzającym*). MSH po otrzymaniu takiego komunikatu, może sprawdzić Referencje (skróty) i uzyskać pewność, że przesyłka dotarła do adresata w postaci nienaruszonej.

W przypadku, jeśli potwierdzana przesyłka nie zawiera podpisu (brak *<ds:Reference>* w oryginalnej przesyłce) dane dla podelement *<ds:Reference>* elementu *<Acknowledgement>* są generowane z danych potwierdzanej przesyłki.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| gr. tryb. Header |  | Grupa atrybutów |  |  |  |
| SOAP:actor | Opcjonalny | Atrybut | xsd:anyURI | Wartość atrybutu określająca MSH przesyłające potwierdzenie - powinna być zgodna z wartością odpowiadającą wartości elementu *<AckRequested>* potwierdzanej przesyłki. | urn:oasis:names:tc:ebxml-msg:actor:nextMSH  urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH |
| Timestamp | Wymagany | Atrybut | xsd:boolean | Czas odbioru potwierdzanej przesyłki przez MSH generujące przesyłkę potwierdzającą. |  |
| RefToMessageId | Wymagany | Element | xsd:string | *MessageId* potwierdzanej przesyłki. |  |
| [From](#FromTo) | Opcjonalny | Element złożony |  | Zawiera identyfikator strony generującej przesyłkę potwierdzającą.  Element opcjonalny pomijany, jeśli strona ta jest zidentyfikowana w *<From>* na poziomie *<MessageHeader>* |  |
| [ds:Reference](#Reference) | Opcjonalny  [0..n] | Element złożony |  | Wykorzystywane, jeśli jest przekazywane podpisywane potwierdzenie odbioru przesyłki  Struktura elementu jest zdefiniowana w specyfikacji XML *Signature* (patrz pkt *ds:Reference*)  Zawartość pobrana z przesyłki potwierdzanej z *<ds:Reference >* elementu *<ds:Signature>* (l*ub wygenerowana - patrz opis powyżej*) |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### ErrorList

Wystąpienie złożonego elementu *<ErrorList>* w nagłówku koperty ebXML informuje o pojawieniu się błędu w przesyłce, której dotyczy bieżąca przesyłka. Przesyłka zawierająca błąd jest identyfikowana za pomocą elementu *<RefToMessageId*> zawartego w elemencie *<MessageHeader>*.

Atrybut *highestSeverity* zawiera informacje o maksymalnym poziomie błędów zgłoszonych w elementach *<Error>*, będących elementami potomnymi dla *<ErrorList>.* Jeśli atrybut *severity* któregokolwiek z elementów *<Error>* posiada wartość "*Error*", wówczas atrybut *highestSeverity* również musi posiadać wartość "*Error*". W przeciwnym przypadku *highestSeverity* musi posiadać wartość "*Warning*".

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| gr. atryb. Header |  | Grupa atrybutów |  |  |  |
| highestSeverity | Wymagany | Atrybut | NMTOKEN | Typ wyliczeniowy na bazie NMTOKEN. Atrybut informujący o ostrości błędu. | Warning, Error |
| [Error](#Error) | Wymagany  [1..n] | Element złożony |  | Zawiera informacje o rodzaju błędu, jego poziomie, lokalizacji i opisie. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### Error

Element *<Error>* może wystąpić raz lub więcej w zależności od ilości błędów zawartych w przesyłce. Jego atrybuty i element *<Description>* zawierają bardziej szczegółowe informacje odnoszące się do powstałych błędów.

Jeśli błąd występuje w elemencie będącym ebXML-owym rozszerzeniem koperty SOAP, atrybut *id* elementu powodującego błąd może być użyty w celu wskazania lokalizacji błędu (atrybut *Location*).

Atrybut *codeContext* identyfikuje przestrzeń nazw lub schemat określające wartości atrybutu *errorCode.* Wartość atrybutu musi być typu URI. Domyślną wartością atrybutu jest „*urn:oasis:names:tc:ebxml-msg:service:error”,* wskazująca że kody raportowanych błędów zgodne są ze Specyfikacją. Jeśli atrybut nie posiada wartości domyślnej oznacza to, że w danej implementacji MSH używane są własne kody błędów.

Użycie innej wartości atrybutu *codeContext* niż wartość domyślna nie jest zalecane.

Nie jest zalecane również stosowanie własnych wartości kodów dla atrybutu *errorCode*, jeśli posiadają one takie samo lub podobne znaczenie jak zdefiniowane w tej Specyfikacji.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| Id | Opcjonalny | Atrybut | Xsd ID | Identyfikuje element *<Error>* | Zgodna z typem prostym ID W3C Schema |
| CodeContext | Opcjonalny | Atrybut | xsd:anyURI |  | Wartość domyślna  “urn:oasis:names:tc:ebxml-msg:service:errors” |
| ErrorCode | Wymagany | Atrybut | xsd:string |  |  |
| Severity | Wymagany | Atrybut | NMTOKEN | Typ wyliczeniowy na bazie NMTOKEN | Warning, Error |
| Location | Opcjonalny | Atrybut | xsd:string | Lokalizacja błędu w elemencie koperty SOAP/ebXML przesylki z błędem |  |
| Dowolny atrybut | Opcjonalny | Atrybut |  |  |  |
| Description | Opcjonalny | Element | xsd:string | Opis błędu w języku określonym przez atrybut *xml:lang* |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

Poniższa tabelki zawierają wartości atrybutu *errorCode* wraz z ich opisem zdefiniowane dla domyślnej przestrzeni nazw atrybutu *codeContext.*

Wartości atrybutu *errorCode* jakie mogą być zgłoszone w odniesieniu do elementów ebXML.

| Wartość atrybutu errorCode | Opis |
| --- | --- |
| ValueNotRecognized | Zawartość elementu lub wartość atrybutu nie jest rozpoznawana.  Chociaż dokument jest poprawnie sformatowany (*well formed*) i jego walidacja nie zgłasza błędów (*valid*), dany element lub atrybut zawiera wartość, która nie jest rozpoznana i z tego powodu nie może być użyta w *ebXML Message Service.* |
| NotSupported | Element lub atrybut nie jest obsługiwany.  Chociaż dokument jest poprawnie sformatowany (well formed) i jego walidacja nie zgłasza błędów (valid), występujący moduł jest zgodny z zasadami specyfikacji ebXML MSS, nie jest jednak obsługiwany przez daną implementację MSH przetwarzającą dokument (patrz pkt *Dodatkowe moduły funkcjonalne MSH*) |
| Inconsistent | Zawartość elementu lub wartość atrybutu jest niespójna z innymi elementami lub atrybutami. Chociaż dokument jest poprawnie sformatowany (well formed) i jego walidacja nie zgłasza błędów (valid), spełnia także wymogi specyfikacji ebXML MSS zawartość elementu lub atrybutu jest niespójna z zawartością innych elementów lub ich atrybutów. |
| OtherXML | Inny błąd dotyczący zawartości elementu lub wartości atrybutu.  Chociaż dokument jest poprawnie sformatowany (well formed) i jego walidacja nie zgłasza błędów (valid), zawartość elementu lub atrybutu nie jest zgodna z zasadami i ograniczeniami specyfikacji ebXML MSS oraz nie jest identyfikowana za pomocą istniejących wartości atrybutu *errorCode.* Zawartość elementu *<Error>* powinna być użyta do opisu natury błędu. |

Nie xml-owe wartości atrybutu *errorCode*.

|  |  |
| --- | --- |
| Wartość atrybutu errorCode | Opis |
| DeliveryFailure | Nie powiodło się dostarczenie przesyłki.  Otrzymana przesyłka (prawdopodobnie lub na pewno) nie mogła być przesłana dalej do następnego odbiorcy (*NextMSH*).  *Uwaga:*  Jeśli atrybut *severity* posiada wartość ”Warning” wówczas istnieje małe prawdopodobieństwo, że przesyłka została dostarczona. |
| TimeToLiveExpired | Dopuszczalny czas oczekiwania na przesyłkę minął.  Przesyłka została odebrana po terminie oczekiwania określonym w elemencie *<TimeToLiveExpired> będącym elementem potomnym <MessageHeader>.* |
| SecurityFailure | Sprawdzenie zabezpieczeń zastosowanych w przesyłce nie powiodło się.  Weryfikacja podpisów lub uwierzytelnienie, autoryzacja nadawcy przesyłki nie powiodła się. |
| MimeProblem | Błąd adresu URI, identyfikującego załącznik.  Zgłaszany jeśli atrybut *xlink:href* , który zawiera URI nie związane z dołączonym załącznikiem lub zasób reprezentowany przez URI nie jest dostępny. |
| Unknown | Nieznany błąd.  Oznacza, że błąd jaki się pojawił nie jest identyfikowany wprost przez żaden z błędów opisanych wyżej. Zawartość elementu *<Error>* powinna być użyta do identyfikacji natury błędu. |

Wymagany atrybut *severity* identyfikuje poziom błędu. Wartość "*Warning*" oznacza, że inne przesyłki składające się na konwersację mogą być generowane w normalny sposób pomimo powstałego problemu. Wartość "*Error*" oznacza, że w przesyłce pojawił się błąd uniemożliwiający jej dalsze przetwarzanie. Odpowiedni stan powinien być zgłoszony *Aplikacji biznesowej*.

Wartość atrybutu *location* wskazuje na tą część przesyłki, w której wystąpił błąd. Jeśli błąd występuje w elemencie ebXML i przesyłka zawierająca ten element jest poprawnie sformatowany (*well formed*) wówczas zawartość atrybutu *location* musi być określona za pomocą wskaźników *XPointer*.

Jeśli błąd jest związany z ebXML-owym załącznikiem (*Payload Container*) wówczas atrybut *location* zawiera *content-id* części MIME, w której wystąpił błąd opisany przy pomocy schematu URI – "*cid*".

Zawartość elementu *<Description>* dostarcza opisu błędu w języku określonym przez atrybut *xml:lang.* Parser XML-owy lub oprogramowanie walidujące przesyłkę zazwyczaj generuje tekst *Komunikatu o błędzie*, którego zawartość jest określona przez sprzedawcę / projektanta oprogramowania generującego element *<Error>.*

### ds:Signature

Element <ds:*Signature>* jest elementem głównym podpisu elektronicznego w formacie XML Signature, jego zawartość obejmuje podpis cyfrowy, referencje do podpisanych obiektów, dane kryptograficzne pozwalające na weryfikację podpisu cyfrowego oraz wszelkie inne dane wynikające z implementacji podpisu cyfrowego.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:Id | Opcjonalny | Atrybut | xsd:ID | Identyfikator elementu <*ds:Signature*>. |  |
| [ds:SignedInfo](#SignedInfo) | Wymagany | Element złożony |  | Element podpisywany cyfrowo. |  |
| ds:Signature Value | Wymagany | Element | xsd: base64-Binary | Element zawierający podpis cyfrowy złożony pod elementem <*ds:SignedInfo*>. |  |
| ds:KeyInfo | Opcjonalny | Element rozszerzalny |  | Dane dotyczące klucza służącego do weryfikacji podpisu cyfrowego. |  |
| ds:Object | Opcjonalny  [0..n] | Element rozszerzalny |  | Element pozwalający na przekazywanie dodatkowych danych, parametrów w ramach sygnatury XML. |  |

W ramach elementu *<ds:Signature>* możemy zdefiniować dodatkowo atrybuty:

1. kwalifikator przestrzeni nazw XML Signature o wartości [http://www.w3.org/2000/09/xmldsig#](http://www.w3.org/2000/09/xmldsig) wraz z definicją prefiksu,
2. opcjonalną lokalizację URI pliku W3C Schema definiującego struktury podpisu XMLSignature np. <http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd>.

Element potomny *<ds:Object>* jest elementem rozszerzalnym, pozwalającym na przekazanie dodatkowych niezdefiniowanych w *<ds:Signature>* danych wynikających z potrzeb konkretnej implementacji. Element ten jest wykorzystywany także do przekazywania np. podpisu kwalifikowanego – patrz opis w pkt. *Podpis elektroniczny w przesyłce ebXML*.

## Elementy występujące w SOAP:Body

Element *<SOAP:Body>* jest drugim elementem potomnym elementu głównego *<SOAP:Envelope>* koperty ebXML.

Musi posiadać kwalifikator przestrzeni nazw o wartości "<http://schemas.xmlsoap.org/soap/envelope>".

Jest to element obowiązkowy w kopercie SOAP więc musi wystąpić w każdej przesyłce ebXML (w komunikatach, w których nie są wykorzystywane podelementy wymienione w poniższej tabeli występuje jako element pusty *<SOAP:Body/>*).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Nazwa*** | ***Status / Ilość powt.*** | ***Rodzaj*** | ***Typ*** | ***Opis*** | ***Wartości dopuszczalne*** |
| [Manifest](#Manifest) | Opcjonalny | Element złożony |  | Identyfikuje załączniki dołączone do przesyłki |  |
| [StatusRequest](#StatusRequest) | Opcjonalny | Element złożony |  | Identyfikuje wcześniejszą przesyłkę, której status jest żądany. |  |
| [StatusResponse](#StatusResponse) | Opcjonalny | Element złożony |  | Użyty do opisania statusu przetwarzanej przesyłki |  |

### 

### Manifest

Element złożony *<Manifest>* może wystąpić jako element potomny elementu *<SOAP:Body>*. Składa się z grupy atrybutów oraz jednego lub większej ilości wystąpień elementu *<Reference>* identyfikującego załączniki wchodzące w skład przesyłki. Załączniki mogą być częścią przesyłki ebXML (*MIME*) lub mogą być dostępne poprzez adres URL. Zaleca się, aby nie umieszczać ich w elemencie *<SOAP:Body>* koperty ebXML. Głównym zadaniem elementu *<Manifest>* jest:

1. ułatwienie obsługi przesyłki ebXML dzięki możliwości szybszej lokalizacji poszczególnych załączników z przesyłki ebXML,
2. pozostawienie aplikacji możliwości przetworzenia załączników bez konieczności ich parsowania.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| gr. atryb. Body |  | Grupa atrybutów |  |  |  |
| [Reference](#Reference) | Wymagany  [1..n] | Element złożony |  | Referencje do załączników |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### Reference

Referencje do załączonego w przesyłce ebXML dokumentu są podawane w formacie prostego linka (*simple link*) zgodnego z [XLINK]. XLINK w tym elemencie został zastosowany w celu zapewnienia jednolitego sposobu opisywania referencji.

Element *<Schema>* występuje, jeśli dokument w załączniku, którego dotyczą referencje ma strukturę zgodną jest z ustalonym schematem (zdefiniowanym w *XML Schema*, DTD, lub innym schematem np. bazodanowym).

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| Id | Opcjonalny | Atrybut | XML ID |  |  |
| xlink:type | Opcjonalny | Atrybut | NMTOKEN | Atrybut wskazujący na typ zdefiniowany w *XML schema* opisującej XLINK bazujący na typie NMTOKEN. | Wartość stała "simple" |
| xlink:href | Wymagany | Atrybut | anyURI | Wartością atrybutu jest URI załącznika, którego dotyczy element *<Reference>*. Powinna być zgodna z kryteriami określonymi w specyfikacji XLINK odnoszącymi się do *simple link*. |  |
| xlink:role | Opcjonalny | Atrybut | anyURI | Atrybut identyfikuje zasoby opisujące załącznik lub jego cel, przeznaczenie. Jeśli występuje powinien być typu URI zgodnego z XLINK. |  |
| Dowolny atrybut | Opcjonalny |  |  |  |  |
| Schema | Opcjonalny  [0..n] | Element pusty | Brak | Przenosi w atrybutach informacje dotyczące lokalizacji schematu dla załącznika |  |
| location | Wymagany | Atrybut | anyURI | Wymagane URI wskazujące na lokalizację schematu załącznika |  |
| version | Opcjonalny | Atrybut | xsd:string | identyfikator wersji schematu dokumentu z załącznika |  |
| Description | Opcjonalny  [0..n] | Element | xsd:string | Dodatkowy opis załącznika wskazanego w referencjach. |  |
| xml:lang | Wymagany | Atrybut | xsd:language | Atrybut elementu <*Description*> określający język użyty do opisu. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### StatusRequest

*<StatusRequest>,* opcjonalny podelement elementu *<SOAP:Body>,* wykorzystywany w *Komunikacie statusowym* *Message Status Request* (patrz pkt *Komunikat Message Status Request*).

Zawiera podelement <*RefToMessageId*>, w którym jest podawany identyfikator przesyłki, której status ma być przekazany w odpowiedzi na żądanie.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| gr. atryb. Body |  | Grupa atrybutów |  |  |  |
| RefToMessageId | Wymagany | Element | xsd:string | Zawiera MessageId przesyłki, której dotyczy zapytanie o status. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

W komunikacie zawierającym *<StatusRequest>* nie mogą występować elementy:

1. w *<SOAP:Body>*:
   1. *<Manifest>*,
   2. *<StatusResponse>*,
2. w *<SOAP:Header>*:
   1. *<ErrorList>*.

### StatusResponse

*<StatusResponse>,*  opcjonalny podelement elementu *<SOAP:Body>,*  wykorzystywany *w Komunikacie statusowym* *Message Status Response* (patrz pkt *Komunikat Message Status Response*).

Wymagany element *<RefToMessageId>* zawiera *<MessageId>* przesyłki, o której statusie się informuje.

Uwaga: podelement *<RefToMessageId>* występuje również w elemencie *<MessageData>* *Komunikatu statusowego* *Message Status Response*, zawiera on jednak wartość identyfikującą *Komunikat statusowy* *Message Status Request,* w odpowiedzi na który jest podawany status wcześniej przesłanej przesyłki (patrz przykład w pkt. *Komunikat Message Status Request*).

Element *<Timestamp>*, w którym jest podawany czas otrzymania przesyłki, o której statusie się informuje

nie może wystąpić, jeśli atrybut *massageStatus* posiada wartość "*NotRecognized*" (przesyłka nie rozpoznana) lub wartość "*UnAuthorized*" (komunikat żądania statusu *Message Status Request* nie został zautoryzowany).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| gr. Atryb. Body |  | Grupa atrybutów |  |  |  |
| messageStatus | Wymagany | Atrybut | NMTOKEN | Typ wyliczeniowy na bazie NMTOKEN. Określa status przesyłki, której dotyczy zapytanie o status. | UnAuthorized, NotRecognized, Received, Processed, Forwarded |
| RefToMessageId | Wymagany | Element | xsd:string | Zawiera MessageId przesyłki, której status jest podawany |  |
| Timestamp | Opcjonalny | Element | xsd:dateTime | Czas otrzymania przesyłki, o której informuje status. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

W komunikacie zawierającym *<StatusResponse>* nie mogą występować elementy:

1. w *<SOAP:Body>*:
   1. *<Manifest>*,
   2. *<StatusRequest>*,
2. w *<SOAP:Header>*:
   1. *<ErrorList>* z wartością atrybutu highestSeverity ustawioną na "*Error*".

## Elementy występujące w ds:Signature

### ds:SignedInfo

Element <ds:*SignedInfo>* jest podpisywany cyfrowo a uzyskany podpis jest umieszczany w elemencie *<ds:SignatureValue>* struktury głównej *<ds:Signature>*.

Zawartość elementu <ds:*SignedInfo>* obejmuje:

1. listę referencji, której poszczególne elementy (<*ds:Reference*>) określają lokalizację oraz skrót każdego dowiązanego obiektu,
2. identyfikatory zastosowanych algorytmów kryptograficznych, funkcji skrótu, metod kanonizacji XML.

Przed podpisaniem element *<ds:SignedInfo>* jest poddawany kanonizacji XML, przez którą należy rozumieć przekształcenie go do tzw. postaci normalnej.

Przejście do postaci normalnej wiąże się z:

1. zmianą strony kodowej zawartości na domyślny UTF-8, znaki końca linii są kodowane przez "#xA",
2. normalizacją atrybutów elementów (porządek alfabetyczny, wartość w podwójnym cudzysłowie),
3. zastąpieniem encji znakowych reprezentowaną wartością,
4. usunięciem deklaracji nagłówka XML, DTD lub XML Schema,
5. normalizacją tzw. białych znaków (spacje, tabulatory, końce linii) za wyjątkiem wartości elementów i atrybutów.

Specyfikacja *ebXML Message Service Specification* zaleca stosowanie algorytmu *Required Canonical XML* – identyfikowanego przez <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>.

Dokładne informacje o postaci normalnej, transformacjach kanonizujących XML i ich implementacji można znaleźć w specyfikacji *Canonical XML v1.0 W3C Recommendation*.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| ds:Id | Opcjonalny | Atrybut | xsd:ID | Identyfikator elementu <*ds:SignedInfo*>. |  |
| ds:CannonicalizationMethod | Wymagany | Element |  | Identyfikuje algorytm doprowadzający elementy XML do postaci normalnej (kanonicznej). |  |
| ds:Algorithm | Wymagany | Atrybut | xsd:any URI | Identyfikator algorytmu. | <http://www.w3.org/TR/2001/REC-xml-c14n-20010315> |
| [ds:Signature Method](#SignatureMethod) | Wymagany | Element złożony |  | Określa algorytmy użyte do generowania i weryfikacji podpisu cyfrowego umieszczonego w elemencie <*ds:SignatureValue*>. |  |
| [ds:Reference](#Reference) | Wymagany  [1..n] | Element złożony |  | Element określa lokalizację i skrót dla dowiązanego obiektu. |  |

### ds:SignatureMethod

Element *<ds:SignatureMethod>* pozwala na wyspecyfikowanie algorytmu kryptograficznego oraz funkcji skrótu używanych przy generowaniu i weryfikacji podpisu cyfrowego. Specyfikacja XML Signature dopuszcza stosowanie następujących algorytmów:

1. DSA z SHA1,
2. RSA z SHA1,
3. HMAC z SHA1.

W specyfikacji ebXML *Message Service Specification* zalecane jest stosowanie algorytmu DSA, którego obsługa powinna być zaimplementowana w oprogramowaniu zgodnym z tą specyfikacją.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| ds:Algorithm | Wymagany | Atrybut | xsd:any URI | Identyfikator algorytmu kryptograficznego i funkcji skrótu. | <http://www.w3.org/200/09/xmldsig#dsa-sha1>(DSA-SHA1)**,**  <http://www.w3.org/2000/09/xmldsig#rsa-sha1> (RSA-SHA1),  <http://www.w3.org/2000/09/xmldsig#hmac-sha1> (HMAC-SHA1). |
| ds: HMAC OutputLength | Opcjonalny | Element | xsd:integer | Zawiera dodatkowy parametr w przypadku zastosowania HMAC. |  |
| Dowolny element | Opcjonalny |  |  |  |  |

### ds:Reference

Element <ds:*Reference>* reprezentuje obiekt, który ma być podpisany cyfrowo w ramach struktury *<ds:SignedInfo>*. W elemencie *<ds:Reference>* wskazywana jest m.in. lokalizacja obiektu, skrót obiektu, parametry określające identyfikator funkcji skrótu dla dowiązanego obiektu oraz operacje, które zostały przeprowadzone przed wyliczeniem skrótu na dowiązanym obiekcie.

Lokalizacja dowiązanego obiektu podawana jest w opcjonalnym atrybucie *ds:URI*, który przyjmuje wartości zgodne ze specyfikacją URI. Zalecane jest stosowanie wyrażeń Xpath w przypadku odwoływania się do fragmentów dokumentów XML.

Przykładowe wartości atrybutu *ds*:*URI* opisujące różne lokalizacje obiektów:

1. "*http://example.com/document.xml*"

zewnętrzny, samodzielny dokument XML dostępny pod danym adresem http,

1. "*http://example.com/document.xml#element*"

węzeł *element* w zewnętrznym dokumencie XML,

1. "*cid://content\_id\_identyfikator*"

dołączony do przesyłki SOAP/ ebXML dokument biznesowy XML (*payload*) o nagłówku *Content-Id* (MIME) równym wartości atrybutu,

1. "" – pusta zawartość

wskazuje bieżącą kopertę SOAP/ebXML zawierającą XML Signature,

1. "*#SignedProperties*"

węzeł <*SignedProperties>* koperty SOAP/ebXML zawierającą XML Signature,

1. "*#xpointer(id(‘id’))*"

element identyfikowany przez atrybut id o wartości "id".

Jeżeli atrybut *ds*:*URI* nie występuje w znaczniku <ds:*Reference>* przyjmuje się, że lokalizacja dowiązanego obiektu jest znana MSH odbiorcy.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:Id | Opcjonalny | Atrybut | xsd:ID | Identyfikator elementu <*ds:Reference*>. |  |
| ds:URI | Opcjonalny | Atrybut | xsd:any URI | Lokalizator dowiązanego obiektu. |  |
| ds:Type | Opcjonalny | Atrybut | xsd:any URI | Identyfikator typu dowiązanego obiektu. | Np.  <http://www.w3.org/2000/09/xmldsig#Object> |
| [ds:Transforms](#Transform) | Opcjonalny | Element złożony |  | Lista transformacji dowiązanego obiektu do przeprowadzenia przed wyliczeniem wartości funkcji skrótu. |  |
| ds:Transform | Wymagany  [1..n] | Element rozszerzalny |  | Pojedyncze przekształcenie. |  |
| ds:DigestMethod | Wymagany | Element |  | Zawiera identyfikator algorytmu obliczania funkcji skrótu. |  |
| ds:Algorithm | Wymagany | Atrybut | xsd:any URI | Identyfikator algorytmu funkcji skrótu | <http://www.w3.org/2000/09/xmldsig#sha-1> |
| ds:DigestValue | Wymagany | Element | xsd: base64 Binary | Wartość obliczonej funkcji skrótu dowiązanego obiektu. |  |

### ds:Transform

Element <*Transform>* określa pojedyncze przekształcenie jakie należy przeprowadzić na dowiązanym obiekcie w formacie XML przed wyliczeniem wartości funkcji skrótu.

Lista przekształceń może obejmować m.in.:

1. doprowadzenie dokumentu do postaci normalnej (kanonicznej),
2. zmianę struktury dokumentu XML (transformacje XSLT),
3. wyselekcjonowanie danych do podpisu z dokumentu XML (selekcje XPath),
4. usunięcie struktury <*ds:Signature*> ze struktury podpisywanego dokumentu XML.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| ds:Algorithm | Wymagany | Atrybut | xsd:any URI | Identyfikator algorytmu przekształcenia. | <http://www.w3.org/TR/2001/REC-xml-c14n-20010315> (kanonizacja),  <http://www.w3.org/TR/1999/REC-xslt-19991116> (XSLT),  <http://www.w3.org/TR/1999/REC-xpath-19991116> (Xpath),  [http://www.w3.org/TR/2000/09/xmldsig #enveloped-signature](http://www.w3.org/TR/2000/09/xmldsig%20#enveloped-signature) (usuniecie sygnatury) |
| ds:Xpath | Opcjonalny | Element | xsd:string | Zawiera definicję wyrażenia Xpath. |  |
| Dowolny element | Opcjonalny |  |  |  |  |

Element <ds:*Reference>* odnoszący się do elementu <*SOAP:Envelope>* powinien zawierać następującą listę transformacji:

1. usunięcie struktury <*ds:Signature*> ze struktury <*SOAP:Envelope>*,
2. usunięcie z <*SOAP:Envelope>* elementów wraz z elementami potomnymi zawierającymi atrybut *SOAP:actor* z wartością wskazującą na kolejny MSH (dotyczy przesyłania danych w modelu Multi-hop),
3. kanonizacja <*SOAP:Envelope>*.

Przykładowy element <ds:*Reference>* dla <*SOAP:Envelope>*.

|  |
| --- |
| <Reference URI=””>  <Transforms>  <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>  <Transform Algorithm="http://www.w3.org/TR/199/REC-xpath-19991116">  <Xpath>  not(ancestor-or self::node()[@SOAP:actot=&quot;urn:oasis:names:tc:ebxml-msg:actor:next&quot]|ancestor-or-self::node()[@SOAP:actot=&quot;http://schemas.xmlsoap.org/soap/actor/next&quot])  </Xpath>  </Transform>  <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>  </Transforms>  <DigestMethod Algorithm="http://www.w3.org/2000/09/  xmldsig#sha1"/>  <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>  </Reference> |

### ds:KeyInfo

Opcjonalny element <ds:*KeyInfo>* pozwala odbiorcy *ToPartyMSH* uzyskać informacje o kluczu niezbędnym do weryfikacji podpisu cyfrowego. W przypadku braku elementu *<ds:KeyInfo>* przyjmuje się, że wszelkie niezbędne informacje są znane stronie weryfikującej podpis.

Zdefiniowane poniżej w tabeli struktury podelementów *<ds:KeyInfo>* pozwalają na przekazywanie kluczy, nazw kluczy, certyfikatów oraz referencji do nich. Można deklarować kilka podelementów różnego typu np. <*ds:KeyValue*>/<*ds:RSAKeyValue*> i <*ds:X509Data*>, z zastrzeżeniem, że każdy z nich jest wykorzystywany w procesie weryfikacji podpisu cyfrowego przenoszonego w elemencie <*ds:SignatureValue*>. Dodatkowo możliwe jest wielokrotne deklarowanie podelementu tego samego typu np. <*ds:X509Data*>, z zastrzeżeniem jak wyżej.

Zarówno element <*ds:KeyInfo*> jaki i jego podelementy <*ds:KeyValue*>, <*ds:X509Data*>, <*ds:KeyValue*>, <*ds:PGPData*>, <*ds:SPKIdata*> są tzw. elementami rozszerzalnymi. Możliwe jest zatem definiowanie i implementacja własnych struktur rozszerzających funkcjonalność <*ds:KeyInfo*> lub jego podelementów. W ramach powyższego mechanizmu dopuszczalne jest zatem definiowanie struktur alternatywnych wobec już istniejących np. <*ds:PGPData*>, <*ds:SPKIData*>. W stosunku do podstawowej definicji W3C XML *Signature Schema* wszelkie nowe struktury muszą być zdefiniowane w oddzielnej przestrzeni nazw.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:Id | Opcjonalny | Atrybut | xsd:ID | Identyfikator elementu <*ds:KeyInfo*>. |  |
| ds:KeyName | Opcjonalny | Element | xsd:string | Nazwa lub identyfikator klucza lub pary kluczy, przyjęty przez stronę podpisującą dane *FromPartyMSH*. |  |
| ds:KeyValue | Opcjonalny  [0..n] | Element rozszerzalny |  | Zawiera dane dotyczące klucza publicznego. |  |
| ds:RetrievalMethod | Opcjonalny  [0..n] | Element rozszerzalny |  | Zawiera informacje o zewnętrznej lokalizacji klucza publicznego względem elementu <*ds:KeyInfo>*. |  |
| ds:X509Data | Opcjonalny  [0..n] | Element rozszerzalny |  | Zawiera dane dotyczące certyfikatów w formacie X509v3. |  |
| ds:PGPdata | Opcjonalny  [0..n] | Element rozszerzalny |  | Zawiera dane binarne dotyczące kluczy publicznych w formacie zgodnym z Open PGP Message Format. |  |
| ds:SPKIData | Opcjonalny  [0..n] | Element rozszerzalny |  | Zawiera dane udostępniane przez infrastrukturę SPKI w formacie binarnym (*S-expression)* . |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### ds:KeyValue

Element *<ds:KeyValue>* zawiera pojedynczy klucz publiczny, używany w procesie weryfikacji podpisu cyfrowego, w formacie zdefiniowanym w ramach podelementu <*ds:DSAKeyValue*> bądź <*ds:RSAKeyValue*>.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| [ds:DSAKeyValue](#DSAKeyValue) | Opcjonalny | Element złożony |  | Zawiera dane dotyczące klucza publicznego DSA. |  |
| [ds:RSAKeyValue](#RSAKeyValue) | Opcjonalny | Element złożony |  | Zawiera dane dotyczące klucza publicznego RSA. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### ds:DSAKeyValue

Element <ds:*DSAKeyValue>* zawiera parametry przekształcenia matematycznego, które są przekazywane w kluczu publicznym DSA. Dodatkowo element <ds:*DSAKeyValue>* może zawierać parametry przydatne m.in. ze względów wydajnościowych w procesie weryfikacji podpisu wykorzystującym algorytm DSA.

W przypadku odwołań z *<ds:RetrievalMethod>* element *<ds:DSAKeyValue>* jest identyfikowany przez URI [http://ww.w3.org/2000/09/xmldsig# DSAKeyValue](http://ww.w3.org/2000/09/xmldsig#RSAKeyValue).

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| ds:P | Opcjonalny | Element | xsd: base64 Binary | Wartość parametru P. |  |
| ds:Q | Opcjonalny | Element | xsd: base64 Binary | Wartość parametru Q. |  |
| ds:J | Opcjonalny | Element | xsd: base64 Binary | Wartość parametru J = (P-1)/Q. |  |
| ds:G | Opcjonalny | Element | xsd: base64 Binary | Wartość parametru G. |  |
| ds:Y | Wymagany | Element | xsd: base64 Binary | Wartość parametru Y. |  |
| ds:Seed | Opcjonalny | Element | xsd: base64 Binary | Parametr dla generatora liczb pierwszych. |  |
| ds:PgenCounter | Opcjonalny | Element | xsd: base64 Binary | Parametr dla generatora liczb pierwszych. |  |

### ds:RSAKeyValue

Element <ds:*RSAKeyValue>* zawiera parametry przekształcenia matematycznego, które są przekazywane w kluczu publicznym RSA.

W przypadku odwołań z *<ds:RetrievalMethod>* element *<ds:RSAKeyValue>* jest identyfikowany przez URI <http://ww.w3.org/2000/09/xmldsig#RSAKeyValue>.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:Modulus | Wymagany | Element | xsd: base64 Binary | Wartość modułu klucza publicznego RSA. |  |
| ds:Exponent | Wymagany | Element | xsd: base64 Binary | Wartość wykładnika klucza publicznego RSA. |  |

### ds:RetrievalMethod

Element <ds:RetrievalMethod> jest używany w sytuacji, kiedy należy zweryfikować podpis cyfrowy w oparciu o informacje dotyczące klucza publicznego, przechowywane w zewnętrznym, względem elementu <ds:*Signature>*, zasobie identyfikowanym poprzez atrybut *ds*:*URI*. Zasób ten może udostępniać dane kryptograficzne w formacie XML np. w postaci elementu <*ds:X509Data*>, bądź też w postaci binarnej akceptowalnej przez aplikację. Typ zdalnego zasobu, o ile nie jest znany, określa atrybut *ds*:*Type.*

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| ds:URI | Opcjonalny | Atrybut | xsd:any URI | Lokalizator dowiązanego zasobu kryptograficznego. |  |
| ds:Type | Opcjonalny | Atrybut | xsd:any URI | Typ dowiązanego zasobu kryptograficznego. | <http://www.w3.org/2000/09/xmldsig#DSAKeyValue>,  <http://www.w3.org/200/09/xmldisg#RSAKeyValue>,  <http://www.w3.org/200/09/xmldisg#X509Data>,  [http://www.w3.org/200/09/xmldisg#PGPData](http://www.w3.org/200/09/xmldisg#PPData).  <http://www.w3.org/200/09/xmldisg#SPKIData>,  <http://www.w3.org/200/09/xmldisg#rawX509Certificate> (binarny format certyfikatu X509 – ASN.1 DER) |
| [ds:Transforms](#Transform) | Opcjonalny | Element złożony |  | Lista przekształceń <ds:*Transform>*  zasobu XML przed pobraniem danych kryptograficznych, np. wyszukiwanie XPath. |  |

### ds:X509Data

Element <ds:*X509Data>* pozwala na przekazywanie danych dotyczących certyfikatów X509v3, które uczestniczą w procesie weryfikacji podpisu cyfrowego. Zawartość <*ds:X509Data*> może obejmować:

1. certyfikat z kluczem publicznym weryfikującym podpis (<*ds:X509Certificate*>),
2. wybrane informacje (<*ds:X509IssuerSerial*>, <*ds:X509SKI*>, <*ds:X509SubjectName*>) dotyczące certyfikatu,
3. certyfikaty CA wchodzące w skład łańcucha certyfikatów, który tworzy tzw. ścieżkę certyfikacji uwierzytelnienia dla certyfikatu (sekwencja <*ds:X509Certificate*>),
4. listę certyfikatów unieważnionych CRL (<*ds:X509CRL*>).

Struktura <*ds:X509Data*> musi zawierać przynajmniej jeden z podelementów: <*ds:X509IssuerSerial*>, <*ds:X509SubjectName*>, <*ds:X509SKI*>, <*ds:X509Certificate*>, <*ds:X509CRL*>.

W przypadku odwołań z *<ds:RetrievalMethod>* element *<ds:X509Data>* jest identyfikowany przez URI [http://ww.w3.org/2000/09/xmldsig#X509Data](http://ww.w3.org/2000/09/xmldsig#RSAKeyValue).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| [ds:X509Issuer Serial](#X509IssuerSerial) | Opcjonalny | Element złożony | xsd: base64 Binary | Element złożony zawierający dane identyfikujące wystawcę certyfikatu w postaci zgodnej ze specyfikacją [RFC2253]. |  |
| ds:X509SKI | Opcjonalny | Element | xsd: base64 Binary | Zawiera zakodowaną metodą base64 (bez użycia pośredniego kodowania DER) wartość identyfikatora klucza publicznego właściciela certyfikatu tzw. *X509v3 SubjectKeyIdentifier extension*. |  |
| ds:X509SubjectName | Opcjonalny | Element | xsd:String | Zawiera wyróżnioną nazwę właściciela certyfikatu. | Zgodna ze specyfikacją RC2253 (LDAP DN). |
| ds:X509Certificate | Opcjonalny  [0..n] | Element | xsd: base64 Binary | Zawiera certyfikat X509v3 w formacie ASN.1 DER, zakodowany base64. |  |
| ds:X509CRL | Opcjonalny | Element | xsd: base64 Binary | Zawiera listę unieważnionych certyfikatów, tzw. CRL. |  |
| Dowolny element | *Opcjonalny*  *[0..n]* |  |  |  |  |

### ds:X509IssuerSerial

Element <*ds:IssuerSerial*> zawiera wyróżnioną nazwę oraz numer seryjny wystawcy certyfikatu w postaci zgodnej ze specyfikacją [RFC2253].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:X509Issuer SerialName | Wymagany | Element | xsd:string | Wyróżniona nazwa wystawcy certyfikatu. |  |
| ds:X509Serial Number | Wymagany | Element | xsd:integer | Numer seryjny wystawcy certyfikatu. |  |

### ds:PGPData

Element <ds:*PGPData>* umożliwia przekazywanie danych dotyczących klucza publicznego w formacie zgodnym ze specyfikacją *Open PGP Message Format*.

W przypadku odwołań z *<ds:RetrievalMethod>* element *<ds:PGPData>* jest identyfikowany przez URI [http://ww.w3.org/2000/09/xmldsig#PGPData](http://ww.w3.org/2000/09/xmldsig#RSAKeyValue).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:PGPKeyId | Opcjonalny | Element | xsd: base64 Binary | Standardowy identyfikator klucza publicznego dostępnego w ramach systemu PGP. |  |
| ds:PGPKeyPacket | Opcjonalny | Element | xsd: base64 Binary | Zawiera tzw. *Key Material Packet* udostępniany w ramach systemu PGP. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### ds:SPKIData

Element <ds:*SPKIData>* umożliwia przekazywanie informacji dotyczących kluczy publicznych, certyfikatów lub innych danych w postaci binarnej udostępnianych w ramach infrastruktury SPKI - Simple Public Key Infrastructure.

W przypadku odwołań z *<ds:RetrievalMethod>* element *<ds:SPKIData>* jest identyfikowany przez URI [http://ww.w3.org/2000/09/xmldsig#SPKIData](http://ww.w3.org/2000/09/xmldsig#RSAKeyValue).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:SPKISexp | Wymagany | Element | xsd: base64 Binary | Zawiera dane dotyczące kluczy, certyfikatów w standardowej postaci *S-expressions* udostępnianej w SPKI. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

### ds:Object

Opcjonalny, rozszerzalny element <ds:*Object>* może zawierać dowolne dane w postaci XML, których typ i kodowanie można określić w atrybutach elementu. Zawartość może być zadeklarowana w oddzielnej przestrzeni nazw XML i być walidowana w oparciu o oddzielną schemę.

| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| --- | --- | --- | --- | --- | --- |
| ds:Id | Opcjonalny | Atrybut | xsd:ID | Identyfikator elementu. |  |
| ds:MimeType | Opcjonalny | Atrybut | xsd:String | Identyfikuje typ MIME zawartości elementu. |  |
| ds:Encoding | Opcjonalny | Atrybut | xsd:any URI | Identyfikuje sposób kodowania zawartości elementu. | np.:  <http://www.w3.org/200/09/xmldsig#baze64> (base64) |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

W ramach struktury <*ds:Object*> są deklarowane opcjonalne elementy złożone <*ds:Manifest*> oraz <*ds:SignatureProperties*>.

Element <*ds:Object*> pozwala na rozszerzenie funkcjonalności XML-Signature. Cecha ta została wykorzystana m.in. w specyfikacji ETSI TS 101903 definiującej XML-owy format podpisu kwalifikowanego.

### ds:Manifest

Opcjonalny element <ds:*Manifest>* zawiera listę złożoną z elementów <ds:*Reference>*,identyfikowaną przez atrybut *Id*. Do elementu <ds:*Manifest>* można odwoływać się z dowolnej sygnatury XML w ramach struktury <ds:*Reference>*, nadając dodatkowo atrybutowi *Type* wartość *http://www.w3.org/2000/09/xmldsig#Manifest*. Obiekty wchodzące w skład listy <*ds:Manifest*> mogą zatem pośrednio, za pośrednictwem wiążącego elementu *<ds:Reference>* wskazującego na listę *<ds:Manifest>*, wejść w skład struktury *<ds:SignedInfo>* i zostać podpisane cyfrowo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| ds:Id | Opcjonalny | Atrybut | xsd:ID | Identyfikator elementu. |  |
| ds:Reference | Wymagany  [1..n] | Element złożony |  | patrz definicja w pkt. *ds:Reference*. |  |

### ds:SignatureProperties

Opcjonalny, rozszerzalny element <ds:*SignatureProperties>* możezawierać dodatkowe informacje dotyczące procesu generowania podpisu oraz współpracy z urządzeniem kryptograficznym. W samodzielnie zdefiniowanych strukturach można umieścić informacje zawierające np.:

1. datę wygenerowanego podpisu,
2. numer urządzenia kryptograficznego użytego do generowania podpisu.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nazwa | Status / Ilość powt. | Rodzaj | Typ | Opis | Wartości dopuszczalne |
| Ds:Id | Opcjonalny | Atrybut | xsd:ID | Identyfikator elementu. |  |
| Ds:Target | Opcjonalny | Atrybut | xsd:any URI | Identyfikuje sygnaturę, przyjmuje wartość atrybutu *Id* elementu *<Signature>*. |  |
| Dowolny element | Opcjonalny  [0..n] |  |  |  |  |

# Zasady działania i funkcjonalność aplikacji Standardu RBE

Związek Banków Polskich Bazy i Systemy Bankowe Sp. z o.o.

Rada Bankowości Elektronicznej

Standard RBE strona - 92 / 106

*Część I: Specyfikacja warstwy komunikacyjnej – wersja 1.0*

## Diagram przepływu danych pomiędzy Aplikacjami biznesowymi partnerów za pośrednictwem MSH



W celu uproszczenia powyższy schemat nie zawiera szeregu czynności związanych z obsługą reliable messaging. Ich wyczerpujący opis został umieszczony w pkt. *Wymiana danych zgodna z reliable messaging*.

## Model obsługi podpisu

### Generowanie sygnatury XML

Proces generowania sygnatury XML w aplikacji implementującej MSH musi obejmować następujące etapy:

1. utworzenie elementu głównego <*ds:Signature*>,
2. przygotowanie elementów <ds:*Reference>* dla każdego obiektu (elementy koperty SOAP/ebXML, dokumenty biznesowe), który ma być podpisany cyfrowo:
   1. wykonanie założonych wcześniej przekształceń XML dla danych w formacie XML i umieszczenie ich w strukturze <*ds:Transforms*>,
   2. wyliczenie skrótu i umieszenie go w elemencie <*ds:DigestValue*>,
3. utworzenie elementu <*ds:SignedInfo*>,
4. włączenie listy elementów <*ds:Reference*> do struktury <*ds:SignedInfo*>,
5. kanonizacja elementu <*ds:SignedInfo*>,
6. podpisanie elementu <*ds:SignedInfo*>,
7. umieszczenie uzyskanego podpisu cyfrowego w elemencie <*ds:SignatureValue*>.

W wygenerowanej sygnaturze XML można zadeklarować opcjonalny element *<ds:KeyInfo>* z danymi kryptograficznymi umożliwiającymi weryfikację podpisu. Zawartość <*ds:KeyInfo*> jest uzależniona od przyjętej przez strony wymieniające dane polityki podpisu.

Dodatkowo dołączony może być także element <*ds:Object*>, którego zawartość może być dowolna, wynikająca z potrzeb konkretnej implementacji MSH.

Schemat: Generowanie sygnatury XML.



### Weryfikacja sygnatury XML

Proces weryfikacji sygnatury XML w aplikacji implementującej MSH obejmuje dwie zależne od siebie kontrole, których pozytywny wynik gwarantuje integralność podpisanych danych oraz uwierzytelnienie nadawcy przesyłki ebXML - *FromPartyMSH*.

Etapy weryfikacji sygnatury XML:

1. weryfikacja podpisu cyfrowego *<ds:SignatureValue>* złożonego pod elementem *<ds:SignedInfo>* w oparciu o przesłane w elemencie *<ds:KeyInfo>* lub znane z kontekstu aplikacji dane dotyczące klucza publicznego nadawcy przesyłki ebXML - *FromPartyMSH*,
2. weryfikacja poszczególnych elementów <ds:*Reference>* - sprawdzenie skrótu w podelementach *<ds:DigestValue>.*

Jeżeli do sygnatury XML dołączono element *<ds:KeyInfo>* z danymi w postaci certyfikatu X509v3 (element *<ds:X509Data>*), możliwa jest także dodatkowa weryfikacja tak dołączonego klucza publicznego w oparciu tzw. zaufaną trzecią stronę – podmiot świadczący usługi certyfikacyjne, który wydał i poświadczył dany certyfikat w ramach infrastruktury PKI. Pełna weryfikacja certyfikatu zawartego w *<ds:KeyInfo>* może obejmować:

1. sprawdzenie ścieżki certyfikacji CA,
2. sprawdzenie złożonego przez CA poświadczenia elektronicznego, które obejmuje dane zawarte w certyfikacie, m.in. także klucz publiczny służący do weryfikacji podpisu w sygnaturze XML,
3. sprawdzenie daty ważności certyfikatu,
4. sprawdzenie listy certyfikatów unieważnionych (CRL) udostępnianych przez CA.

W przypadku wystąpienia błędu w trakcie procesu weryfikacji podpisu cyfrowego w sygnaturze XML np. błędny skrót w elemencie *<ds:Reference>*, błędny podpis elementu *<ds:SignedInfo>*, unieważniony lub błędny certyfikat, MSH weryfikujące podpis musi wygenerować *Komunikat o błędzie* zawierający element *<Error>* z atrybutami *errorCode*=„*SecurityFailure*” oraz *severity*=„*Error*”.

Pozostałe błędy wynikające z kontroli struktury elementu *<ds:Signature>* lub z braku dostępu do podpisywanych obiektów należy obsługiwać zgodnie z regułami protokołu ebXML opisanymi w rozdziale *Moduł obsługi błędów*.

Schemat: Weryfikacja sygnatury XML



## Wymiana przesyłek ebXML przez HTTP

### Wysyłanie przesyłek ebXML przez HTTP

Zgodnie ze specyfikacją *ebXML MSS* protokół HTTP 1.1 może być wykorzystywany jako warstwa transportowa dla przesyłek ebXML.

Protokół HTTP udostępnia dwie metody wysłania zapytań: metodę GET i metodę POST. W celu przekazania w zapytaniu HTTP przesyłki ebXML stosowana jest tylko metoda POST.

W zapytaniu wysłanym metodą POST, w części identyfikującej zdalny zasób, możliwe jest zawarcie względnego adresu komponentu obsługującego zapytanie w MSH, np.: POST /servlet/ebxmlhandler HTTP/1.1

Przesyłka ebXML wysłana przez HTTP musi spełniać specyficzne dla HTTP ograniczenia kanonicznych form MIME, co zostało opisane w [RFC2616].

Niektóre z protokołów transportowych wymagają zakodowania danych przekazywanych w poszczególnych częściach przesyłki MIME do 7-bitowej reprezentacji. Informacja o zastosowanej metodzie kodowania jest odnotowywana w nagłówku MIME w polu *content-transfer-encoding*.

Protokół HTTP może przenosić 8-bitowe dane znakowe i binarne w związku z tym kodowanie danych występujących w przesyłce ebXML nie jest zalecane. Jednakże dodatkowe kodowanie danych, np. metodą base64, jest dopuszczalne (jest opcjonalne).

Zasady tworzenia wiadomości HTTP zawierającej przesyłkę ebXML.

W nagłówku wiadomości *HTTP Header* są podawane informacje, pobrane z nagłówka przesyłki MIME stanowiącej przesyłkę ebXML i dodatkowe wymagane przez protokół HTTP związane z obsługą MIME.

Nagłówek HTTP zawiera wszystkie pola z nagłówka MIME przesyłki ebXML, w tym:

* 1. pole *Content-Type* określające typ wiadomości razem ze wszystkimi parametrami podanymi w nagłówku przesyłki MIME, związanymi z przetwarzaniem MIME np. łańcuch graniczny "*boundary*".
     1. pole musi mieć wartość "*multipart/related*", która określa strukturę MIME zawartą w wiadomości HTTP
  2. pole *SOAPAction,* które jest polem obowiązkowym nagłówka HTTP i powinno przyjmować wartość "*ebXML*"
  3. wyjątek stanowi pole *MIME-Version*, które nie może wystąpić w nagłówku HTTP i wszystkie inne nie dopuszczalne w wersji HTTP 1.1
  4. dodatkowo na poziomie nagłówka HTTP nie powinno wystąpić *Content-Transfer-Encoding (*HTTP obsługuje 8-bitowe dane, czyli kodowanie nie jest zalecane).

Kolejne elementy struktury przesyłki MIME (ebXML) stanowiące kolejne części MIME/Multipart wraz z rozdzielającym je ciągiem znaków (*boundary*) są dołączane jako tzw. ciało wiadomości HTTP po deklaracji nagłówka HTTP.

Poniżej przedstawiono przykład zapytania HTTP z przesyłką ebXML.

|  |
| --- |
| POST /servlet/ebXMLhandler http/1.1  Host: [www.example2.com](http://www.example2.com)  SOAPAction: "ebXML"  Content-type: multipart/related; boundary="BoundarY"; type="text/xml"  start="<ebxhmheader111@example.com>"  --BoundarY  Content-ID: <ebxhmheader111@example.com>  <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd"  xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>">    <SOAP:Header>  <eb:MessageHeader eb:version=”2.0” SOAP:mustUnderstand=”1”>  <eb:From><eb:PartyId>urn:duns:123456789</eb:PartyId></eb:From>  <eb:To><eb:PartyId>urn: duns:912345678</eb:PartyId></eb:To>  <eb:CPAId>20001209-133003-28572</eb:CPAId>  <eb:ConversationId>20001209-133003-28572</eb:ConversationId>  <eb:Service>urn:services:SuplierOrderProcessing</eb:Service>  <eb:Action>NewOrder</eb:Action>  <eb:MessageData>  **<**eb:MessageId**>**20001209-133003-28572@example.com**<**/eb:MessageId**>**  <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>  </eb:MessageData>  </eb:MessageHeader>  </SOAP:Header>  <SOAP:Body>  <eb:Manifest eb:version="2.0">  <eb:Reference xlink:href=<cid:ebxmlpayload111@example.com>  xlink:role="XlinkRole" xlink:type="simple">  <eb:Description xml:lang-"en-US">Purchase Order 1</eb:Description>  </eb:Reference>  </eb:Manifest>  </SOAP:Body>  </SOAP:Envelope>  --BoundarY  Content-ID: [ebxmlpayload111@example.com](mailto:ebxmlpayload111@example.com)  Content-Type: text/xml  <?xml version="1.0" encoding="UTF-8"?>  <purchase\_order>  <po\_number>1</po\_number>  <part\_number>123</part\_number>  <price currency="USD">500.00</price>  </purchase\_order>  --BoundarY-- |

### Odbiór przesyłek ebXML przez HTTP

HTTP jest protokołem umożliwiającym synchroniczną wymianę danych w ramach pojedynczego połączenia TCP/IP. W przypadku poprawnego odbioru przesyłki ebXML z zapytania HTTP zwracana odpowiedź HTTP musi być zawsze opatrzona kodem 200 OK.

Jeżeli parametr *syncReplyMode* odbierającego MSH przyjmuje wartość inną niż "*none*" oraz w przesyłce ebXML z zapytania POST był zawarty element *<SyncReply>*, to przesyłka ebXML będąca odpowiedzią odbierającego MSH musi być zwracana jako ciało odpowiedzi HTTP w tym samym połączeniu TCP/IP co wcześniejsze zapytanie POST.

Jeżeli parametr "*syncReplyMode*"przyjmuje wartość "*none*" i w przesyłce ebXML nie ma elementu *<SyncReply>*, to jako odpowiedź w trybie synchronicznym jest zwracana wiadomość HTTP zawierająca puste ciało HTTP (tylko nagłówek HTTP). Przesyłka ebXML będąca odpowiedzią może być wówczas przekazana jako:

1. samodzielne wiadomość HTTP POST,
2. przesyłka pocztowa (SMTP).

Pozostałe kombinacje parametru *syncReplyMode*  oraz elementu <S*yncReply*> powodują wygenerowanie *Komunikatu o błędzie*, co zostało opisane w punkcie *SyncReply*.

Zasady formatowania odpowiedzi HTTP, m.in.: deklaracja nagłówka HTTP i sposób włączenia koperty MIME, są identyczne jak w przypadku wysyłania przesyłki ebXML metodą POST, co zostało opisane w poprzednim punkcie.

Poniżej przedstawiono przykład odpowiedzi HTTP z przesyłką ebXML będącej synchroniczną odpowiedzią MSH

|  |
| --- |
| HTTP/1.1 200 OK.  SOAPAction: "ebXML"  Content-Type:multipart/related; boundary="----=\_Part\_0\_26878.1004101997985"; type="text/xml"; start="<ebxhmheader112@example.com>"  ----=\_Part\_0\_26878.1004101997985  Content-ID: <ebxhmheader112@example.com>  Content-Type: text/xml  <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd"  xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>">    <SOAP:Header>  <eb:MessageHeader eb:version=”2.0” SOAP:mustUnderstand=”1”>  <eb:From><eb:PartyId>urn:duns:912345678</eb:PartyId></eb:From>  <eb:To><eb:PartyId> urn:duns:123456789</eb:PartyId></eb:To>  <eb:CPAId>20001209-133003-28572</eb:CPAId>  <eb:ConversationId>20001209-133003-28579</eb:ConversationId>  <eb:Service>urn:services:SuplierOrderProcessing</eb:Service>  <eb:Action>OrderConform</eb:Action>  <eb:MessageData>  **<**eb:MessageId**>**20001209-133003-28573@example.com**<**/eb:MessageId**>**  <eb:Timestamp>2001-02-15T11:12:22</eb:Timestamp>  <eb:RefToMessageId>20001209-133003-28572@example.com</eb:RefToMessageId>  </eb:MessageData>  </eb:MessageHeader>  </SOAP:Header>  <SOAP:Body>  <eb:Manifest eb:version="2.0">  <eb:Reference xlink:href=[cid:ebxmlpayload112@example.com](cid:ebxmlpayload111@example.com)  xlink:role="XlinkRole" xlink:type="simple">  <eb:Description xml:lang-"en-US">Conform Order 1</eb:Description>  </eb:Reference>  </eb:Manifest>  </SOAP:Body>  </SOAP:Envelope>  ----=\_Part\_0\_26878.1004101997985  Content-ID: [ebxmlpayload112@example.com](mailto:ebxmlpayload111@example.com)  Content-Type: text/xml  <?xml version="1.0" encoding="UTF-8"?>  <conform\_order>  <po\_number>1</po\_number>  <part\_number>123</part\_number>  </conform \_order>  ----=\_Part\_0\_26878.1004101997985 |

### Kody statusów odpowiedzi HTTP w obsłudze błędów SOAP/ebXML

W modelu obsługi błędów związanych z obsługą przesyłek ebXML w przypadku korzystania z protokołu HTTP należy zastosować kody statusów odpowiedzi HTTP.

Mogą być zwrócone następujące kody odpowiedzi HTTP informujące o błędzie:

1. kody z zakresu 3xx – błędy odwołania np. nieaktualne ścieżki dostępu do MSH,
2. kody z zakresu 4xx – błędy klienta np. brak uprawnień pozwalających na dostęp do wskazanego MSH,
3. kody z zakresu 5xx - tzw. wewnętrzne błędy serwera HTTP lub sygnalizowany synchronicznie błąd przetwarzania koperty SOAP/ebXML na poziomie elementów SOAP.

Odpowiedzi HTTP z kodami statusu 3xx, 4xx, 5xx nie zawierają ciała wiadomości HTTP (tylko nagłówek HTTP). Wyjątek stanowi odpowiedź HTTP raportująca wystąpienia błędu na poziomie elementów koperty SOAP, która jako ciało wiadomości HTTP powinna zawierać komunikat SOAP z elementem <*SOAP:Fault*>, zgodnie ze specyfikacją protokołu SOAP 1.1.

W przypadku wystąpienia błędów na poziomie ebXMLwygenerowany *Komunikat o błędzie* może być przesyłany synchronicznie w odpowiedzi HTTP z kodem statusu z zakresu 2xx (więcej informacji w punktach opisujących <*SyncReply>* oraz parametr s*yncReplyMode*).

### Kontrola dostępu na poziomie HTTP

Implementacje MSH wykorzystujące protokół HTTP mogą być zabezpieczone przed nieautoryzowanym dostępem za pomocą mechanizmów HTTP, opisanych w specyfikacji *HTTP Authentication: Basic and Digest Access Authenticati*on [RFC2617].

Implementacje MSH mogą stosować wszystkie schematy sterowania dostępem, łącznie z mechanizmami uwierzytelnienia, które zostały zdefiniowane w powyższej specyfikacji.

Implementacje MSH wykorzystujące mechanizmy uwierzytelnienia typu podstawowego (*Basic Authentication*) powinny dodatkowo wykorzystywać bezpieczny protokół sieciowy np. TLS (kompatybilny wstecznie z SSL).

### Poufność na poziomie protokołu komunikacyjnego

Implementacje MSHmogą stosować szyfrowanie w warstwie komunikacyjnej HTTP w celu zapewnienia poufności przesyłek ebXML oraz nagłówków HTTP. Dopuszczalne jest stosowanie bezpiecznych protokołów sieciowych (np. TLS, SSL), które oprócz zapewnienia poufności danych, pozwalają także na obustronne uwierzytelnienie oraz autoryzację stron uczestniczących w połączeniu HTTP.

Implementacje MSH wykorzystujące TLS muszą być zdolne do działania w trybie kompatybilności wstecznej z SSL.

Implementacje MSHmogą stosować dowolny, zgodny ze specyfikacją TLS, algorytm szyfrowania oraz rozmiar klucza. Wymagane jest jednak, aby rozmiary klucza oraz algorytmy kryptograficzne zapewniały wsteczną kompatybilność z SSL.

### Wymiana przesyłek ebXML przez SMTP

Specyfikacja protokołu komunikacyjnego *Simple Mail Transfer Protocol* (SMTP), powszechnie znana jako *Internet Electronic Mail (eMail)*, obejmuje m.in.:

1. *Multipurpose Internet Mail Extensions* [RFC2045], [RFC2046], [RFC2387] – standard formatowania komunikatów pocztowych,
2. *SMTP Service Extension for Authentication*  [RFC2554] – kontrola dostępu,
3. *SMTP Service Extension for Secure SMTP over TLS* [RFC2487] - metody wykorzystania bezpiecznego protokołu sieciowego TLS (wstecznie kompatybilnego z SSL).

Specyfikacja *eMail* wyróżnia dwa typ agentów:

1. *Message Transfer Agent* (MTA): program wysyłający i odbierający wiadomości pocztowe z innego agenta typu MTA, w imieniu programu klienta poczty elektronicznej (MUA). Przykładem MTA są *Microsoft Exchange Server* lub *Sendmail*.
2. *Mail User Agent* (MUA): program klienta poczty elektronicznej wykorzystywany do tworzenia elektronicznych wiadomości pocztowych i do komunikacji z MTA w celu wysyłania/odbioru wiadomości pocztowych. Przykładem MUA jest *Microsoft Outlook*.

W rozdziale tym zostaną opisane metody wymiany przesyłek ebXML w ramach protokołu SMTP z perspektywy MUA współpracującego z MSH.

### Wysyłanie przesyłek ebXML przez SMTP

Przesyłka ebXML sformatowana zgodnie z wymogami specyfikacji *ebXML Message Service,* opisanymi w rozdziale *Struktura przesyłki ebXML*, może być, jako struktura MIME, włączona do przesyłki pocztowej. Należy zwrócić szczególna uwagę na kodowanie poszczególnych fragmentów struktury MIME. Wiele typów danych, które chcemy wysłać pocztą elektroniczną, jest w postaci binarnej lub znakowej 8-bitowej. Danych w tej postaci nie można transmitować przez SMTP, który ogranicza wiadomości pocztowe do danych w 7-bitowym kodzie US-ASCII o długości linii do 1000 znaków wraz z separatorem linii CRLF. Jeżeli wysyłające MSH wie, że odbierający MTA lub jakikolwiek pośrednik MTA jest ograniczony do obsługi danych 7-bitowych, wtedy każdy fragment przesyłki ebXML w reprezentacji 8-bitowej lub binarnej musi zostać poddany kodowaniu zgodnie z regułami zawartymi w specyfikacji MIME (kodowanie base64). W przypadku, jeśli MSH wie, iż odbierający MTA i każdy pośrednik MTA potrafi obsłużyć dane 8-bitowe, wtedy nie ma potrzeby kodowania fragmentów przesyłki ebXML.

Dla dokumentów XML wchodzących w skład przesyłki ebXML stosuje się kodowanie UTF-8.

Zasady tworzenia nagłówka przesyłki pocztowej zawierającej przesyłkę ebXML.

Nagłówek wiadomości pocztowej zawiera:

1. wszystkie pola z nagłówka MIME przesyłki ebXML, w tym:
   * 1. pole *Content-Type* określające typ wiadomości razem ze wszystkimi parametrami podanymi w nagłówku przesyłki MIME, związanymi z przetwarzaniem MIME np. łańcuch graniczny podawany w atrybucie *boundary*.
        1. pole musi mieć wartość "*multipart/related*", która określa strukturę MIME zawartą w wiadomości HTTP
     2. pole *MIME-Version* z wartością *"1.0"*
     3. pole *SOAPAction,* z wartością "*ebXML*"
2. pole *To,* które musi zawierać adres e-mail odbierającego MSH*.*
3. pole *From* , które musi zawierać adres e-mail wysyłającego MSH*.*
4. pole *Date* zawierające datę utworzenia przesyłki pocztowej
5. inne pola wymagane przez SMTP związane z obsługą przesyłki ebXML stanowiącej strukturę MIME np. *Content-Transfer-Encoding*

Przykładowa zawartość wiadomości pocztowej z przesyłką ebXML:

|  |
| --- |
| From: [ebXMLhandler@example.com](mailto:ebXMLhandler@example.com)  To: [ebXMLhandler@example2.com](mailto:ebXMLhandler@example2.com)  Date: Thu, 08 Feb 2001 19:32:11 CST  MIME-Version: 1.0  SOAPAction: "ebXML"  Content-type: multipart/related; boundary="BoundarY"; type="text/xml"  start="<ebxhmheader111@example.com>"  --BoundarY  Content-ID: <ebxhmheader111@example.com>  Content-Type: text/xml  <?xml version=”1.0” encoding=”utf-8” ?>  <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/  http:/www.oasis-open.org/comittees/ebxml- msg/schema/envelope.xsd">  <SOAP:Header xmlns:eb="<http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd>">  <eb:MessageHeader eb:version=”2.0” SOAP:mustUnderstand=”1”>  <eb:From><eb:PartyId>urn:duns:123456789</eb:PartyId></eb:From>  <eb:To><eb:PartyId>urn: duns:912345678</eb:PartyId></eb:To>  <eb:CPAId>20001209-133003-28572</eb:CPAId>  <eb:ConversationId>20001209-133003-28572</eb:ConversationId>  <eb:Service>urn:services:SuplierOrderProcessing</eb:Service>  <eb:Action>NewOrder</eb:Action>  <eb:MessageData>  **<**eb:MessageId**>**20001209-133003-28572@example.com**<**/eb:MessageId**>**  <eb:Timestamp>2001-02-15T11:12:12<eb:Timestamp>  </eb:MessageData>  <eb:DuplicateElimination/>  </eb:MessageHeader>  </SOAP:Header>  <SOAP:Body xmlns:eb="<http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd>">  <eb:Manifest eb:version="2.0">  <eb:Reference xlink:href=<cid:ebxmlpayload111@example.com>  xlink:role="XlinkRole" xlink:type="simple">  <eb:Description xml:lang-"en-US">Purchase Order 1</eb:Description>  </eb:Reference>  </eb:Manifest>  </SOAP:Body>  </SOAP:Envelope>  --BoundarY  Content-ID: [ebxmlpayload111@example.com](mailto:ebxmlpayload111@example.com)  Content-Type: text/xml  <?xml version="1.0" encoding="UTF-8"?>  <purchase\_order>  <po\_number>1</po\_number>  <part\_number>123</part\_number>  <price currency="USD">500.00</price>  </purchase\_order>  --BoundarY-- |

### Pocztowy Komunikat odbioru

Przy wymianie danych przez SMTP wszystkie odpowiedzi ebXML, łącznie z *Komunikatami o błędach* i *Komunikatami potwierdzającymi*, są przekazywane w trybie asynchronicznym. Każda przesyłka pocztowa odpowiedzi musi być utworzona zgodnie z zasadami opisanymi w poprzednim rozdziale.

MSHmusibyć w stanie odebrać, za pośrednictwem MUA, wysłany przez MTA *Komunikat o błędzie*.

MSH, które nie może rozpoznać odebranej wiadomości jako poprawnej przesyłki ebXML lub *Komunikatu o błędzie*, powinno zatrzymać niezidentyfikowaną przesyłkę w folderze *dead mail*.

MSH powinno umieścić stosowną informację o stanie każdej odebranej przesyłki pocztowej w logu systemowym.

### Kontrola dostępu

Aplikacje implementujące MSH korzystające z protokołu transportowego SMTP mogą być zabezpieczone przed nieautoryzowanym dostępem za pomocą mechanizmów kontroli dostępu SMTP, opisanych w specyfikacji *SMTP Service Extension for Authentication* [RFC2554].

### Poufność na poziomie protokołu komunikacyjnego

W celu zapewnienia poufności przesyłek ebXML można stosować szyfrowanie w warstwie komunikacyjnej SMTP. Niezbędne szczegóły techniczne oraz listę dostępnych opcji zawiera specyfikacja IETF *SMTP Service Extension for Secure SMTP over TLS* [RFC2487].

## Parametry do zdefiniowania w CPA

| Nazwa parametru CPA | Opis | Dopuszczalne wartości |
| --- | --- | --- |
| CPAId | Unikalny identyfikator dokumentu CPA. | Zgodne z typem *String*  W3C XMLSchema |
| DigitalEnvelope | Określa rodzaj i wersję bezpiecznego standardu MIME, który jest wykorzystywany do szyfrowania dokumentów biznesowych. | Np.: S/MIME v2.0 |
| DuplicateElimination | Określa zasady eliminacji duplikatów w wymianie przesyłek ebXML. | Never, always, perMessage |
| ErrorURI | Określa *Error Reporting Location* I, gdzie mają być przesłane przesyłki informujące o błędzie | Zgodne z typem a*nyURI*  W3C XMLSchema |
| HashFunction | Określa algorytm funkcji skrótu wykorzystywany przy generowaniu i weryfikacji sygnatury XML. | SHA-1 |
| PersistDuration | Określa minimalną długość czasu, w którym dane z komunikatu przesłanego w protokole *reliable* *messaging* są przechowywane w *persistent storage* przez *ToPartyMSH*. | Zgodne z typem *duration* W3C XMLSchema np.:P15D (15 dni), P1M (1 miesiąc) |
| Retries | Określa maksymalną liczbę ponowień wysłania jakie powinno wykonać MSH aby przy użyciu tego samego protokołu komunikacyjnego dostarczyć przesyłkę ebXML w protokole *reliable messaging* | Zgodne z typem *integer*  W3C XMLSchema |
| SyncReplyMode | Określa rodzaj odpowiedzi ebXML w synchronicznej wymianie danych. | none*,* mshSignalsOnly, signalsAndResponse, signalsOnly*,* responseOnly, signalsAndResponse |
| SignatureAlgotithm | Określa algorytm kryptograficzny wykorzystywany przy generowaniu i weryfikacji sygnatury XML | Np.:RSA, DES, HMAC |
| TransportSecurity | Określa rodzaj zabezpieczeń przyjętych dla protokołu transportowego, który jest stosowany przez MSH do odbioru lub wysyłania danych. | Np.: SSLv3 |

# Bibliografia

Poniżej jest podana lista dokumentów, na które powołuje się Specyfikacja.

RFC wyspecyfikowane poniżej są udostępniane na stronach organizacji *The Internet Engineering Task Force* [(IETF)](http://www.ietf.org/glossary.html#IETF) w katalogu <http://www.ietf.org/rfc/>.

W przypadku specyfikacji opracowanych przez *W3C*, *ETSI* podano adresy URL, pod którymi są one publicznie dostępne.

[HMAC] RFC 2104. HMAC: Keyed-Hashing for Message Authentication. H. Krawczyk, M. Bellare, R. Canetti. February 1997

[HTTP] IETF RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. January 1997

[IPSEC] IETF RFC2402 IP Authentication Header. S. Kent, R. Atkinson. November 1998. RFC2406 IP Encapsulating Security Payload (ESP). S. Kent, R. Atkinson. November 1998.

[OpenPGP] RFC 2440. OpenPGP Message Format J. Callas, L. Donnerhacke, H. Finney, R. Thayer. November 1998.

[PGP/MIME] IETF RFC2015. MIME Security with Pretty Good Privacy (PGP). M. Elkins. October 1996.

[RFC2045] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N Freed & N Borenstein. November 1996

[RFC2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed, N. Borenstein. November 1996.

[RFC2253] Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. M. Wahl , S. Kille , T. Howes.December 1997.

[RFC2387] The MIME Multipart/Related Content-type. E. Levinson. August 1998.

[RFC2392] Content-ID and Message-ID Uniform Resource Locators. E. Levinson. August 1998.

[RFC2396] IUniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, Published August 1998

[RFC2487] SMTP Service Extension for Secure SMTP over TLS. P. Hoffman. January 1999.

[RFC2554] SMTP Service Extension for Authentication. J. Myers. March 1999.

[RFC2616] Hypertext Transfer Protocol, HTTP/1.1. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, June 1999.

[RFC2617] HTTP Authentication: Basic and Digest Access Authentication. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E. and L. Stewart, June 1999.

[RFC2817] Upgrading to TLS Within HTTP/1.1. Khare, R. and S. Lawrence. May 2000.

[RFC2818] HTTP Over TLS. E. Rescorla. May 2000.

[S/MIME] IETF RFC2311. S/MIME Version 2 Message Specification. S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.

[S/MIMECH] IETF RFC 2312. S/MIME Version 2 Certificate Handling. S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein. March 1998.

[S/MIMEV3] IETF RFC 2633. S/MIME Version 3 Message Specification. B. Ramsdell, Ed.. June 1999.

[SMTP] IETF RFC 822. Simple Mail Transfer Protocol, D Crocker, August 1982

[SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1. Don Box, DevelopMentor; David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman, Henrik Frystyk Nielsen, Satish Thatte, Microsoft; Noah Mendelsohn, Lotus Development Corp.; Dave Winer, UserLand Software, Inc. May 2000, http://www.w3.org/TR/SOAP.

[SOAPATTACH] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish Thatte and Henrik Frystyk Nielsen, Microsoft.October 2000, http://www.w3.org/TR/SOAP-attachments.

[SSL3] The SSL 3.0 Protocol, Netscape Communications Corp. A. Frier, P. Karlton, and P. Kocher,, Nov 18, 1996, <http://home.netscape.com/eng/ssl3/draft302.txt>.

[TLS] RFC2246. The TLS Protcol Version 1.0. T. Dierks, C. Allen. January 1999.

[XAdES] XML Advanced Electronic Signature. ETSI. February 2002, http://portal.etsi.org.

[XLINK] W3C XML Linking Candidate Recommendation. June 2001, http://www.w3.org/TR/2001/REC-xlink-20010627/.

[XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second Edition), October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[XML Namespace] W3C Recommendation for Namespaces in XML, World Wide Web Consortium. January 1999, http://www.w3.org/TR/REC-xml-names.

[XMLDSIG] RFC 3275. W3C/IETF XML-Signature Syntax and Processing specification, February 2002,  
<http://www.w3.org/TR/xmldsig-core/>.

[XMLMedia] IETF RFC 3023, XML Media Types. M. Murata, S. St.Laurent. January 2001

[XMLENC] XML Encryption Syntax and Processing. T. Imamura, B. Dillaway, E. Simon. October 2002, http://www.w3.org/TR/xmlenc-core/.