ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 71811— 2024

Цифровая промышленность

УНИФИЦИРОВАННАЯ АРХИТЕКТУРА OPC

Часть 6

Сопоставления

(IEC 62541-6:2020, NEQ)

Издание официальное

Москва
Российский институт стандартизации
2025

Предисловие

1 РАЗРАБОТАН Ассоциацией «Цифровые инновации в машиностроении» (АЦИМ) и Федеральным государственным бюджетным учреждением «Российский институт стандартизации» (ФГБУ «Институт стандартизации»)

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 306 «Измерения, управление и автоматизация в промышленных процессах»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 29 ноября 2024 г. № 1807-ст

4 Настоящий стандарт разработан с учетом основных нормативных положений международного стандарта МЭК 62541-6:2020 «Унифицированная архитектура OPC. Часть 6. Сопоставления» (IEC 62541-6:2020 «OPC Unified Architecture — Part 6: Mappings», NEQ)

5 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.rst.gov.ru)

Содержание

Введение

Настоящий стандарт входит в серию стандартов под общим наименованием «Цифровая промышленность. Унифицированная архитектура OPC», будучи руководством по применению основных нормативных положений, определяющих одно или несколько отображений, поддерживающих реализацию конкретных приложений. Например, одно из них — это сетевые службы XML, однако службы, описанные в настоящем стандарте, используют методы сетевых служб в контексте WSDL.

Настоящий стандарт является руководством для управления созданием умных производств, в котором с помощью представленных сопоставлений определено понимание отдельных частей многокомпонентной системы стандартов в цифровой промышленности.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Цифровая промышленность

УНИФИЦИРОВАННАЯ АРХИТЕКТУРА OPC

Часть 6

Сопоставления

Digital industry.
OPC unified architecture.
Part 6. Mapping

Дата введения — 2025—02—01

1 Область применения

Настоящий стандарт является руководством по применению основных нормативных положений, определяющих сопоставление нескольких отображений, поддерживающих реализацию конкретных приложений.

Настоящий стандарт предназначен для управления процессом создания умных производств в цифровой промышленности.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ГОСТ Р 59799 Умное производство. Модель эталонной архитектуры индустрии 4.0 (RAMI 4.0)

ГОСТ Р 70988 Система стандартов в цифровой промышленности. Основные положения. Общие требования к системе

ГОСТ Р 70992 Цифровая промышленность. Интеграция и интероперабельность систем. Термины и определения

ГОСТ Р 71808 Цифровая промышленность. Унифицированная архитектура OPC. Часть 3. Модель адресного пространства

ГОСТ Р 71809 Цифровая промышленность. Унифицированная архитектура OPC. Часть 4. Сервисы

ГОСТ Р 71810 Цифровая промышленность. Унифицированная архитектура OPC. Часть 5. Информационная модель

Примечание — При пользовании настоящим стандартом целесообразно проверить действие ссылочных стандартов в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если заменен ссылочный стандарт, на который дана недатированная ссылка, то рекомендуется использовать действующую версию этого стандарта с учетом всех внесенных в данную версию изменений. Если заменен ссылочный стандарт, на который дана датированная ссылка, то рекомендуется использовать версию этого стандарта с указанным выше годом утверждения (принятия). Если после утверждения настоящего стандарта в ссылочный стандарт, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение

Издание официальное

рекомендуется применять без учета данного изменения. Если ссылочный стандарт отменен без замены, то положение, в котором дана ссылка на него, рекомендуется применять в части, не затрагивающей эту ссылку.

3 Термины, определения и сокращения

3.1 Термины и определения

В настоящем стандарте применены термины по ГОСТ Р 70992, а также следующие термины с соответствующими определениями:

3.1.1 **кодирование данных:** Способ сериализации сообщений OPC UA и структур данных.

Примечание — OPC UA — это стандарт, устанавливающий методы обмена сообщениями между сервером и клиентом OPC, не зависящие от вида аппаратно-программной платформы и от типа взаимодействующих систем и сетей.

3.1.2 **платформа разработки:** Набор инструментов и/или языков программирования, используемых для создания программного обеспечения.

3.1.3 **протокол безопасности:** Протокол, обеспечивающий целостность и конфиденциальность сообщений UA, которыми обмениваются приложения OPC UA.

3.1.4 **профиль стека:** Комбинация сопоставлений кодировки данных (DataEncodings), протокола безопасности (SecurityProtocol) и транспортного протокола (TransportProtocol).

Примечание — Приложения OPC UA реализуют один или несколько профилей стеков (StackProfiles) и могут взаимодействовать только с приложениями OPC UA, которые соотносятся с поддерживаемыми ими профилями стека (StackProfile).

3.1.5 **сертифицированный дайджест:** Короткий идентификатор, используемый для однозначной идентификации сертификата X.509 v3.

Примечание — Данный идентификатор работает как функция свертки, преобразующая массив данных сертификата в выходную битовую строку.

3.1.6 **сопоставление:** Спецификация того, как реализовать функцию OPC UA с помощью конкретной технологии.

Примечание — Например, двоичное кодирование OPC UA — это отображение, определяющее, как сериализовать структуры данных OPC UA в виде последовательностей байтов.

3.1.7 **транспортное соединение:** Полнодуплексный канал связи, установленный между приложениями OPC UA.

Примечание — Сокет TCP/IP является примером транспортного соединения (TransportConnection).

3.1.8 **транспортный протокол:** Способ обмена сериализованными сообщениями OPC UA между приложениями OPC UA.

3.2 Сокращения

В настоящем стандарте приведены следующие сокращения:

ЦС — центр сертификации;
API — интерфейс прикладного программирования (application programming interface);
AS — автономная система IP-сетей и маршрутизаторов, управляемых одним или несколькими операторами (autonomous system);
ASN.1 — абстрактная синтаксическая нотация № 1 (используется в X690) (Abstract Syntax Notation #1 (used in X690);
CRL — список сертификатов, которые удостоверяющий центр пометил как отозванные (certificate revocation list);
CSV — значение, разделенное запятыми (формат файла) (comma separated value (file format);
HTTP — протокол передачи гипертекста (Hypertext Transfer Protocol);
HTTPS — защищенный протокол передачи гипертекста (Secure Hypertext Transfer Protocol);
IPSec — безопасность интернет-протокола (Internet Protocol Security);

NTP — технология, при помощи которой можно обеспечить точным временем компьютеры или другие сетевые устройства;
OID — идентификатор объекта (используется с ASN.1) (object identifier (used with ASN.1);
PRF — псевдослучайная функция (pseudo random function);
RSA — аббревиатура от фамилий Ривест, Шамир и Адлеман (система шифрования с открытым ключом) (Rivest, Shamir and Adleman (public key encryption system);
SOAP — простой протокол доступа к объектам (Simple Object Access Protocol);
SSL — уровень защищенных сокетов (определен в SSL/TLS) (Secure Sockets Layer (defined in SSL/TLS);
TAI — высокоточный атомный координатный стандарт времени, основанный на условном течении собственного времени на геоиде Земли (Международное атомное время) (temps atomique international);
TCP — протокол управления передачей (Transmission Control Protocol);
TLS — безопасность транспортного уровня (определена в SSL/TLS) (Transport Layer Security;
UA — унифицированная архитектура (Unified Architecture);
UA CP — протокол подключения OPC UA (OPC UA Connection Protocol);
UA SC — защищенный разговор OPC UA (OPC UA Secure Conversation);
WS-* — спецификации веб-сервисов XML (XML Web Services specifications);
WSDL — язык описания веб-сервисов и доступа к ним, основанный на языке XML (Web Services Description Language);
XML — расширяемый язык разметки (eXtensible Markup Language).

4 Общие положения

Другие части серии стандартов под общим наименованием «Цифровая промышленность. Унифицированная архитектура OPC» изложены в ГОСТ Р 70988 таким образом, чтобы быть независимыми от технологии, применяемой для реализации данной архитектуры. Такой подход означает, что OPC UA представляет собой гибкую спецификацию, которая будет продолжать применяться по мере развития цифровых технологий. В то же время невозможно создать приложение OPC UA с учетом информации, содержащейся в данной серии стандартов, так как опущены существенные детали ее реализации.

Настоящий стандарт определяет сопоставления между абстрактными спецификациями и технологиями, которые можно использовать для их реализации. Данные сопоставления организованы в три группы: DataEncodings (кодировки данных), SecurityProtocols (протоколы безопасности) и TransportProtocols (транспортные протоколы).

Различные сопоставления объединены для создания профилей стеков. Приложения OPC UA должны реализовывать минимум один профиль стеков и иметь возможность взаимодействия с другими приложениями OPC UA, выполняющими такой же профиль стеков.

Настоящий стандарт определяет DataEncodings в разделе 5, SecurityProtocols в 5.4 и TransportProtocols, StackProfiles приведены в [1].

Взаимосвязь между приложениями OPC UA основана на обмене сообщениями. Параметры, содержащиеся в сообщениях, рассмотрены в ГОСТ Р 71809; однако их формат определен DataEncoding и TransportProtocol. По этой причине каждое сообщение (см. ГОСТ Р 71809) должно иметь такое описание, которое точно устанавливает, что должно передаваться по проводу. Нормативные описания установлены в приложениях OPC UA.

Стек — набор программных библиотек, реализующих один или несколько StackProfile. Интерфейс между приложением OPC UA и стеком — это API, который не раскрывает детали реализации стека. API зависит от конкретной платформы разработки. Из-за ограничений платформы разработки типы данных, представленные в API для данной платформы разработки, могут не соответствовать типам данных, определенным в спецификации. Например, язык программирования Java не поддерживает целое число без знака, что означает, что любой API Java должен сопоставлять целые числа без знака с целочисленным типом со знаком.

На рисунке 1 представлен обзор стека OPC UA, показывающий взаимосвязь между различными концепциями, определенными в настоящем стандарте.

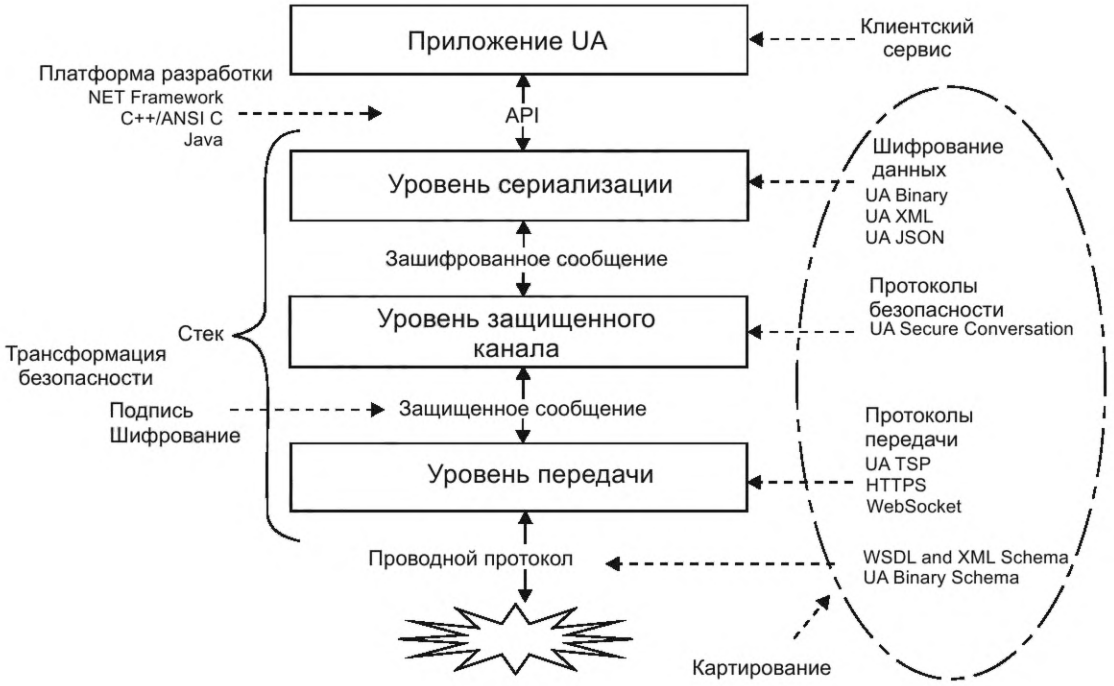

Рисунок 1 — Обзор стека OPC UA

Уровни, описанные в настоящем стандарте, не соответствуют уровням 7-уровневой модели OSI (X200). Каждый профиль стека OPC UA следует рассматривать как отдельный протокол уровня 7, построенный на основе существующих протоколов уровней 5, 6 или 7, такого как TCP/IP, TLS или HTTP. Уровень SecureChannel присутствует даже в том случае, если SecurityMode имеет значение None. Реализация SecurityProtocol должна поддерживать логический канал с уникальным идентификатором. Требуются дополнительные проверки SecureChannel с SecurityMode, для которого установлено значение None, если приложение не работает в физически защищенной сети или не используется протокол низкого уровня, такой как IPSec.

5 Кодирование данных

5.1 Общие положения

5.1.1 Общие сведения

В настоящем стандарте установлены три кодировки данных: OPC UA Binary, OPC UA XML и OPC UA JSON, а также приведен порядок создания сообщений, применяя каждую из указанных кодировок.

5.1.2 Встроенные типы данных

Все кодировки данных OPC UA основаны на правилах, определенных для стандартного набора встроенных типов, которые используют для создания структур, массивов и сообщений согласно ГОСТ Р 70988. Описание встроенных типов приведено в таблице 1.

Таблица 1 — Встроенные типы данных

ID	Имя	Описание
1	Boolean	Логическое состояние (истина или ложь)
2	SByte	Целое значение в диапазоне от –128 до +127 включ.
3	Byte	Целое значение в диапазоне от 0 до 255 включ.
4	Int16	Целое значение в диапазоне от –32 768 до +32 767 включ.
5	UInt16	Целое значение в диапазоне от 0 до 65 535 включ.

Окончание таблицы 1

ID	Имя	Описание
6	Int32	Целое значение в диапазоне от –2 147 483 648 до +2 147 483 647 включ.
7	UInt32	Целое значение в диапазоне от 0 до 4 294 967 295 включ.
8	Int64	Целое значение в диапазоне от –92 23 372 036 854 775 808 до +9 223 372 036 854 775 807 включ.
9	UInt64	Целое значение в диапазоне от 0 до 18 446 744 073 709 551 615 включ.
10	Float	32-битное значение с плавающей точкой IEEE
11	Double	64-битное значение с плавающей точкой IEEE
12	String	Последовательность символов Unicode
13	DateTime	Временная отметка
14	Guid	16-байтное значение, которое может быть использовано как глобальный уникальный идентификатор
15	ByteString	Последовательность октетов
16	XMLElement	Элемент XML
17	NodeId	Идентификатор для узла в адресном пространстве OPC UA сервера
18	ExpandedNodeId	NodeId, позволяющий указать URI пространства имен вместо индекса
19	StatusCode	Числовой идентификатор ошибки или условия, связанного со значением или операцией
20	QualifiedName	Имя, определенное пространством имен
21	LocalizedText	Читаемый человеком текст с необязательным локальным идентификатором
22	ExtensionObject	Структура, содержащая тип данных, специфичный для приложения, который может не распознаваться получателем
23	DataValue	Значение данных со связанным кодом состояния и метками времени
24	Variant	Объединение всех типов, указанных выше
25	DiagnosticInfo	Структура, содержащая подробную информацию об ошибках и диагностическую информацию, связанную с StatusCode

Большинство приведенных в таблице 1 типов данных аналогичны абстрактным типам в соответствии с ГОСТ Р 71808 и ГОСТ Р 71809. В настоящем стандарте определены типы данных ExtensionObject и Variant, а также представление типа данных Guid согласно ГОСТ Р 71808.

5.1.3 Guid

Тип данных Guid — это 16-байтовый глобальный уникальный идентификатор, структура которого приведена в таблице 2.

Таблица 2 — Структура Guid

Компонент	Типы данных
Data1	UInt32
Data2	UInt16
Data3	UInt16
Data4	Byte 8

Значения Guid могут быть представлены в виде строки в следующем виде:

<Data1>-<Data2>-<Data3>-<Data4[0:1]>-<Data[2:7]>,

где Data1 имеет ширину 8 символов, Data2 и Data3 — 4 символа, а каждый байт в Data4 — 2 символа. Каждое значение форматируется как шестнадцатеричное число с добавленными нулями. Типичное значение Guid представляется в формате строки:

C496578A-0DFE-4B8F-870A-745238C6AEAE

5.1.4 ByteString

Тип данных ByteString структурно аналогичен одномерному массиву байтов и представлен как отдельный встроенный тип данных, так как позволяет кодерам оптимизировать передачу значения. Допускается, что некоторые платформы разработки не сохраняют различие между ByteString и одномерным массивом Byte.

Если декодер для платформы разработки не сохраняет различие, он должен преобразовать все одномерные массивы Byte в ByteStrings.

Каждый элемент в одномерном массиве ByteString может иметь разную длину, что означает структурное отличие от двумерного массива Byte, где длина каждого измерения одинаковая. Таким образом, декодеры должны сохранять различие между двумя или более массивами измерений Byte и одним или несколькими массивами измерений ByteString.

Если платформа разработки не поддерживает целые числа без знака, ей необходимо представлять ByteStrings как массивы SByte. В этом случае требования для Byte будут применены к SByte.

5.1.5 ExtensionObject

Тип данных ExtensionObject — это контейнер для любых структурированных типов данных, которые невозможно закодировать как один из других встроенных типов данных. ExtensionObject содержит сложное значение, сериализованное в виде последовательности байтов или элемента XML, а также идентификатор, указывающий, какие данные содержатся в контейнере и как они закодированы.

Структурированные типы данных представлены в адресном пространстве сервера как подтипы типа данных структуры DataEncodings, доступные для любых заданных структурированных типов данных, представлены как объект DataTypeEncoding в адресном пространстве сервера. NodeId для объекта DataTypeEncoding — это идентификатор, хранящийся в ExtensionObject согласно ГОСТ Р 71808, описывает, как узлы DataTypeEncoding связаны с другими узлами AddressSpace.

Разработчики сервера должны применять числовые идентификаторы NodeId, соответствующие пространству имен, для любых объектов DataTypeEncoding, которые они определяют. Применение данных идентификаторов сведет к минимуму накладные расходы, связанные с упаковкой значений структурированного типа данных в ExtensionObject.

Объекты типов данных ExtensionObject и Variant допускают неограниченную вложенность, что может привести к ошибкам переполнения стека, даже если размер сообщения меньше максимально допустимого. Декодеры должны поддерживать не менее 100 уровней вложенности и сообщать об ошибке, если количество уровней вложенности превышает поддерживаемое.

5.1.6 Variant

Тип данных Variant — объединение всех встроенных типов данных, включая ExtensionObject. Variants также могут содержать массивы любого из этих встроенных типов. Variants используются для хранения значения или параметра с типом данных BaseDataType или одним из его подтипов.

Variants могут быть пустыми. Пустой Variant имеет нулевое значение (NULL), и он рассматривается как пустой столбец в базе данных SQL. Допустимо, чтобы NULL в Variant не совпадал с NULL для тех типов данных, которые поддерживают значения NULL, например строки. Также допустимо, что некоторые платформы разработки не сохраняют различие между NULL в DataType и NULL в Variant, поэтому приложения не должны полагаться на это различие. Данное требование также означает, что, если атрибут поддерживает запись NULL, он также должен поддерживать запись NULL в Variant, и наоборот.

Variant могут содержать массивы Variants, но не могут напрямую применять другой Variant.

Типы DiagnosticInfo имеют значение только в том случае, если они возвращаются в ответном сообщении со связанным StatusCode и таблицей строк. В результате Variants не могут содержать экземпляры DiagnosticInfo.

Значения атрибутов неизменно возвращаются в экземплярах DataValue. Следовательно, DataType атрибута не может быть DataValue. Variants могут содержать DataValue при использовании в других контекстах, таких как аргументы метода или сообщения PubSub. Variant в DataValue не может прямо или косвенно применять другое DataValue.

Переменные с DataType BaseDataType сопоставляются с Variant, однако атрибуты ValueRank и ArrayDimensions накладывают ограничения на то, что разрешено в Variant. Например, если ValueRank имеет значение Scalar, тогда Variant может содержать только скалярные значения.

Типы данных ExtensionObject и Variant допускают неограниченную вложенность, что может привести к ошибкам переполнения стека, даже если размер сообщения меньше максимально допустимого. Декодеры должны поддерживать не менее 100 уровней вложенности и должны сообщать об ошибке, если количество уровней вложенности превышает поддерживаемое.

5.1.7 Decimal

Тип данных Decimal — это десятичное число со знаком высокой точности. Он состоит из целочисленного немасштабированного значения произвольной точности и целочисленного масштаба.

Decimal — это степень десяти, которая применяется к немасштабированному значению. Decimal имеет поля, указанные в таблице 3.

Таблица 3 — Расположение десятичных дробей

Поле	Тип	Описание полей
TypeId	NodeId	Идентификатор десятичного типа данных
Encoding	Byte	Неизменное значение –1
Length	Int32	Длина десятичной дроби. Если длина менее или равна 0, то десятичное значение равно 0
Scale	Int16	Целое число со знаком, представляющее степень десяти, используемое для масштабирования значения, т. е. десятичное число значения, умноженное на 10 по шкале. Целое число кодируется начиная с младшего бита
Value	Byte	Целое число со знаком с двумя дополнениями, представляющее немасштабированное значение. Количество бит выводится из длины поля длины. Если количество бит равно 0, то и значение равно 0. Целое число кодируется сначала младшим значащим байтом

Когда Decimal закодирован в Variant, встроенный тип устанавливается в ExtensionObject. Декодеры, которые не понимают тип данных Decimal, должны обрабатывать его как любую другую неизвестную структуру и передавать приложению. Декодеры, которые понимают десятичное число, могут анализировать значение и использовать любую конструкцию, подходящую для платформы разработки.

Если десятичное число встроено в другую структуру, то DataTypeDefinition для поля должен указывать NodeId десятичного узла в качестве типа данных. Если сервер публикует описание двоичного типа OPC для структуры, то в описании типа для поля DataType должно быть установлено значение ExtensionObject.

5.2 Двоичные данные OPC UA

5.2.1 Общие сведения

Двоичное кодирование данных OPC UA — это формат данных, разработанный для удовлетворения потребностей в производительности приложений OPC UA. Этот формат предназначен в первую очередь для быстрого кодирования декодирования, однако также учитывает размер закодированных данных, передаваемых по сети.

Двоичное кодирование данных OPC UA основано на нескольких примитивных типах данных с четко определенными правилами кодирования, которые можно последовательно записывать или читать из двоичного потока. Подобная структура кодируется путем последовательной записи закодированной формы каждого поля. Если поля характеризуются одинаковой структурой, то значения каждого из полей последовательно фиксируются перед записью следующего поля в данной структуре. В поток данных должны быть записаны все поля, даже если они содержат NULL. Кодировки для каждого примитивного типа определяют, как кодировать либо NULL, либо значение по умолчанию для конкретного типа.

Двоичное кодирование данных OPC UA не включает какую-либо информацию о типе или имени поля, поскольку предполагается, что все приложения OPC UA заранее наделены информацией о службах и структурах, которые они поддерживают. Исключением является ExtensionObject, предоставляю-

щий идентификатор и размер структуры Structured DataType, которую он представляет, что позволяет декодеру пропускать типы, которые он не распознает.

5.2.2 Встроенные типы

5.2.2.1 Логическое значение

Логическое значение должно быть закодировано как 1 байт, где значение 0 (ноль) является ложным, а любое ненулевое значение — истинным. Кодировщики должны применять значение 1 для указания истинного значения, однако декодеры должны рассматривать как истинное любое ненулевое значение.

5.2.2.2 Целое число

Все целочисленные типы должны кодироваться как значения с прямым порядком байтов, при этом наименее значащий байт появляется в потоке первым.

На рисунке 2 показано, как значение 1 000 000 000 (шестнадцатеричное: 3B9ACA00) кодируется в потоке как 32-битное целое число.

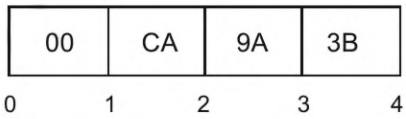

Рисунок 2 — Кодирование целых чисел в двоичный поток

5.2.2.3 Плавающая запятая

Все значения с плавающей запятой должны быть закодированы с помощью соответствующего двоичного представления IEEE-754, которое состоит из трех основных компонентов: знака, показателя степени и дроби. Битовые диапазоны, назначенные каждому компоненту, зависят от ширины типа. В таблице 4 перечислены диапазоны битов для поддерживаемых типов с плавающей запятой.

Таблица 4 — Поддерживаемые типы чисел с плавающей запятой

Имя	Длина, бит	Доля	Экспонента	Знак
Float	32	От 0 до 22	От 20 до 30	31
Double	64	От 0 до 51	От 52 до 62	63

Кроме того, значимым является порядок байтов в потоке. Все значения с плавающей запятой должны кодироваться так, чтобы младший байт стоял первым (т. е. с прямым порядком байтов).

На рисунке 3 показано, как значение 6,5 (шестнадцатеричное: C0D00000) кодируется как число с плавающей запятой.

Тип с плавающей запятой поддерживает положительную и отрицательную бесконечность, а также нечисловые значения (NaN). Спецификация IEEE допускает несколько вариантов NaN, однако кодеры/декодеры могут не сохранять это различие. Кодеры должны кодировать значение NaN как «тихое» NAN IEEE (000000000000F8FF) или (0000C0FF). Любые неподдерживаемые типы, такие как денормализованные числа, должны кодироваться как «тихая» NAN IEEE. Любой тест на равенство значений NaN всегда не будет считаться пройденным.

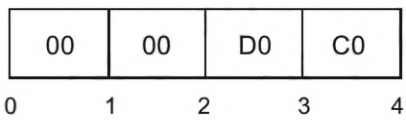

Рисунок 3 — Кодирование чисел с плавающей запятой в двоичном потоке

5.2.2.4 Строки

Все значения String кодируются как последовательность символов UTF-8 без нулевого символа-терминатора, которым предшествует длина в байтах.

Длина в байтах кодируется как Int32. Значение –1 используется для обозначения нулевой строки.

На рисунке 4 показано, как многоязычная строка Boy кодируется в потоке байтов.

Рисунок 4 — Кодирование строк в двоичный поток

5.2.2.5 DateTime

Значение DateTime должно быть закодировано как 64-битное целое число со знаком (см. 5.2.2.2), которое представляет количество 100-наносекундных интервалов с 1 января 1601 года (UTC).

Не все платформы разработки смогут предоставлять полный диапазон дат и времени, которые могут быть представлены с помощью этого DataEncoding. Например, структура UNIX time_t имеет разрешение только в 1 с и не может представлять даты до 1970 года. По этой причине при работе со значениями даты/времени, выходящими за пределы динамического диапазона платформы разработки, должны применяться следующие правила:

а) значение даты/времени кодируется как 0, если:

1) значение равно или ранее 1601-01-01 00:00 UTC,

2) значение представляет собой наиболее раннюю дату, которая может быть представлена с помощью кодировки платформы разработки;

б) дата/время кодируются как максимальное значение для Int64, если:

1) согласно универсальному мировому стандарту времени (UTC) значение равно или более 9999-12-31 23:59:59 UTC,

2) значение — это наиболее поздняя дата, которую можно представить с помощью кодировки платформы разработки;

в) дата/время декодируется как самое раннее время, которое может быть представлено на платформе, если:

1) или закодированное значение равно,

2) или закодированное значение представляет собой время, предшествующее самому раннему времени, которое может быть представлено с помощью кодировки платформы разработки;

г) дата/время декодируется как самое позднее время, которое может быть представлено на платформе, если:

1) закодированное значение является максимальным значением для Int64,

2) закодированное значение представляет собой более позднее время, чем самое позднее время, которое может быть представлено с помощью кодировки платформы разработки.

Данные правила подразумевают, что самые ранние и самые поздние промежутки времени, которые могут быть представлены на данной платформе, являются недопустимыми значениями даты и времени и должны обрабатываться приложениями соответствующим образом.

Если декодер фиксирует значение DateTime с разрешением, превышающим разрешение, поддерживаемое на платформе разработки, он должен искать значение допустимое.

5.2.2.6 Guid

Guid кодируется в структуру (см. таблицу 2). Поля кодируются последовательно в соответствии с типом данных поля. На рисунке 5 показано, как Guid «72962B91-FA75-4AE6-8D28-B404DC7DAF63» кодируется в потоке байтов.

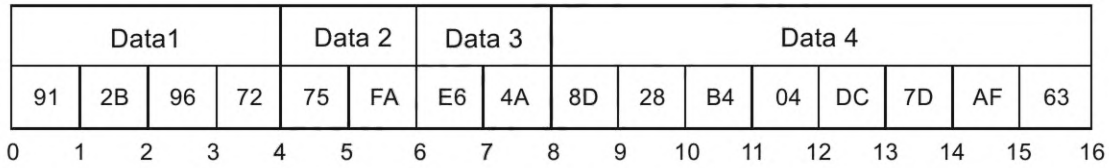

Рисунок 5 — Кодирование Guid в двоичном потоке

5.2.2.7 ByteString

ByteString кодируется как последовательность байтов, которой предшествует ее длина в байтах. Длина кодируется как 32-битное целое число со знаком, как описано выше.

Если длина строки байтов равна –1, то строка байтов имеет значение «ноль».

5.2.2.8 XmlElement

XmlElement — это фрагмент XML, сериализованный как строка UTF-8, а затем закодированный как ByteString. На рисунке 6 показано, как XmlElement «<A>Hot</A>» кодируется в потоке байтов.

Рисунок 6 — Кодирование XmlElement в двоичный поток

Декодер может выбрать анализ XML после декодирования; если возникает неисправимая ошибка синтаксического анализа, декодер должен продолжить обработку потока. Например, если XmlElement является телом Variant или элементом массива, который является телом Variant, то об этой ошибке требуется сообщить, установив значение StatusCode Bad_DecodingError для параметра Variant, определяющего тело Variant.

5.2.2.9 NodeId

Компоненты NodeId приведены в таблице 5.

Таблица 5 — Компоненты NodeId

Имя	Тип данных	Описание компонент NodeId
Namespace	UInt16	Индекс для URI пространства имен. Индекс 0 используется для NodeId-идентификаторов, определенных OPC UA
IdentifierType	Перечисление	Формат и тип данных идентификатора. Значение может быть одним из следующих: NUMERIC представляет собой UIInteger; STRING — String; GUID — Guid; OPAQUE — ByteString
Value	Не определен	Идентификатор узла в адресном пространстве сервера OPC UA

DataEncoding для NodeId варьируется в зависимости от содержимого экземпляра. По этой причине первый байт закодированной формы указывает формат остальной части закодированного NodeId. Возможные форматы кодирования данных приведены в таблице 6. В таблицах 6—9 приведено описание структуры каждого возможного формата (исключают байт, обозначающий формат).

Таблица 6 — Значения NodeId DataEncoding

Имя	Значение	Описание NodeId DataEncoding
Two Byte	0x00	Числовое значение, соответствующее 2-байтовому представлению
Four Byte	0x01	Числовое значение, соответствующее 4-байтовому представлению
Numeric	0x02	Числовое значение, которое не вписывается в 2- или 4-байтовые представления
String	0x03	Строковое значение
Guid	0x04	Ориентировочное значение
ByteString	0x05	Непрозрачное значение (ByteString)
NamespaceUri Flag	0x80	См. ExpandedNodeId в 5.2.2.10
ServerIndex Flag	0x40	

Стандартное кодирование данных NodeId имеет структуру, представленную в таблице 7. Стандартное кодирование данных NodeId применяют для всех форматов, для которых явный формат не определен.

Таблица 7 — Стандартное кодирование двоичных данных NodeId

Имя	Значение	Описание двоичных данных NodeId
Namespace	UInt16	NamespaceIndex
Identifier	Не определено	Идентификатор, который кодируется по следующим правилам: NUMERIC — UInt32; STRING — String; GUID — Guid; OPAQUE — ByteString

Пример строкового NodeId с пространством имен, равным 1, и идентификатором, равным Hot, показан на рисунке 7.

Рисунок 7 — Строковый NodeId, определяемый шифрованием и идентификатором

2-байтовое кодирование данных NodeId имеет структуру, приведенную в таблице 8.

Таблица 8 — Кодирование 2-байтовых двоичных данных NodeId

Имя	Тип данных	Описание 2-байтовых двоичных данных NodeId
Identifier	Byte	Namespace — это пространство имен OPC UA по умолчанию (т. е. 0). Тип Identifier — Numeric. Identifier должен находиться в диапазоне от 0 до 255

Пример 2-байтового NodeId с идентификатором, равным 72, дан на рисунке 8.

Шифрование	Идентификатор
00	72
0	1 2

Рисунок 8 — 2-байтовый NodeId

4-байтовое кодирование данных NodeId имеет структуру согласно таблице 9.

Таблица 9 — Кодирование 4-байтовых двоичных данных NodeId

Имя	Тип данных	Описание 4-байтовых двоичных данных NodeId
Namespace	Byte	Namespace должен находиться в диапазоне от 0 до 255
Identifier	UInt16	Тип Identifier — Numeric. Identifier должен находиться в диапазоне от 0 до 65 535

Пример 4-байтового NodeId с пространством имен, равным 5, и идентификатором, равным 1025, представлен на рисунке 9.

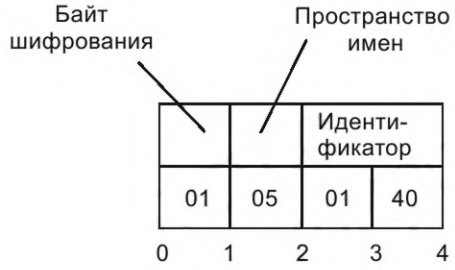

Рисунок 9 — 4-байтовый NodeId

5.2.2.10 ExpandedNodeId

ExpandedNodeId расширяет структуру NodeId, позволяя явно указывать NamespaceUri вместо использования NamespaceIndex. NamespaceUri является необязательным. Если он указан, то NamespaceIndex внутри NodeId должен игнорироваться.

ExpandedNodeId кодируется путем сначала кодирования NodeId, как описано в 5.2.2.9, а затем кодирования Namespace Uri как строки.

Экземпляр ExpandedNodeId может использовать NamespaceIndex вместо NamespaceUri. В этом случае NamespaceUri не кодируется в потоке. Присутствие NamespaceUri в потоке обозначается установкой флага NamespaceUri в байте формата кодирования для NodeId.

Если NamespaceUri присутствует, то кодер должен кодировать NamespaceIndex как 0 в потоке, когда кодируется часть NodeId. Неиспользуемый NamespaceIndex включен в поток для обеспечения единообразия.

ExpandedNodeId также может иметь ServerIndex, который закодирован как UInt32 после NamespaceUri. Флаг ServerIndex в байте кодирования NodeId указывает, присутствует ли ServerIndex в потоке. ServerIndex опускается, если он равен нулю.

Кодировка ExpandedNodeId имеет структуру, представленную в таблице 10.

Таблица 10 — Кодирование двоичных данных ExpandedNodeId

Имя	Тип данных	Описание двоичных данных ExpandedNodeId
NodeId	NodeId	Флаги NamespaceUri и ServerIndex в кодировке NodeId указывают, присутствуют ли эти поля в потоке
NamespaceUri	String	Не присутствует, если значение равно нулю или пусто
ServerIndex	UInt32	Не присутствует, если 0

5.2.2.11 Код состояния

StatusCode кодируется как UInt32.

5.2.2.12 DiagnosticInfo

Структура DiagnosticInfo описана в ГОСТ Р 71809, в котором указан ряд полей, которые могут отсутствовать. По этой причине при кодировании используется битовая маска, чтобы определить, какие поля действительно присутствуют в закодированной форме.

Поля SymbolicId, NamespaceUri, LocalizedText и Locale являются индексами в таблице строк, которая возвращается в заголовке ответа (см. ГОСТ Р 71809). Кодируется только индекс соответствующей строки в таблице строк. Индекс –1 указывает, что для строки значение отсутствует.

Представленное в таблице 11 кодирование DiagnosticInfo допускает неограниченное вложение, что может привести к ошибкам переполнения стека, даже если размер сообщения меньше максимально допустимого. Декодеры должны поддерживать не менее 100 уровней вложенности и сообщать об ошибке, если количество уровней вложенности превышает поддерживаемое.

Таблица 11 — Кодирование двоичных данных DiagnosticInfo

Имя	Тип данных	Описание двоичных данных DiagnosticInfo
Encoding Mask	Byte	Битовая маска, указывающая, какие поля присутствуют в потоке. Маска имеет биты: 0x01 — Symbolic Id 0x02 — Namespace 0x04 — LocalizedText 0x08 — Locale 0x10 — Additional Info 0x20 — InnerStatusCode 0x40 — InnerDiagnosticInfo
SymbolicId	Int32	Символическое имя для кода состояния
NamespaceUri	Int32	Пространство имен, определяющее символический идентификатор
Locale	Int32	Языковой стандарт, используемый для локализованного текста
LocalizedText	Int32	Удобочитаемая краткая информация о коде состояния
Additional Info	String	Подробная диагностическая информация для конкретного приложения
Inner StatusCode	StatusCode	Код состояния, предоставляемый базовой системой
Inner DiagnosticInfo	DiagnosticInfo	Диагностическая информация, связанная с внутренним кодом состояния

5.2.2.13 QualifiedName

Структура QualifiedName кодируется согласно таблице 12. Абстрактная структура QualifiedName приведена в ГОСТ Р 71808.

Таблица 12 — Кодирование двоичных данных QualifiedName

Имя	Тип данных	Описание двоичных данных QualifiedName
NamespaceIndex	UInt16	Индекс пространства имен
Name	String	Имя

5.2.2.14 LocalizedText

Структура LocalizedText содержит два поля, которые могут отсутствовать. По этой причине при кодировании используется битовая маска, чтобы указать, какие поля действительно присутствуют в закодированной форме.

Абстрактная структура LocalizedText приведена в ГОСТ Р 71808, а кодирование двоичных данных LocalizedText представлено в таблице 13.

Таблица 13 — Кодирование двоичных данных LocalizedText

Имя	Тип данных	Описание двоичных данных LocalizedText
EncodingMask	Byte	Битовая маска, указывающая, какие поля присутствуют в потоке. Маска имеет биты: 0x01 — Locale; 0x02 — Text
Locale	String	Языковой стандарт. Пропущено или пусто
Text	String	Текст в указанной локализации. Пропущено или пусто

5.2.2.15 ExtensionObject

Объект ExtensionObject кодируется как последовательность байтов, перед которыми находятся NodeId, его DataTypeEncoding и количество закодированных байтов.

ExtensionObject может быть закодирован приложением, что означает, что он передается кодировщику как ByteString или XmlElement. В этом случае кодер сможет записать количество байтов в объекте до того, как он закодирует байты. Однако ExtensionObject может знать, как кодировать/декодировать себя, что означает, что кодер должен вычислить количество байтов перед кодированием объекта или иметь возможность выполнять поиск назад в потоке и обновлять длину после кодирования тела.

Если декодер идентифицирует ExtensionObject, он должен проверить, распознает ли он идентификатор DataTypeEncoding. Если да, то он может вызвать соответствующую функцию для декодирования тела объекта. Если декодер не распознает тип, он должен использовать кодировку, чтобы определить, является ли тело ByteString или XmlElement, а затем декодировать тело объекта или обработать его как непрозрачные данные и пропустить его.

Сериализованная форма ExtensionObject приведена в таблице 14.

Таблица 14 — Кодирование двоичного типа данных расширенного объекта

Имя	Тип данных	Описание кодов полей
TypeId	NodeId	Идентификатор узла DataTypeEncoding в адресном пространстве сервера. Объектам ExtensionObject, определенным спецификацией OPC UA, присвоен числовой идентификатор узла с NamespaceIndex, равным 0. Числовые идентификаторы определены в A.3. Декодеры используют это поле для определения синтаксиса тела. Например, если это поле является NodeId объекта кодировки JSON для DataType, то Body представляет собой ByteString, содержащий документ JSON, закодированный как строка UTF-8
Encoding	Byte	Перечисление, указывающее, как кодируется тело. Параметр может иметь следующие значения: 0x00 — тело не закодировано; 0x01 — тело закодировано как ByteString; 0x02 — тело закодировано как XmlElement
Length	Int32	Длина тела объекта. Длина должна быть указана, если тело закодировано
Body	Byte	Тело объекта. Данное поле содержит необработанные байты для тел ByteString. Для тел XmlElement это поле содержит XML, закодированный как строка UTF-8 без нулевого символа-терминатора. Некоторые структуры с двоичным кодированием могут иметь сериализованную длину, не кратную 8 бит. Кодировщики должны добавлять 0 бит, чтобы гарантировать, что сериализованная длина кратна 8 бит. Декодеры, понимающие сериализованный формат, должны игнорировать биты заполнения

Объекты ExtensionObjects используются в двух контекстах: как значения, содержащиеся в структурах Variant, или как параметры в сообщениях OPC UA.

Декодер может выбрать анализ тела XmlElement после декодирования; если возникает неисправимая ошибка синтаксического анализа, декодер должен попытаться продолжить обработку потока. Например, если ExtensionObject является телом Variant или элементом массива, который является телом Variant, то об этой ошибке можно сообщить, установив значение Variant в StatusCode Bad_DecodingError.

5.2.2.16 Variant

Variant — объединение встроенных типов. Структура Variant представлена в таблице 15.

Таблица 15 — Структура Variant двоичного кодирования данных

Имя	Тип данных	Описание кодов полей
EncodingMask	Byte	Тип данных, закодированных в потоке. Значение 0 указывает NULL и отсутствие кодирования других полей. Маске присвоены следующие биты: 0:5 — встроенный идентификатор типа (см. таблицу 1); 6 — истинно, если поле «Размеры массива» закодировано; 7 — истинно, если закодирован массив значений. Встроенные идентификаторы типов с 26 по 31 в настоящее время не назначены, но могут использоваться в дальнейшем. Декодеры должны принимать встроенные идентификаторы, предполагая, что значение содержит строку байтов, и должны передавать в приложение. Встроенные кодировщики не должны использовать эти идентификаторы
ArrayLength	Int32	Количество элементов в массиве. Данное поле присутствует, если бит массива установлен в маске кодирования. Многомерные массивы кодируются как одномерный массив, и в этом поле указывается общее количество элементов. Исходный массив можно восстановить по измерениям, которые закодированы после поля значения. Измерения более высокого ранга сериализуются в первую очередь. Например, массив размерностями [2,2,2] записывается в таком порядке: [0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1]
Value	Не определено	Значение, закодированное в соответствии со встроенным типом данных. Если в маске кодирования установлен бит массива, то каждый элемент массива кодируется последовательно. Поскольку многие типы имеют кодировку переменной длины, каждый элемент должен декодироваться по порядку. Значение не должно быть вариантом, но может быть массивом вариантов. Многие платформы реализации не различают одномерные массивы байтов и байтовые строки. По этой причине декодерам разрешено автоматически преобразовывать массив байтов в ByteString
ArrayDimensions Length	Int32	Количество измерений. Это поле присутствует только в том случае, если в маске кодирования установлен флаг ArrayDimensions
ArrayDimensions	Int32	Длина каждого измерения, закодированная как последовательность значений Int32. Поле присутствует, если в маске кодирования установлен флаг ArrayDimensions. Измерения более низкого ранга появляются в массиве первыми. Все размеры должны быть указаны и должны быть более нуля. Если ArrayDimensions несовместимы с ArrayLength, декодер должен остановиться и вызвать Bad_DecodingError

Типы и их идентификаторы, которые допускается закодировать в Variant, показаны в таблице 1.

5.2.2.17 DataValue

DataValue неизменно предшествует маске, указывающей, какие поля присутствуют в потоке. Поля DataValue приведены в таблице 16.

Таблица 16 — Двоичное кодирование данных

Имя	Тип данных	Описание кодирования данных
Encoding Mask	Byte	Битовая маска, указывающая, какие поля присутствуют в потоке. Маска имеет следующие биты: 0x01 — False, если значение равно нулю; 0x02 — False, если код состояния хороший; 0x04 — False, если временная метка источника — DateTime.MinValue; 0x08 — False, если временная метка сервера — DateTime.MinValue; 0x10 — False, если исходные пикосекунды равны 0; 0x20 — False, если пикосекунды сервера равны 0
Value	Variant	Значение отсутствует, если бит Value в EncodingMask имеет значение False
Status	StatusCode	Статус, связанный со значением. Отсутствует, если бит StatusCode в EncodingMask имеет значение False
SourceTimestamp	DateTime	Исходная временная метка, связанная со значением. Отсутствует, если бит SourceTimestamp в EncodingMask имеет значение False
SourcePicoSeconds	UInt16	Число 10-пикосекундных интервалов для SourceTimestamp. Отсутствует, если бит SourcePicoSeconds в EncodingMask имеет значение False. Если временная метка источника отсутствует, пикосекунды игнорируются
ServerTimestamp	DateTime	Временная метка сервера, связанная со значением. Отсутствует, если бит ServerTimestamp в EncodingMask имеет значение False
ServerPicoSeconds	UInt16	Количество 10-пикосекундных интервалов для ServerTimestamp. Отсутствует, если бит ServerPicoSeconds в EncodingMask имеет значение False. Если временная метка сервера отсутствует, пикосекунды игнорируются

Поля Picoseconds хранят разницу между меткой времени высокого разрешения с разрешением 10 пс и значением поля Timestamp, которое имеет разрешение всего 100 нс. Поля Picoseconds должны содержать значения менее 10 000. Декодер должен обрабатывать значения, большие или равные 10 000, как значение 9999.

5.2.3 Десятичные

Десятичные дроби кодируются, как описано в 5.1.7. Десятичное число не имеет NULL.

5.2.4 Перечисления

Перечисления кодируются как значения Int32. Перечисление не имеет NULL.

5.2.5 Массивы

Одномерные массивы кодируются как последовательность элементов, которым предшествует количество элементов, закодированных как значение Int32. Если массив имеет значение NULL, его длина кодируется как −1. Массив нулевой длины отличается от массива, имеющего NULL, поэтому кодеры и декодеры должны сохранить такое различие.

Многомерные массивы кодируются как массив Int32, содержащий измерения, за которыми следует список всех значений в массиве. Общее количество значений равно произведению измерений. Количество значений равно 0, если одно или несколько измерений менее или равны 0.

Процесс восстановления многомерного массива описан в 5.2.2.16.

5.2.6 Структуры

Структуры кодируются как последовательность полей в том порядке, в котором они появляются в определении. Кодировка каждого поля определяется встроенным типом поля.

Все поля, указанные в структуре, должны быть закодированы. Информация о необязательных полях в структуре приведена в 5.2.7.

Структуры не имеют NULL. Если кодировщик написан на языке программирования, который позволяет структурам иметь NULL, тогда кодировщик должен создать новый экземпляр со значениями по

умолчанию для всех полей и сериализовать его. В этой ситуации кодировщики не должны генерировать ошибку кодирования.

Ниже приведен пример структуры с использованием синтаксиса C/C++:

```
struct Type2
{
Int32 A;
Int32 B;
};

struct Type1
{
Int32 X;
Byte NoOfY;
Type2* Y;
Int32 Z;
};
```

В приведенном выше примере C/C++ поле Y является указателем на массив, длина которого хранится в NoOfY.

При кодировании массива длина является частью кодировки массива, поэтому поле NoOfY не кодируется. Тем не менее кодеры и декодеры используют NoOfY во время кодирования.

Экземпляр Type1, который содержит массив из двух экземпляров Type2, будет закодирован как 28-байтовая последовательность. Если бы экземпляр Type1 был закодирован в ExtensionObject, он бы имел дополнительный префикс, представленный в таблице 17, общая длина которого составит 37 байт. TypeId, Encoding и Length — это поля, определенные ExtensionObject. Кодировка экземпляров Type2 не включает какой-либо идентификатор типа, так как он явно определен в Type1.

Таблица 17 — Пример структуры двоичного кодирования OPC UA

Поле	Байты	Значения двоичных кодов OPC UA
Type Id	4	Идентификатор узла двоичного кодирования Type1
Encoding	1	0x1 для ByteString
Length	4	28
X	4	Значение поля 'X'
Y.Length	4	2
Y.A	4	Значение поля 'Y[0].A'
Y.B	4	Значение поля 'Y[0].B'
Y.A	4	Значение поля 'Y[1].A'
Y.B	4	Значение поля 'Y[1].B'
Z	4	Значение поля 'Z'

Значения атрибута DataTypeDefinition для узла DataType, описывающего Type1 и Type 2, приведены в таблицах 18 и 19.

Таблица 18 — Характеристика атрибута DataTypeDefinition для узла DataType, описывающего Type1

Имя	Тип	Описание атрибута
defaultEncodingId	NodeId	NodeId узла «Type1_Encoding_DefaultBinary»
baseDataType	NodeId	“i = 22” [Structure]
structureType	StructureType	Structure_0 [Structure без опциональных полей]
fields [0]:	StructureField:	—

Окончание таблицы 18

Имя	Тип	Описание атрибута
name	String	"X"
description	LocalizedText	Описание X
dataType	NodeId	"i = 6" [Int32]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	False
fields [1]:	StructureField:	—
name	String	"Y"
description	LocalizedText	Описание Y-массива
dataType	NodeId	NodeId узла Type2 DataType (e.g. "ns = 3; s = MyType2")
valueRank	Int32	1 (OneDimension)
isOptional	Boolean	False
fields [2]:	StructureField:	—
name	String	"Z"
description	LocalizedText	Описание Z
dataType	NodeId	"i = 6" [Int32]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	False

Таблица 19 — Характеристика атрибута DataTypeDefinition для узла DataType, описывающего Type2

Имя	Тип	Описание атрибута
defaultEncodingId	NodeId	NodeId узла "Type2_Encoding_DefaultBinary"
baseDataType	NodeId	"i = 22" [Structure]
structureType	StructureType	Structure_0 [Structure без опциональных полей]
fields [0]:	StructureField:	—
name	String	"A"
description	LocalizedText	Описание A
dataType	NodeId	"i = 6" [Int32]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	False
fields [1]:	StructureField:	—
name	String	"B"
description	LocalizedText	Описание B
dataType	NodeId	"i = 6" [Int32]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	False

5.2.7 Структуры с необязательными полями

Структуры с необязательными полями кодируются с помощью маски кодирования, предшествующей последовательности полей в том порядке, в котором они появляются в определении. Кодировка каждого поля соотносится с типом данных поля.

Encoding Mask — 32-битное целое число без знака. Каждому необязательному полю назначается ровно 1 бит. Первому необязательному полю присваивается бит 0, второму обязательному полю назначается бит 1 и т. д., пока всем необязательным полям не будут присвоены биты. В одной структуре допускается максимум 32 необязательных поля. Неназначенные биты устанавливаются кодировщиками в значение 0. Декодеры должны сообщать об ошибке, если неназначенные биты не равны 0.

Пример структуры с необязательными полями, использующей синтаксис C++:

```
struct TypeA
{
Int32 X;
Int32* O1; SByte Y; Int32* O2;
};
```

где O1 и O2 — необязательные поля, которые имеют значение NULL, если они отсутствуют.

Экземпляр TypeA, который содержит два обязательных (X и Y) и два необязательных (O1 и O2) поля, будет закодирован как последовательность байтов. Длина последовательности байтов зависит от доступных необязательных полей. Поле маски кодирования определяет доступные дополнительные поля.

Экземпляр TypeA, где поле O2 доступно, а поле O1 недоступно, будет закодирован как 13-байтовая последовательность. Если экземпляр TypeA закодирован в ExtensionObject, то он будет иметь закодированную форму, представленную в таблице 20, и общую длину 22 байта. Длина TypeId, Encoding и Length — это атрибуты полей, определенных ExtensionObject.

Таблица 20 — Пример двоичной кодированной структуры OPC UA с необязательными полями

Поле	Bytes	Значение атрибута
Type Id	4	Идентификатор узла двоичного кодирования TypeA
Encoding	1	0x1 для ByteString
Length	4	13
EncodingMask	4	0x02 для O2
X	4	Значение X
Y	1	Значение Y
O2	4	Значение O2

Если структура с необязательными полями имеет подтип, подтипы расширяют EncodingMask, определенную для родительского элемента.

Значение атрибута DataTypeDefinition для узла DataType, описывающего TypeA, представлено в таблице 21.

Таблица 21 — Характеристика атрибута DataTypeDefinition для узла DataType, описывающего TypeA

Имя	Тип	Описание атрибута
defaultEncodingId	NodeId	NodeId узла "TypeA_Encoding_DefaultBinary"
baseDataType	NodeId	"i = 22" [Structure]
structureType	StructureType	StructureWithOptionalFields_1 [Structure без опциональных полей]
fields [0]:	StructureField:	—
name	String	"X"
description	LocalizedText	Описание X

Окончание таблицы 21

Имя	Тип	Описание атрибута
dataType	NodeId	"i = 6" [Int32]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	False
fields [1]:	StructureField:	—
name	String	"O1"
description	LocalizedText	Описание O1
dataType	NodeId	"i = 6" [Int32]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	True
fields [2]:	StructureField:	—
name	String	"Y"
description	LocalizedText	Описание Z
dataType	NodeId	"i = 2" [SByte]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	False
fields [3]:	StructureField:	—
name	String	"O2"
description	LocalizedText	Описание O2
dataType	NodeId	"i = 6" [Int32]
valueRank	Int32	–1 (Scalar)
isOptional	Boolean	True

5.2.8 Объединения

Unions кодируются как поле переключения, предшествующее одному из возможных полей. Кодировка выбранного поля определяется типом данных поля. Поле переключателя кодируется как UInt32.

Поле переключателя — индекс доступных полей объединения начиная с 1. Если поле переключателя равно 0, то поле отсутствует. Для любого значения, превышающего количество определенных полей объединения, кодеры и декодеры должны сообщить об ошибке.

Unions без полей имеет такое же значение, как и значение NULL. Объединение с любым присутствующим полем не является значением NULL, даже если значение самого поля равно NULL.

Пример Unions с использованием синтаксиса C/C++:

```
struct Type2
{
Int32 A;
Int32 B;
};

struct Type1
{
Byte Selector;

union
{
Int32 Field1;
```

```
Type2 Field2;
}
Value;
};
```

В приведенном примере C/C++ Selector и Value семантически связаны, образуя объединение. Порядок полей не имеет значения.

Экземпляр Type1 будет закодирован как последовательность байтов. Длина последовательности байтов зависит от выбранного поля.

Экземпляр Type1, где доступно поле Field1, будет закодирован как 8-байтовая последовательность. Если бы экземпляр типа 1 был закодирован в ExtensionObject, он имел бы закодированную форму, представленную в таблице 22, и общую длину 17 байт. TypeId, Encoding и Length — это поля, определенные ExtensionObject.

Таблица 22 — Пример двоичной кодированной структуры OPC UA

Поле	Bytes	Описание полей
Type Id	4	Идентификатор для Type1
Encoding	1	0x1 для ByteString
Length	4	8
SwitchValue	4	1 для Field1
Field1	4	Значение Field1

5.2.9 Сообщения

Messages — структуры, закодированные как последовательность байтов с префиксом NodeId для двоичного кодирования типа данных OPC UA, определенного для сообщения.

Каждая служба OPC UA согласно ГОСТ Р 71809 имеет сообщение запроса и ответа. Идентификаторы DataTypeEncoding, назначенные каждой Service, указаны в А.3.

5.3 OPC UA XML

5.3.1 Встроенные типы

5.3.1.1 General

Большинство встроенных типов закодированы в XML с использованием форматов, определенных в спецификации схемы XML, часть 2. Требование учета особых ограничений определило возможность применения XML-схемы с синтаксисом, определенным в схемах XML, часть 2.

Префикс xs используется для обозначения символа, определенного спецификацией схемы XML.

5.3.1.2 Boolean

Логическое значение кодируется как значение xs:boolean.

5.3.1.3 Integer

Целочисленные значения кодируются с использованием одного из подтипов типа xs:decimal. Сопоставления между целочисленными типами OPC UA и типами данных схемы XML приведены в таблице 23.

Таблица 23 — Сопоставления типов данных схемы XML для целых чисел

Имя	Тип данных XML
SByte	xs:byte
Byte	xs:unsignedByte
Int16	xs:short
UInt16	xs:unsignedShort
Int32	xs:int
UInt32	xs:unsignedInt
Int64	xs:long
UInt64	xs:unsignedLong

5.3.1.4 Floating Point

Значения с плавающей запятой кодируются с использованием одного из типов XML с плавающей запятой. Сопоставления между типами с плавающей запятой OPC UA и типами данных схемы XML представлены в таблице 24.

Таблица 24 — Сопоставления типов данных схемы XML с плавающей запятой

Имя	Тип данных XML
Float	xs:float
Double	xs:double

Тип XML с плавающей запятой поддерживает положительную бесконечность (INF), отрицательную бесконечность (–INF) и нечисловое число (NaN).

5.3.1.5 String

Значение String кодируется как значение xs:string.

5.3.1.6 DateTime

Значение DateTime кодируется как значение xs:dateTime.

Все значения DateTime должны быть закодированы как время UTC или с явно указанным часовым поясом.

Правильное кодирование:

2002-10-10T00:00:00+05:00;

2002-10-09T19:00:003.

Неверное кодирование:

2002-10-09T19:00:00.

Рекомендуется, чтобы все значения xs:dateTime были представлены в формате UTC.

Наиболее ранние и наиболее поздние значения даты/времени, которые могут быть представлены на платформе разработки, имеют особое значение и не должны кодироваться в XML.

Наиболее раннее значение даты/времени на платформе разработки должно быть закодировано в XML как «0001-01-01T00:00:00Z».

Последнее значение даты/времени на платформе разработки должно быть закодировано в XML как ‘9999-12-31T23:59:593’.

Если декодер идентифицирует значение xs:dateTime, которое не может быть представлено на платформе разработки, он должен преобразовать данное значение в самое раннее или самое позднее значение даты/времени, которые возможно представить на платформе разработки. Если обнаружено значение даты, выходящее за пределы допустимого диапазона, декодер XML не должен генерировать ошибку.

Самое раннее значение даты/времени на платформе разработки эквивалентно NULL даты/времени.

5.3.1.7 Guid

Guid кодируется с использованием строкового представления, определенного в 5.1.3.

Схема XML для Guid:

```
<xs:complexType name=”Guid”>
<xs:sequence>
<xs:element name=”String” type=”xs:string” minOccurs=”0”/>
</xs:sequence>
</xs:complexType>
```

5.3.1.8 ByteString

Значение ByteString кодируется как значение xs:base64Binary.

Схема XML для ByteString:

```
<xs:element name=”ByteString” type=”xs:base64Binary” nillable=”true”/>
```

5.3.1.9 XmlElement

Значение XmlElement кодируется как xs:complexType со следующей схемой XML:

```
<xs:complexType name=”XmlElement”>
<xs:sequence>
<xs:any minOccurs=”0” maxOccurs=”1” processContents=”lax”/>
```

</xs:sequence>
</xs:complexType>
XmlElements можно использовать только внутри значений Variant или ExtensionObject.
5.3.1.10 NodeId
Значение NodeId кодируется как xs:string с синтаксисом:
ns=<namespaceindex>;<type>=<value>.
Элементы синтаксиса приведены в таблице 25.

Таблица 25 — Компоненты NodeId

Поле	Тип данных	Описание полей NodeId
<namespaceindex>	UInt16	NamespaceIndex отформатировано как число по основанию 10. Если индекс равен 0, то все 'ns=0;', пункт опускается
<type>	Enumeration	Флаг, указывающий IdentifierType. Флаг имеет следующие значения: i — NUMERIC (UInt32); s — STRING (String); g — GUID (Guid); b — OPAQUE (ByteString)
<value>	Enumeration	Идентификатор, закодированный в виде строки. Идентификатор форматируется с использованием сопоставления типов данных XML для IdentifierType. Следует иметь в виду, что идентификатор может содержать любой ненулевой символ UTF-8, включая пробелы

Например, синтаксис NodeIds может быть следующим:
i=13
ns=10; i=-1
ns=10; s=Hello:World
g=09087e75-8e5e-499b-954f-f2a9603db28a
ns=1; b=M/RbKBsRVkePCePcx24oRA==
Схема XML дляNodeId:
<xs:complexType name="NodeId">
<xs:sequence>
<xs:element name="Identifier" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
5.3.1.11 ExpandedNodeId
Значение ExpandedNodeId кодируется как строка xs со следующим синтаксисом:
svr=<serverindex>;ns=<namespaceindex>;<type>=<value>
или
svr=<serverindex>;nsu=<uri>;<type>=<value>.
Возможные поля приведены в таблице 26.

Таблица 26 — Компоненты ExpandedNodeId

Поле	Тип данных	Описание полей ExpandedNodeId
<serverindex>	UInt32	ServerIndex отформатировано как число по основанию 10. Если ServerIndex равно 0, то весь 'svr=0;', пункт опускается
<namespaceindex>	UInt16	NamespaceIndex отформатировано как число по основанию 10. Если NamespaceIndex равно 0, то все 'ns=0;', пункт опускается. NamespaceIndex не должно присутствовать, если присутствует URI

Окончание таблицы 26

Поле	Тип данных	Описание полей ExpandedNodeId
<uri>	String	NamespaceUri в формате строки. Любые зарезервированные символы в URI должны быть заменены знаком «%», за которым следует его 8-битовое значение ANSI, закодированное в виде двух шестнадцатеричных цифр (без учета регистра). Например, символ ';' будет заменен на «%3B». Зарезервированные символы: ';' и '%'. Если NamespaceUri имеет значение NULL или пустое, тогда 'nsu=;', пункт опускается
<type>	Enumeration	Флаг, указывающий IdentifierType. Это поле описано в таблице 25

Схема XML для ExpandedNodeId:

```
<xs:complexType name="ExpandedNodeId">
<xs:sequence>
<xs:element name="Identifier" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

5.3.1.12 StatusCode

StatusCode кодируется как xs:unsignedInt по следующей схеме XML:

```
<xs:complexType name="StatusCode">
<xs:sequence>
<xs:element name="Code" type="xs:unsignedInt" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

5.3.1.13 DiagnosticInfo

Значение DiagnosticInfo кодируется как xs:complexType по следующей схеме XML:

```
<xs:complexType name="DiagnosticInfo">
<xs:sequence>
<xs:element name="SymbolicId" type="xs:int" minOccurs="0"/>
<xs:element name="NamespaceUri" type="xs:int" minOccurs="0"/>
<xs:element name="Locale" type="xs:int" minOccurs="0/>
<xs:element name="LocalizedText" type="xs:int" minOccurs="0/>
<xs:element name="AdditionalInfo" type="xs:string" minOccurs="0"/>
<xs:element name="InnerStatusCode" type="tns:StatusCode" minOccurs="0"/>
<xs:element name="InnerDiagnosticInfo" type="tns:DiagnosticInfo" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

DiagnosticInfo допускает неограниченное вложение, что может привести к ошибкам переполнения стека, даже если размер сообщения меньше максимально допустимого. Декодеры должны поддерживать не менее 100 уровней вложенности. Декодеры должны сообщать об ошибке, если количество уровней вложенности превышает поддерживаемое.

5.3.1.14 QualifiedName

Значение QualifiedName кодируется как xs:complexType по следующей схеме XML:

```
<xs:complexType name="QualifiedName">
<xs:sequence>
<xs:element name="NamespaceIndex" type="xs:int" minOccurs="0"/>
<xs:element name="Name" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

5.3.1.15 LocalizedText

Значение LocalizedText кодируется как xs:complexType по следующей схеме XML:

```
<xs:complexType name="LocalizedText">
<xs:sequence>
<xs:element name="Locale" type="xs:string" minOccurs="0"/>
```

<xs:element name=”Text” type=”xs:string” minOccurs=”0”/>
</xs:sequence>
</xs:complexType>

5.3.1.16 ExtensionObject

Значение ExtensionObject кодируется как xs:complexType по следующей схеме XML:

<xs:complexType name=”ExtensionObject”>
<xs:sequence>
<xs:element name=”TypeId” type=”tns:NodeId” minOccurs=”0”/>
<xs:element name=”Body” minOccurs=”0”>
<xs:complexType>
<xs:sequence>
<xs:any minOccurs=”0” processContents=”lax”/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

Тело ExtensionObject содержит один элемент, который представляет собой структуру в кодировке ByteString или XML. Декодер может отличить их, проверив элемент верхнего уровня. Элемент с именем tns:ByteString содержит тело в двоичной кодировке OPC UA. Любое другое имя должно содержать тело, закодированное в формате XML OPC UA. TypeId указывает синтаксис тела ByteString, который может быть JSON в кодировке UTF-8, двоичным UA или каким-либо другим форматом.

TypeId — это NodeId для объекта DataTypeEncoding.

5.3.1.17 Variant

Значение Variant кодируется как xs:complexType по следующей схеме XML:

<xs:complexType name=”Variant”>
<xs:sequence>
<xs:element name=”Value” minOccurs=”0” nillable=”true”>
<xs:complexType>
<xs:sequence>
<xs:any minOccurs=”0” processContents=”lax”/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

Если Variant представляет скалярное значение, он должен содержать один дочерний элемент с именем встроенного типа. Например, значение с плавающей запятой одинарной точности 3,1415 будет закодировано как:

<tns:Float>3.1415</tns:Float>

Если Variant представляет собой одномерный массив, он должен содержать один дочерний элемент с префиксом ListOf и именем встроенного типа. Например, Array будет закодирован как:

<tns:ListOfString>
<tns:String>Hello</tns:String>
<tns:String>World</tns:String>
</tns:ListOfString>

Если вариант представляет собой многомерный массив, он должен содержать дочерний элемент с именем Matrix с двумя подэлементами, показанными в примере:

<tns:Matrix>
<tns:Dimensions>
<tns:Int32>2</tns:Int32>
<tns:Int32>2</tns:Int32>
</tns:Dimensions>
<tns:Elements>
<tns:String>A</tns:String>
<tns:String>B</tns:String>

<tns:String>C</tns:String>
<tns:String>D</tns:String>
</tns:Elements>
</tns:Matrix>

В приведенном примере массив состоит из элементов:

[0,0] = «A»; [0,1] = «B»; [1,0] = «C»; [1,1] = «D»

Элементы многомерного массива неизменно сводятся в одномерный массив, где измерения более высокого ранга сериализуются первыми. Полученный одномерный массив кодируется как дочерний элемент элемента Elements. Элемент Dimensions представляет собой массив Int32 значений, определяющих размеры массива, начиная с измерения самого низкого ранга. Многомерный массив допустимо реконструировать, используя закодированные измерения. Все размеры должны быть указаны и должны быть больше нуля. Если размеры не соответствуют количеству элементов в массиве, декодер должен остановиться и выдать ошибку Bad_DecodingError.

Полный набор имен встроенных типов приведен в таблице 1.

5.3.1.18 DataValue

Значение DataValue кодируется как xs:complexType по следующей схеме XML:

<xs:complexType name="DataValue">
<xs:sequence>
<xs:element name="Value" type="tns:Variant" minOccurs="0" nillable="true"/>
<xs:element name="StatusCode" type="tns:StatusCode" minOccurs="0"/>
<xs:element name="SourceTimestamp" type="xs:dateTime" minOccurs="0"/>
<xs:element name="SourcePicoseconds" type="xs:unsignedShort" minOccurs="0"/>
<xs:element name="ServerTimestamp" type="xs:dateTime" minOccurs="0"/>
<xs:element name="ServerPicoseconds" type="xs:unsignedShort" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

5.3.2 Десятичные

Decimal закодировано как xs:complexType по следующей схеме XML:

<xs:complexType name="Decimal">
<xs:sequence>
<xs:element name="TypeId" type="tns:NodeId" minOccurs="0"/>
<xs:element name="Body" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:element name="Scale" type="xs:unsignedShort"/>
<xs:element name="Value" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

NodeId — неизменно десятичный тип данных. При кодировании в Variant десятичное число кодируется как ExtensionObject. Массивы десятичных знаков являются массивами объектов расширения. Их значения записываются как целые числа со знаком по основанию 10 без ограничений по размеру (см. 5.1.7 для описания полей Scale и Value).

5.3.3 Перечисления

Enumerations, которые используются в качестве параметров в сообщениях согласно ГОСТ Р 71809, кодируются как xs:string с синтаксисом:

<symbol>_<value>

Элементы синтаксиса представлены в таблице 27.

Таблица 27 — Компоненты перечисления

Поле	Тип	Описание элементов синтаксиса
<symbol>	String	Символическое имя перечисляемого значения
<value>	UInt32	Числовое значение, связанное с перечисляемым значением

Например, схема XML для перечисления Node Class:

<xs:simpleType name="NodeClass">
<xs:restriction base="xs:string">
<xs:enumeration value="Unspecified_0"/>
<xs:enumeration value="Object_1"/>
<xs:enumeration value="Variable_2"/>
<xs:enumeration value="Method_4"/>
<xs:enumeration value="ObjectType_8"/>
<xs:enumeration value="VariableType_16"/>
<xs:enumeration value="ReferenceType_32"/>
<xs:enumeration value="DataType_64"/>
<xs:enumeration value="View_128"/>
</xs:restriction>
</xs:simpleType>

Перечисления, хранящиеся в Variant, кодируются как значение Int32. Например, любая переменная может иметь значение с типом данных NodeClass. В этом случае соответствующее числовое значение помещается в Variant (например, объект NodeClass будет сохранен как 1).

5.3.4 Массивы

Параметры одномерного массива неизменно кодируются путем упаковки элементов в элемент-контейнер и вставки контейнера в структуру. Имя элемента контейнера должно совпадать с именем параметра. Имя элемента в массиве должно быть именем типа.

Например, если служба чтения принимает массив ReadValueIds, схема XML будет представлена следующим образом:

<xs:complexType name="ListOfReadValueId">
<xs:sequence>
<xs:element name="ReadValueId" type="tns:ReadValueId" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
</xs:sequence>
</xs:complexType>

Атрибут nillable должен быть указан, так как кодировщики XML будут удалять элементы в массивах, если эти элементы пустые. Параметры многомерного массива кодируются с использованием типа Matrix, определенного в 5.3.1.17.

5.3.5 Структуры

Структуры кодируются как xs:complexType со всеми полями, расположенными в определенной последовательности.

Все поля кодируются как элемент xs:element. Для всех элементов параметр minOccurs установлен в значение 0, чтобы обеспечить компактное представление XML. Если элемент отсутствует, используется значение по умолчанию для типа поля. Если тип поля — это структура, то значением по умолчанию является экземпляр структуры со значениями по умолчанию для каждого содержащегося поля.

Типы, для которых определено значение NULL, должны иметь установленный флаг nillable="true".

Например, если служба чтения имеет в запросе структуру ReadValueId, то схема XML будет представлена следующим образом:

<xs:complexType name="ReadValueId">
<xs:sequence>
<xs:element name="NodeId" type="tns:NodeId"minOccurs="0" nillable="true"/>
<xs:element name="AttributeId" type="xs:int" minOccurs="0"/>
<xs:element name="IndexRange" type="xs:string" minOccurs="0" nillable="true"/>
<xs:element name="DataEncoding" type="tns:NodeId" minOccurs="0" nillable="true"/>
</xs:sequence>
</xs:complexType>

5.3.6 Структуры с необязательными полями

Структуры с необязательными полями кодируются как xs:complexType, все поля появляются последовательно. Первым элементом является битовая маска, определяющая кодируемые поля. Биты маски последовательно назначаются необязательным полям в порядке, в котором они появляются в структуре.

Для обеспечения компактного XML, в XML может быть пропущено любое поле, поэтому декодеры должны назначать значения по умолчанию на основе типа поля для всех обязательных полей.

Например, если следующая структура имеет одно обязательное и два необязательных поля, то схема XML будет представлена следующим образом:

```
<xs:complexType name="OptionalType">
<xs:sequence>
<xs:element name="EncodingMask" type="xs:unsignedLong"/>
<xs:element name="X" type="xs:int" minOccurs="0"/>
<xs:element name="O1" type="xs:int" minOccurs="0"/>
<xs:element name="Y" type="xs:byte" minOccurs="0"/>
<xs:element name="O2" type="xs:int" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

В приведенном примере EncodingMask имеет значение 3, если закодированы и O1, и O2. Кодеры должны устанавливать неиспользуемые биты в значение 0, а декодеры должны игнорировать неиспользуемые биты.

Если структура с необязательными полями имеет подтип, подтипы расширяют EncodingMask, определенную для родительского элемента.

5.3.7 Объединения

Unions кодируются как xs:complexType, содержащий xs:sequence с двумя записями.

Первый элемент имеет значение SwitchField, равное 1, а последнее значение имеет SwitchField, равное количеству вариантов выбора.

Дополнительные элементы в последовательности не допускаются. Если SwitchField отсутствует или равен 0, то объединение имеет значение NULL. Кодеры и декодеры должны сообщать об ошибке для любого поля Switch Field значение большее, чем количество определенных полей объединения.

Например, если следующий Unions имеет два поля, то схема XML должна быть представлена таким образом:

```
<xs:complexType name="Type1">
<xs:sequence>
<xs:element name="SwitchField" type="xs:unsignedInt" minOccurs="0"/>
<xs:choice>
<xs:element name="Field1" type="xs:int" minOccurs="0"/>
<xs:element name="Field2" type="tns:Field2" minOccurs="0"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
```

5.3.8 Сообщения

Messages кодируются как xs:complexType. Параметры в каждом Message сериализуются так же, как и поля Structure.

5.4 OPC UA JSON

5.4.1 Общие положения

Кодирование данных OPC UA JSON дает возможность приложениям OPC UA взаимодействовать с веб-приложениями и корпоративным программным обеспечением, использующим предлагаемый формат. Кодирование данных OPC UA JSON определяет стандартные представления JSON для всех встроенных типов OPC UA.

Формат JSON приведен в [2]. Данный формат частично самоописателен, поскольку в дополнение к значению в каждом поле закодировано имя; однако в JSON отсутствует механизм для уточнения имен с помощью пространств имен.

Формат JSON не имеет опубликованного стандарта схемы, которую можно использовать для описания содержимого документа JSON. Механизмы схемы, определенные в [2], допускается применять для описания документов JSON, в частности DataTypeDescription.

Структура, приведенная в ГОСТ Р 71808, может определять любой документ JSON, соответствующий правилам, описанным ниже.

Серверы, поддерживающие кодировку данных JSON, должны добавлять узлы DataTypeEncoding, называемые «JSON по умолчанию», ко всем типам данных, которые сериализуются с помощью кодировки JSON. NodeId этих узлов определены информационной моделью, которая устанавливает тип данных.

NodeIds применяются ExtensionObjects, как описано в 5.4.2.16.

Применяются два важных варианта кодировки JSON: облачные приложения, которые используют сообщения PubSub, и клиенты JavaScript (JSON — предпочтительный формат сериализации для JavaScript). В случае применения облачного приложения сообщение PubSub должно быть самодостаточным, что означает, что оно не может содержать числовые ссылки на внешне определенную таблицу пространства имен. В облачных приложениях для обработки входящих сообщений допустимо использование языков сценариев, поэтому для обеспечения точности во время декодирования в DataEncoding не требуется применения артефактов. Данное DataEncoding определяет необратимую форму, предназначенную для удовлетворения потребностей облачных приложений. Приложения, такие как клиенты JavaScript, которые используют DataEncoding для связи с другими приложениями OPC UA, применяют обычный или обратимый формат. Различия между обратимыми и необратимыми формами описаны для каждого типа.

5.4.2 Built-in Types

5.4.2.1 General

Значение встроенного типа, равное NULL, должно быть закодировано как литерал JSON «null», если оно является элементом массива. Если значение NULL — это поле внутри структуры или объединения, то данное поле не должно кодироваться.

5.4.2.2 Boolean

Логическое значение должно быть закодировано как литерал JSON — «Truth» или «False».

5.4.2.3 Integer

Целочисленные значения, отличные от Int64 и UInt64, должны кодироваться как номер JSON.

Значение Int64 и UInt64 должны быть отформатированы как десятичное число, закодированное в виде строки JSON.

5.4.2.4 Floating point

Обычные значения Float и Double должны быть закодированы как число JSON. Специальные числа с плавающей запятой, такие как положительная бесконечность (INF), отрицательная бесконечность (–INF) и нечисловые числа (NaN), должны быть представлены значениями «Infinity», «–Infinity» и «NaN», закодированными как строки JSON.

5.4.2.5 String

Строковые значения должны быть закодированы как строки JSON. Любые символы, недопустимые в строках JSON, экранируются с использованием правил, определенных в [2].

5.4.2.6 DateTime

Значения DateTime должны быть отформатированы (см. [3]) и закодированы как строка JSON.

Значения DateTime, которые превышают минимальные или максимальные значения, поддерживаемые платформой, должны быть закодированы как «0001-01-01T00:00:00Z» или «9999-12-31T23:59:59Z» соответственно.

Во время декодирования эти значения преобразуются в минимальные или максимальные значения, поддерживаемые платформой. Значения DateTime, равные «0001-01-01T00:00:00Z», считают значениями NULL.

5.4.2.7 Guid

Значения Guid должны быть отформатированы, как описано в 5.1.3, и закодированы как строка JSON.

5.4.2.8 ByteString

Значения ByteString должны быть отформатированы как текст Base64 и закодированы как строка JSON.

Любые символы, недопустимые в строках JSON, экранируются с использованием правил (см. [2]).

5.4.2.9 XmlЭлемент

Значение XmlElement должно быть закодировано как строка (см. 5.3.1.9).

5.4.2.10 NodeId

Значения NodeId должны быть закодированы как объект JSON с полями, приведенными в таблице 28.

Абстрактная структура NodeId (см. ГОСТ Р 71808) имеет три абстрактных поля: Identifier, IdentifierType и NamespaceIndex, представленные в таблице 28.

Таблица 28 — Поля, определенные объектом JSON для NodeId

Имя	Описание полей
IdType	IdentifierType закодирован как номер JSON. Допустимые значения: 0 — идентификатор UInt32, закодированный как номер JSON; 1 — строковый идентификатор, закодированный как строка JSON; 2 — идентификатор Guid, закодированный согласно описанию в 5.4.2.7; 3 — идентификатор ByteString, закодированный согласно описанию в 5.4.2.8. Для идентификаторов UInt32 представление поля опускается
Id	Идентификатор. Значение поля id определяет кодировку этого поля
Namespace	NamespaceIndex для NodeId. Поле кодируется как номер JSON для обратимого кодирования. Поле опускается, если NamespaceIndex равен 0. Для необратимого кодирования полем является NamespaceUri, связанное с NamespaceIndex, закодированным как строка JSON. NamespaceIndex, равный 1, неизменно кодируется как номер JSON

5.4.2.11 ExpandedNodeId

Значения ExpandedNodeId должны быть закодированы как объект JSON с полями, определенными в таблице 29. Абстрактная структура ExpandedNodeId (см. ГОСТ Р 71808) имеет пять полей: Identifier, IdentifierType, NamespaceIndex, NamespaceUri и ServerIndex. Их представление приведено в таблице 29.

Таблица 29 — Поля, определенные объектом JSON для ExpandedNodeId

Имя	Описание полей
IdType	IdentifierType закодирован как номер JSON. Допустимые значения: 0 — идентификатор UInt32, закодированный как номер JSON; 1 — строковый идентификатор, закодированный как строка JSON; 2 — идентификатор Guid, закодированный согласно описанию в 5.4.2.7; 3 — идентификатор ByteString, закодированный согласно описанию в 5.4.2.8. Для идентификаторов UInt32 поле опускается
Id	Идентификатор. Значение поля t определяет кодировку этого поля
Namespace	NamespaceIndex или NamespaceUri для ExpandedNodeId. Если NamespaceUri не указано, NamespaceIndex кодируется по следующим правилам: - поле кодируется как номер JSON для обратимого кодирования; - поле опускается, если NamespaceIndex равен 0. Для необратимого кодирования полем является NamespaceUri, связанное с NamespaceIndex, закодированный как строка JSON. NamespaceIndex, равный 1, неизменно кодируется как номер JSON. Если указан NamespaceUri, в этом поле он кодируется как строка JSON
ServerUri	ServerIndex для ExpandedNodeId. Это поле кодируется как номер JSON для обратимого кодирования и опускается, если ServerIndex равен 0. Для необратимого кодирования это поле представляет собой ServerUri, связанный с частью ServerIndex ExpandedNodeId, закодированный как строка JSON

5.4.2.12 NodeId

Значения StatusCode должны быть закодированы как номер JSON для обратимого кодирования.

Для необратимой формы значения StatusCode должны быть закодированы как объект JSON с полями, определенными в таблице 30.

Таблица 30 — Определение объекта JSON для StatusCode

Имя	Описание полей
Code	Числовой код, закодированный как число JSON. Код опускается, если числовой код равен 0 («хорошо»)
Symbol	Строковый литерал, связанный с числовым кодом, закодированным как строка JSON. Например, 0x80AB0000 имеет связанный с ним литерал BadInvalidArgument. Символ опускается, если числовой код равен 0 («хорошо»)

Код статуса Good (0) рассматривается как NULL и не кодируется. Если это элемент массива JSON, он кодируется как литерал «JSON».

5.4.2.13 DiagnosticInfo

Значения DiagnosticInfo должны быть закодированы как объект JSON с полями, представленными в таблице 31.

Таблица 31 — Определение объекта JSON для DiagnosticInfo

Имя	Тип данных	Описание полей
SymbolicId	Int32	Символическое имя для кода состояния
NamespaceUri	Int32	Пространство имен, определяющее символический идентификатор
Locale	Int32	Языковой стандарт, используемый для локализованного текста
LocalizedText	Int32	Удобочитаемая краткая информация о коде состояния
AdditionalInfo	String	Подробная диагностическая информация для конкретного приложения
InnerStatusCode	StatusCode	Код состояния, предоставляемый базовой системой
InnerDiagnosticInfo	DiagnosticInfo	Диагностическая информация, связанная с внутренним кодом состояния

Каждое поле кодируется с использованием правил, определенных для встроенного типа, указанного в графе «Тип данных».

Поля SymbolicId, NamespaceUri, Locale и LocalizedText кодируются как числа JSON, которые ссылаются на StringTable, содержащуюся в ResponseHeader.

5.4.2.14 QualifiedName

Значения QualifiedName должны быть закодированы как объект JSON с полями, показанными в таблице 32.

Абстрактная структура QualifiedName (см. ГОСТ Р 71808) имеет два поля: Name и NamespaceIndex. NamespaceIndex представлен полем Uri в объекте JSON.

Таблица 32 — Определение объекта JSON для квалифицированного имени

Имя	Описание полей
Name	Компонент Name QualifiedName
Uri	Компонент NamespaceIndex QualifiedName, закодированный как номер JSON. Поле Uri опускается, если NamespaceIndex равен 0. Для необратимой формы NamespaceUri, связанный с частью NamespaceIndex QualifiedName, кодируется как строка JSON, если только NamespaceIndex не равен 1 или если NamespaceUri неизвестен. В этих случаях NamespaceIndex кодируется как номер JSON

5.4.2.15 LocalizedText

Значения LocalizedText должны быть закодированы как объект JSON с полями, представленными в таблице 33. Абстрактная структура LocalizedText (см. ГОСТ Р 71808) имеет два поля: Text и Locale.

Таблица 33 — Определение объекта JSON для Localized Text

Имя	Описание полей
Locale	Часть Locale значений LocalizedText должна быть закодирована как строка JSON
Text	Текстовая часть значений LocalizedText должна быть закодирована как строка JSON

Для необратимой формы значение LocalizedText должно быть закодировано как строка JSON, содержащий текстовый компонент.

5.4.2.16 ExtensionObject

Значения ExtensionObject должны быть закодированы как объект JSON с полями, представленными в таблице 34.

Таблица 34 — Определение объекта JSON для ExtensionObject

Имя	Описание полей
TypeId	NodeId узла DataTypeEncoding, отформатированный с использованием правил по 5.4.2.10
Encoding	Формат поля «Тело», закодированный как номер JSON. Значение равно 0, если тело представляет собой структуру, закодированную как объект JSON (см. 5.4.6). Значение равно 1, если тело представляет собой значение ByteString, закодированное как строка JSON (см. 5.4.2.8). Значение равно 2, если тело представляет собой значение XmlElement, закодированное как строка JSON (см. 5.4.2.9). Это поле опускается, если значение равно 0
Body	Тело ExtensionObject. Тип этого поля определен полем «Кодировка». Если Body пусто, ExtensionObject имеет значение NULL и опускается или кодируется как NULL JSON

Для необратимой формы значения ExtensionObject должны быть закодированы как объект JSON, содержащий только значение поля Body. Поля Type и Encoding при этом удаляются.

5.4.2.17 Variant

Значения Variant должны быть закодированы как объект JSON с полями, приведенными в таблице 35.

Таблица 35 — Определение объекта JSON для Variant

Имя	Описание полей
Type	Встроенный тип значения, содержащегося в теле (см. таблицу 1), закодированного как номер JSON. Если тип равен NULL, Variant содержит значение NULL, а содержащий его объект JSON должен быть опущен или заменен литералом «JSON NULL»
Body	Если значение является скаляром, оно кодируется с использованием правил, указанных для Type. Если значение представляет собой одномерный массив, оно кодируется как массив JSON (см. 5.4.5). Многомерные массивы кодируются как одномерный массив JSON, который восстанавливается по значению поля «Измерения» (см. 5.2.2.16)
Dimensions	Размеры массива, закодированные как массив чисел JSON. Размеры опускаются для значений скалярных и одномерных массивов

Для необратимой формы значения Variant должны быть закодированы как объект JSON, содержащий только значение поля Body. Поля TypeId и Encoding удалены. Многомерные массивы кодируются как многомерный массив JSON (см. 5.4.5).

5.4.2.18 DataValue

Значения DataValue должны быть закодированы как объект JSON с полями, приведенными в таблице 36.

Таблица 36 — Определение объекта JSON для значения DataValue

Имя	Тип данных	Описание полей
Value	Variant	Значение
Status	StatusCode	Статус, связанный со значением
SourceTimestamp	DateTime	Исходная временная метка, связанная со значением
SourcePicoSeconds	UInt16	Число 10-пикосекундных интервалов для SourceTimestamp
ServerTimestamp	DateTime	Временная метка сервера, связанная со значением
ServerPicoSeconds	UInt16	Количество 10-пикосекундных интервалов для ServerTimestamp

Если поле имеет NULL или значение по умолчанию, оно опускается. Каждое поле кодируется с использованием правил, определенных для встроенного типа, указанного в графе «Тип данных» таблицы 36.

5.4.3 Десятичные

Десятичные значения должны быть закодированы как объект JSON с полями, представленными в таблице 37.

Таблица 37 — Определение объекта JSON для десятичной дроби

Имя	Описание полей
Scale	Число JSON со шкалой, примененной к значению
Value	Строка JSON со значением, закодированным как целое число со знаком по основанию 10 (см. XML-кодирование целочисленных значений, описанное в 5.3.1.3)

Описание полей Scale и Value приведено в 5.1.7.

5.4.4 Перечисления

Значения перечислений должны быть закодированы как номер JSON для обратимого кодирования.

В необратимой форме значения перечисления кодируются как литералы, а значение добавляется в виде строки JSON.

Формат строкового литерала:

<name>_<value>

где name — литерал перечисления, а value — числовое значение.

Если литерал неизвестен кодировщику, числовое значение кодируется как строка JSON.

5.4.5 Массивы

Одномерные массивы должны быть закодированы как массивы JSON.

Если элемент имеет значение NULL, этот элемент должен быть закодирован как JSON литерал NULL. В противном случае элемент кодируется в соответствии с правилами, определенными для типа. Многомерные массивы кодируются как вложенные массивы JSON. Внешний массив — это первое измерение, а самый внутренний массив — последнее измерение. Например, матрица

0 2 3
1 3 4

кодируется JSON как [[0, 2, 3], [1, 3, 4]].

5.4.6 Структуры

Структуры должны быть закодированы как объекты JSON. Если значение поля равно NULL, оно должно быть исключено из кодирования. Например, экземпляры структур:

```
struct Type2
{
Int32 A;
```

Int32 B;
Char* C;
};
struct Type1
{
Int32 X;
Int32 NoOfY;
Type2* Y;
Int32 Z;
представлены в JSON как:
{
“X”:1234,
“Y”:[{ “A”:1, “B”:2, “C”:”Hello” }, { “A”:3, “B”:4 }], “Z”:5678
}
где «C» пропущен во втором экземпляре Type2, поскольку он имеет значение NULL.

5.4.7 Структуры с необязательными полями

Структуры с необязательными полями должны быть закодированы как объекты JSON, как показано в таблице 38.

Таблица 38 — Определение объекта JSON для структуры с необязательными полями

Имя	Описание полей
EncodingMask	Битовая маска, указывающая, какие поля закодированы в структуре (см. 5.2.7). Эта маска кодируется как число JSON. Биты последовательно назначаются необязательным полям в том порядке, в котором они определены
<FieldName>	Структура полей кодируется в соответствии с правилами, определенными для их типа данных

В необратимой форме структуры с необязательными полями кодируются как структуры. Если структура с необязательными полями имеет подтип, то подтипы расширяют EncodingMask, определенную для родительского элемента. Ниже приведен пример структуры с необязательными полями, использующей синтаксис C++:

struct TypeA
{
Int32 X;
Int32* O1;
SByte Y;
Int32* O2;
};

O1 и O2 — необязательные поля, значение NULL указывает на отсутствие поля. Например, O1 не указан и значение O2 равно 0.

При обратимом шифровании применяется запись:

{ ”EncodingMask”: 2, ”X”: 1, ”Y”: 2 }

где декодеры присвоили бы значение по умолчанию 0 для O2, так как бит маски установлен, если было опущено (поведение определено для Int32 DataType). Декодеры должны пометить O1 как «не указан».

5.4.8 Unions

Unions должны быть закодированы как объекты JSON, как представлено в таблице 39 для обратимого кодирования.

Таблица 39 — Определение объекта JSON для Unions

Имя	Описание полей
SwitchField	Идентификатор поля в Union, который закодирован как номер JSON
Value	Значение поля, закодированное с использованием правил, применимых к типу данных

В необратимой форме значения Union кодируются с применением правила для текущего значения. Экземпляры объединения:

```
struct Union1
{
Byte Selector;
{
Int32 A;
Double B;
Char* C;
}
Value;
};
```

в обратимой форме должны быть представлены как:

{ "SwitchField":2, "Value":3.1415 }.

В необратимом виде значения Union представляются как 3.1415.

5.4.9 Messages

Messages представляют собой закодированные объекты ExtensionObject (см. 5.4.2.16).

6 Message Security Protocols

6.1 Протокол безопасности

Все протоколы безопасности должны реализовывать услуги OpenSecureChannel и CloseSecureChannel (см. ГОСТ Р 71809). Эти услуги определяют, как установить SecureChannel и как применять безопасность к сообщениям, которыми обмениваются SecureChannel. Сообщения обмена и алгоритмы безопасности, примененные к ним, показаны на рисунке 10.

Протоколы безопасности должны поддерживать три режима безопасности: None, Sign и SignAndEncrypt. Если SecurityMode имеет значение None, то безопасность не используется и подтверждение безопасности, показанное на рисунке 10, не требуется. Однако реализация Security Protocol по-прежнему должна поддерживать логический канал и предоставлять уникальный идентификатор SecureChannel.

Рисунок 10 — Подтверждение безопасности

Каждое сопоставление SecurityProtocol точно определяет, как применить алгоритмы безопасности к сообщению. Набор алгоритмов безопасности, которые должны использоваться вместе во время подтверждения безопасности, называется SecurityPolicy. Стандартные политики безопасности как части стандартных профилей, которые, как предполагается, будут поддерживать приложения OPC UA, а также URI для каждой стандартной политики безопасности, приведены в [1]. Стек будет иметь встроенную информацию о политиках безопасности, которые он поддерживает. Приложения указывают SecurityPolicy, которую они хотят использовать, передавая URI в стек. В таблице 40 представлено содержимое SecurityPolicy. Каждое сопоставление SecurityProtocol определяет, как применять каждый из параметров в SecurityPolicy и не применять все параметры.

Таблица 40 — Политика безопасности

Имя	Описание применяемых алгоритмов
PolicyUri	URI, назначенный SecurityPolicy
SymmetricSignatureAlgorithm	Алгоритм симметричной подписи
SymmetricEncryptionAlgorithm	Алгоритм симметричного шифрования
AsymmetricSignatureAlgorithm	Алгоритм асимметричной подписи
AsymmetricEncryptionAlgorithm	Алгоритм асимметричного шифрования
MinAsymmetricKeyLength	Минимальная длина асимметричного ключа в битах
MaxAsymmetricKeyLength	Максимальная длина асимметричного ключа в битах
KeyDerivationAlgorithm	Используемый алгоритм получения ключа

Окончание таблицы 40

Имя	Описание применяемых алгоритмов
DerivedSignatureKeyLength	Длина в битах производного ключа, используемого для аутентификации сообщения
CertificateSignatureAlgorithm	Алгоритм асимметричной подписи, используемый для подписи сертификатов
SecureChannelNonceLength	Длина в байтах одноразовых номеров, которыми обмениваются при создании SecureChannel

KeyDerivationAlgorithm используется для создания ключей, предназначенных для защиты сообщений, отправляемых через SecureChannel. Длина ключей, используемых для шифрования, определяется алгоритмом SymmetricEncryptionAlgorithm; длина ключей, применяемых для создания подписей, — параметром DerivedSignatureKeyLength.

MinAsymmetricKeyLength и MaxAsymmetricKeyLength — это ограничения, которые применяют ко всем сертификатам (включая эмитенты в цепочке). Длина ключа выдаваемых сертификатов должна быть меньше или равна длине ключа сертификата эмитента (см. 6.2.3 для получения информации о цепочках сертификатов).

CertificateKeyAlgorithm и EphemeralKeyAlgorithm используют для создания новых пар асимметричных ключей с сертификатами и во время установления связи SecureChannel. В [1] определена длина битов, которая должна поддерживаться для каждой политики безопасности.

CertificateSignatureAlgorithm применяется к сертификату и всем сертификатам эмитента. Если CertificateSignatureAlgorithm допускает применение более одного алгоритма, то алгоритмы будут перечислены в порядке возрастания приоритета. Каждый эмитент в цепочке должен иметь алгоритм, который имеет тот же или более высокий приоритет, чем любой выдаваемый им сертификат.

SecureChannelNonceLength определяет длину одноразовых номеров, которыми обмениваются при установлении SecureChannel (см. 6.7.4).

6.2 Сертификаты

6.2.1 Общие сведения

Приложения OPC UA используют сертификаты для хранения открытых ключей, необходимых для операций асимметричной криптографии. Все протоколы безопасности применяют сертификаты X.509 v3 (см. X.509 v3), закодированные в формате DER (см. X690). Сертификаты, используемые приложениями OPC UA, должны также соответствовать установленным нормам (см. [4], в котором определен профиль для сертификатов X.509 v3, когда они используются как часть интернет-приложения.

Параметры ServerCertificate и ClientCertificate, используемые в абстрактной службе OpenSecureChannel, являются экземплярами типа данных сертификата экземпляра приложения. В 6.2.2 описано, как создать сертификат X.509 v3, который можно применять в качестве сертификата экземпляра приложения.

6.2.2 Сертификат экземпляра приложения

Сертификат экземпляра приложения — строка байтов, содержащая закодированную DER форму (см. X690) сертификата X.509 v3. Данный сертификат выдается сертифицирующим органом и идентифицирует экземпляр приложения, работающего на одном хосте. Поля X.509 v3, содержащиеся в сертификате экземпляра приложения, описаны в таблице 41 (см. также [4]).

В таблице 41 также показано сопоставление терминов (см. [4]) с терминами, используемыми в абстрактном определении сертификата экземпляра приложения в соответствии с ГОСТ Р 71809.

Таблица 41 — Сертификат экземпляра приложения

Имя	Имя параметра (см. ГОСТ Р 71809)	Описание терминов
Application Instance Certificate	—	X.509 v3 Certificate
version	version	Должно быть "V3"

Окончание таблицы 41

Имя	Имя параметра (см. ГОСТ Р 71809)	Описание терминов
serialNumber	serialNumber	Серийный номер, присвоенный эмитентом
signatureAlgorithm	signatureAlgorithm	Алгоритм, используемый для подписания сертификата
signature	signature	Подпись, созданная эмитентом
issuer	issuer	Отличительное имя сертификата, используемого для создания подписи. Поле эмитента полностью описано в [4]
subjectAltName	applicationUri, hostnames	Альтернативные имена экземпляра приложения. Должен включать UniformResourceIdentifier, равный applicationUri. URI должен быть действительным URL-адресом (см. [5]) или действительным URN (см. [6]). Серверы должны указать частичное или полное имя DNS или статический IP-адрес, который идентифицирует компьютер, на котором работает экземпляр приложения. Дополнительные DNSNames можно указать, если у машины несколько имен. Поле subjectAltName полностью описано в [4]
validity	validTo, validFrom	Когда сертификат становится действительным и когда истекает его срок действия
subject	subject	Отличительное имя экземпляра приложения. Атрибут Common Name должен быть указан и должен быть ProductName или подходящим эквивалентом. Атрибут «Имя организации» должен быть наименованием организации, которая выполняет экземпляр приложения. Эта организация обычно не является поставщиком приложения. Могут быть указаны другие атрибуты. Предметная область полностью описана в [4]
keyUsage	keyUsage	Указывает, как можно использовать ключ сертификата. Должен включать digitalSignature, NonRepudiation, keyEncipherment и dataEncipherment. Разрешено другое использование ключа
publicKey	publicKey	Открытый ключ, связанный с сертификатом
extendedKeyUsage	keyUsage	Указывает дополнительные способы использования ключа для сертификата. Необходимо указать serverAuth и/или clientAuth. Разрешено другое использование ключа
authorityKeyIdentifier	Нет сопоставления	Предоставляет дополнительную информацию о ключе, используемом для подписи сертификата. Ключ должен быть указан для сертификатов, подписанных сертифицирующим органом, а также ключ указывается для самозаверяющих сертификатов

6.2.3 Цепочки сертификатов

Любой сертификат X.509 v3 может быть подписан ЦС. Это означает, что для проверки подписи требуется доступ к сертификату X.509 v3, принадлежащему подписавшему ЦС. Каждый раз, когда приложение проверяет подпись, оно должно рекурсивно создавать цепочку сертификатов, находя сертификат издателя, проверяя сертификат и затем повторяя процесс для сертификата издателя. Цепочка заканчивается самозаверяющим сертификатом.

Количество ЦС, используемых в системе, должно быть небольшим, поэтому необходимые ЦС устанавливают на каждую машину с приложением OPC UA. Однако приложения имеют возможность

включать частичную или полную цепочку каждый раз, когда они передают сертификат одноранговому узлу во время согласования SecureChannel и «рукопожатия» CreateSession/ActivateSession. Все приложения OPC UA должны принимать частичные или полные цепочки в любом поле, которое содержит сертификат в кодировке DER.

Цепочки хранятся в ByteString путем добавления сертификатов в кодировке DER. Первый сертификат должен быть конечным сертификатом, за которым следует его выдавший. Если корневой ЦС отправляется как часть цепочки, последний сертификат добавляется к ByteString.

Цепочки анализируются путем извлечения длины каждого сертификата из кодировки DER. Для сертификатов длиной менее 65 535 байт — это UInt16 в кодировке MSB начиная с третьего байта.

6.3 Синхронизация времени

Все протоколы безопасности требуют, чтобы системные часы на взаимодействующих машинах были четко синхронизированы для проверки сроков действия сертификатов или сообщений. Допустимый сдвиг часов зависит от требований безопасности системы, и приложения должны позволять администраторам настраивать приемлемый сдвиг часов при проверке времени. Подходящее значение по умолчанию — 5 мин.

Протокол сетевого времени NTP обеспечивает стандартный способ синхронизации машинных часов с сервером времени в сети. Системы, работающие на машине с полнофункциональной операционной системой, такой как Windows или Linux, уже поддерживают NTP или его эквивалент. Устройства под управлением встроенных операционных систем должны поддерживать NTP.

Если операционная система устройства не поддерживает NTP, то для приложения OPC UA допускается применение временных меток в заголовке ответа в соответствии с ГОСТ Р 71809 для синхронизации своих часов. В этом сценарии приложению OPC UA необходимо знать URL-адрес сервера Discovery на той машине, на которой установлено правильное время. Приложение OPC UA или отдельная фоновая утилита вызовет службу FindServers и установит ее часы на время, указанное в ResponseHeader. Данный процесс необходимо периодически повторять, поскольку со временем часы могут смещаться.

6.4 UTC и международное атомное время

Все время в OPC UA указано в формате UTC, однако UTC может включать разрывы из-за дополнительных секунд или повторяющихся секунд, добавленных для учета изменений орбиты и вращения Земли. Серверы, имеющие доступ к источнику TAI, могут использовать его вместо формата UTC. Тем не менее клиенты должны быть готовы к сбоям в работе из-за формата UTC или потому, что системные часы настроены на сервере.

6.5 Выпущенные токены идентификации пользователя

6.5.1 Kerberos

Kerberos UserIdentityTokens можно передать на сервер с помощью IssuedIdentityToken. Тело токена представляет собой элемент XML, содержащий токен WS-Security, как определено в спецификации профиля токена a Kerberos.

Серверы, поддерживающие аутентификацию Kerberos, должны предоставить UserTokenPolicy, который определяет, какая версия профиля токена Kerberos используется, а также область Kerberos и основное имя Kerberos для сервера. Имя Realm и Principal объединяются с помощью простого синтаксиса и помещаются в IssuerEndpointUri, как показано в таблице 42.

Таблица 42 — Kerberos UserTokenPolicy

Имя	Описание Kerberos
tokenType	ISSUEDTOKEN_3
issuedTypeType	http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1
issuerEndpointUri	Строка вида \\<область>\<имя субъекта сервера>, где: <realm> — имя области Kerberos (например, домен Windows); <имя участника сервера> — имя участника Kerberos для сервера OPC UA

Интерфейс между клиентскими и серверными приложениями и службой аутентификации Kerberos зависит от приложения. Область применения приложений определяется характеристиками контроллера домена Windows в качестве поставщика Kerberos.

6.5.2 Веб-токен JSON (JWT)

Веб-токен JSON (JWT) UserIdentityTokens можно передать на сервер с помощью IssuedIdentityToken. Тело токена представляет собой строку, содержащую JWT, как определено в [2].

Серверы, поддерживающие аутентификацию JWT, должны предоставить UserTokenPolicy, который определяет службу авторизации, предоставляющую токен, и параметры, необходимые для доступа к этой службе. Параметры задаются объектом JSON, указанным как IssuerEndpointUrl. Содержимое этого объекта JSON представлено в таблице 43. Общие настройки UserTokenPolicy для JWT даны в таблице 44.

Таблица 43 — Определение JWT IssuerEndpointUrl

Имя	Тип	Описание полей
IssuerEndpointUrl	JSON object	Указывает параметры для JWT UserIdentityToken
ua:resourceId	String	URI, идентифицирующий сервер для службы авторизации. Если не указано, используется ApplicationUri сервера
ua:authorityUrl	String	Базовый URL-адрес службы авторизации применяется для получения дополнительной информации об органе власти. Данное поле эквивалентно эмитенту, определенному в OpenID-Discovery
ua:authorityProfileUri	String	Профиль, определяющий взаимодействие с органом власти, если не указан адрес URI: http: //opcfoundation.org/UA/Authorization#OAuth2
ua:authorizationEndpoint	String	Путь относительно базового URL-адреса, используемый для проверки учетных данных пользователя. Данный путь эквивалентен полю authorization_endpoint, определенному в OpenID-Discovery
ua:requestTypes	JSON array String	Список типов запросов, поддерживаемых центром. Возможные значения зависят от AuthorityProfileUri. Значение определено по умолчанию для каждого профиля полномочий (см. [1])
ua:scopes	JSON array String	Список областей, которые понимает сервер. Если не указано иное, клиент может иметь доступ к любой области, поддерживаемой службой авторизации. Данный путь эквивалентен полю scopes_supported, определенному в OpenID-Discovery

Таблица 44 — JWT UserTokenPolicy

Имя	Описание полей
tokenType	ISSUEDTOKEN_3
issuedTokenType	http://opcfoundation.org/UA/UserToken#JWT
issuerEndpointUrl	Для JWT — объект JSON с полями, приведенными в таблице 43

6.5.3 OAuth2

6.5.3.1 General

Платформа авторизации OAuth2 (см. [7]) предоставляет веб-механизм для запроса токенов доступа на основе утверждений из службы авторизации (AS), которая поддерживается многими крупными компаниями, предоставляющими облачную инфраструктуру. Эти токены доступа передаются на сервер клиентом в UserIdentityToken в соответствии с ГОСТ Р 71809.

Спецификация OpenID Connect основана на спецификации OAuth2 с более жестким определением содержимого токенов доступа.

Спецификация OAuth2 поддерживает ряд вариантов использования (называемых «потоки») для удовлетворения различных требований приложений. Варианты использования, имеющие отношение к OPC UA, рассмотрены далее.

6.5.3.2 Токены доступа

Веб-токен JSON — это формат токена доступа, который требуется в настоящем стандарте при использовании OAuth2. JWT поддерживает подписи с использованием асимметричной криптографии, что означает, что серверы, принимающие токен доступа, должны иметь доступ к сертификату, используемому службой авторизации (AS). Спецификация OpenID Connect Discovery реализована во многих продуктах AS и предоставляет механизм получения сертификата AS через протокол HTTP.

Если AS не поддерживает спецификацию обнаружения, тогда сертификат подписи необходимо предоставить серверу при добавлении местоположения AS в конфигурацию сервера.

Срок действия токенов доступа истекает, и все серверы должны отозвать любые привилегии, предоставленные сеансу, когда истекает срок действия токена доступа. Если сервер допускает анонимных пользователей, сервер должен разрешить сеансу оставаться открытым, но обращаться с ним как с анонимным пользователем. Если сервер не допускает анонимных пользователей, он должен немедленно закрыть сеанс.

Если известен срок действия токена доступа, требуется запросить новый токен доступа и вызвать ActivateSession до истечения срока действия старого токена доступа. Формат JWT позволяет AS вставлять любое количество полей. Обязательные поля определены в [2]; дополнительные поля, применяемые в заявках на токены доступа, приведены в таблице 45 (см. [8]).

Таблица 45 — Заявки на токены доступа

Поле	Описание полей
sub	Тема токена. client_id, который идентифицирует клиента. Если токен возвращается от поставщика удостоверений, он является уникальным идентификатором пользователя
aud	Слышимость токена. Идентификатор ресурса, который идентифицирует сервер или сервер ApplicationUri
name	Идентифицируемое имя клиентского приложения или пользователя
scp	Список областей, предоставленных субъекту. Области применения применяются к токену доступа и ограничивают его использование
nonce	Одноразовый номер, используемый для смягчения атак повторного воспроизведения. Должно быть значение, предоставленное клиентом в запросе
groups	Список групп, присвоенных субъекту. Список уникальных идентификаторов групп безопасности для конкретной платформы. Например, с помощью данного утверждения могут быть возвращены группы учетных записей пользователей Azure AD
roles	Список ролей, назначенных субъекту. Роли применяются к запрашивающей стороне и описывают, что запрашивающая сторона может делать с ресурсом. Данный список является перечнем уникальных идентификаторов ролей, известных службе авторизации. Значения, указанные в списке, сопоставляются с ролями в соответствии с ГОСТ Р 71808

6.5.3.3 Код авторизации

Поток кода авторизации доступен тем клиентам, которые позволяют взаимодействовать с пользователем-человеком. Клиентское приложение отображает окно с веб-браузером, который отправляет HTTP GET поставщику удостоверений. Когда пользователь-человек вводит учетные данные, поставщик удостоверений проверяет и возвращает код авторизации, который передается в службу авторизации. Служба авторизации проверяет код и возвращает токен доступа клиенту. Полный поток описан в [7] (4.1).

Тип запроса «authorization_code» в UserTokenPolicy (см. 6.5.2) означает, что служба авторизации поддерживает поток кода авторизации.

6.5.3.4 Обновление токена

Поток обновления токена применяется, когда клиентское приложение имеет доступ к токену, возвращенному в предыдущем ответе на запрос кода авторизации. Токен обновления позволяет прило-

жениям пропустить шаг, требующий взаимодействия пользователя-человека с поставщиком удостоверений. Этот поток инициируется, когда клиент отправляет токен обновления в службу авторизации, которая проверяет его и возвращает токен доступа. Клиент, который сохраняет обновленный токен для последующего применения, должен использовать шифрование или другие средства, чтобы гарантировать, что обновленный токен не будет доступен посторонним лицам. Полный поток описан в [7] (раздел 6). Тип запроса заранее не определяется, поскольку поддержка обновления токена формируется на основании проверки ответа на запрос кода авторизации.

6.5.3.5 Учетные данные клиента

Поток учетных данных клиента применяется, когда клиентское приложение не может запросить ввод данных у пользователя. Для данного потока требуется согласованный со службой авторизации код, который может защитить приложение. Данный поток инициируется, когда клиент отправляет client_secret в службу авторизации, которая проверяет его и возвращает токен доступа. Полный поток описан в [7] (4.4). Тип запроса «client_credentials» в UserTokenPolicy (см. 6.5.2) означает, что служба авторизации поддерживает поток учетных данных клиента.

6.6 OPC UA Secure Conversation

6.6.1 Общие положения

UA SC обеспечивает безопасную связь с использованием сообщений в двоичном кодировании.

UA SC предназначен для работы с различными транспортными протоколами, которые могут иметь ограниченные размеры буфера. По этой причине безопасный диалог OPC UA разбивает сообщения OPC UA на несколько частей (называемых «MessageChunks»), которые меньше размера буфера, разрешенного TransportProtocol. UA SC требует, чтобы размер буфера TransportProtocol составлял не менее 8192 байт.

Вся безопасность применяется к отдельным фрагментам сообщения, а не ко всему сообщению OPC UA. Стек, реализующий UA SC, отвечает за проверку безопасности каждого MessageChunk. После этого получено и восстановлено исходное сообщение OPC UA.

Всем MessageChunks будет присвоена 4-байтовая последовательность. Эти порядковые номера используются для обнаружения и предотвращения атак повторного воспроизведения.

UA SC требует TransportProtocol, который сохранит порядок MessageChunks, однако реализация UA SC не обязательно обрабатывает сообщения в том порядке, в котором они были получены.

6.6.2 Структура MessageChunk

6.6.2.1 Общие положения

На рисунке 11 показана структура MessageChunk и то, как к сообщению применяется безопасность.

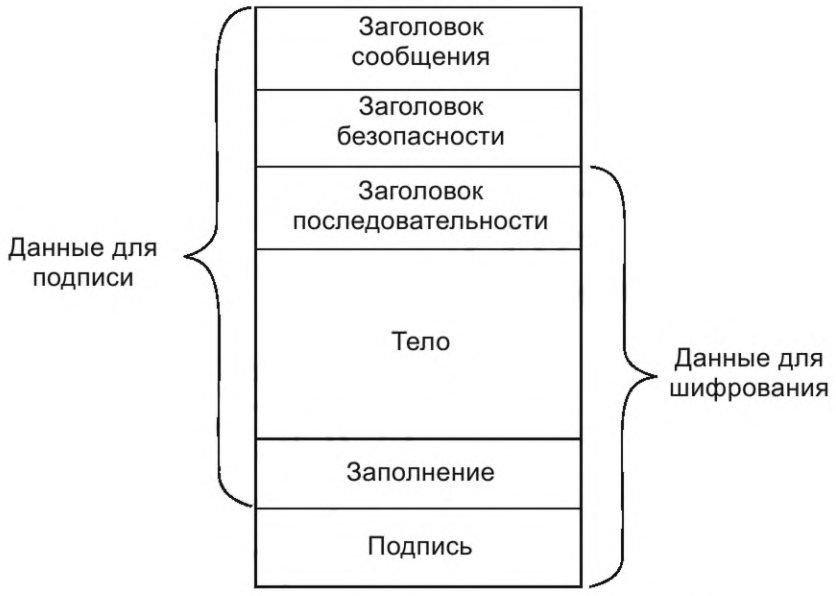

Рисунок 11 — MessageChunk безопасного диалога OPC UA

6.6.2.2 Заголовок сообщения

Каждый MessageChunk имеет заголовок Message с полями, приведенными в таблице 46.

Таблица 46 — Заголовок сообщения безопасного диалога OPC UA

Имя	Тип данных	Описание полей
MessageSize	UInt32	Длина MessageChunk в байтах. Длина начинается с начала поля MessageType
MessageType	Byte	Трехбайтовый код ASCII, определяющий тип сообщения. На данный момент определены нижеприведенные значения: - сообщение MSG, защищенное ключами, связанными с каналом; - сообщение OPN OpenSecureChannel; - сообщение CLO CloseSecureChannel
IsFinal	Byte	Однобайтовый код ASCII, указывающий, является ли MessageChunk последним фрагментом в сообщении. Сообщение. На данный момент определены следующие значения: C — промежуточный фрагмент; F — последний кусок; A — последний фрагмент (используется, когда произошла ошибка и сообщение было прервано). Это поле имеет значение только для типа сообщения «MSG». Для других типов сообщений это поле неизменно равно F
SecureChannelId	UInt32	Уникальный идентификатор SecureChannel, назначенный сервером. Если сервер получает SecureChannelId, который он не распознает, он должен возвратить соответствующую ошибку транспортного уровня. При запуске сервера первый используемый SecureChannelId должен быть значением, которое, вероятно, будет уникальным после каждого перезапуска. Это гарантирует, что перезапуск сервера не приведет к тому, что ранее подключенные клиенты случайно «повторно используют» SecureChannels, которые им не принадлежали

6.6.2.3 Заголовок безопасности

За заголовком сообщения следует заголовок безопасности, который указывает, какие криптографические операции были применены к сообщению. Существует две версии заголовка безопасности, которые зависят от типа безопасности, примененного к сообщению. Заголовок безопасности, предназначенный для асимметричных алгоритмов, приведен в таблице 47. Асимметричные алгоритмы используют для защиты сообщения OpenSecureChannel. PKCS #1 определяет набор асимметричных алгоритмов, которые могут быть применены путем реализации UA SC. Элемент AsymmetricKeyWrapAlgorithm структуры SecurityPolicy, приведенный в таблице 40, не используется реализациями UA SC.

Таблица 47 — Заголовок асимметричного алгоритма обеспечения безопасности

Имя	Тип данных	Описание полей
SecurityPolicyUriLength	Int32	Длина SecurityPolicyUri в байтах. Это значение не должно превышать 255 байт. Если URI не указан, это значение может быть 0 или –1. Другие отрицательные значения недействительны
SecurityPolicyUri	Byte	URI политики безопасности, используемый для защиты сообщения. Данное поле кодируется как строка UTF-8 без нулевого признака-терминатора
SenderCertificateLength	Int32	Длина SenderCertificate в байтах. Данное значение не должно превышать MaxSenderCertificateSize. Если сертификат не указан, то значение может быть 0 или –1. Другие отрицательные значения недействительны

Окончание таблицы 47

Имя	Тип данных	Описание полей
SenderCertificate	Byte	Сертификат X.509 v3, назначенный экземпляру отправляющего приложения. Данный сертификат является большим двоичным объектом, закодированным в DER. Структура сертификата X.509 v3 определена в X.509 v3. Формат DER для сертификата определен в X690. Это указывает, какой закрытый ключ использовался для подписи MessageChunk. Стек должен закрыть канал и сообщить приложению об ошибке, если SenderCertificate превышает размер буфера, поддерживаемого транспортным уровнем, если сообщение не подписано, поле должно быть нулевым. Если сертификат подписан ЦС, сертификат ЦС в кодировке DER может быть добавлен после сертификата в массиве байтов. Если сертификат ЦС также подписан другим ЦС, этот процесс повторяется до тех пор, пока вся цепочка сертификатов не окажется в буфере или пока не будет достигнут предел MaxSenderCertificateSize (процесс останавливается после последнего целого сертификата, который можно добавить, не превышая предел MaxSenderCertificateSize). Размер сертификата содержится в заголовке DER (см. X.509 v3). Получатели, которые не обрабатывают цепочки сертификатов, должны игнорировать дополнительные байты
ReceiverCertificate ThumbprintLe ngth	Int32	Длина ReceiverCertificateThumbprint в байтах. В зашифрованном виде длина этого поля составляет 20 байт. Если не зашифровано, значение может быть 0 или –1. Другие отрицательные значения недействительны
ReceiverCertificate Thumbprint	Byte	Отпечаток сертификата X.509 v3, назначенного принимающему экземпляру приложения. Отпечаток — сертифицированный дайджест формы сертификата, закодированной в DER. Данная кодировка указывает, какой открытый ключ применен для шифрования MessageChunk. Если сообщение не зашифровано, поле должно быть нулевым

Параметры асимметричного алгоритма обеспечения безопасности включают в себя:

MaxSenderCertificateSize = MessageChunkSize —

12 — // размер заголовка.

4 — // SecurityPolicyUriLength.

SecurityPolicyUri — // строка в кодировке UTF-8.

4 — // длина сертификата отправителя.

4 — // длина сертификата отпечатка получателя.

20 — // отпечаток сертификата получателя.

8 — // размер заголовка последовательности.

1 — // минимальный размер тела.

1 — // PaddingSize, если присутствует.

Padding — // заполнение, если оно присутствует.

ExtraPadding — // ExtraPadding, если присутствует.

AsymmetricSignatureSize // если присутствует.

MessageChunkSize зависит от транспортного протокола, но должен составлять не менее 8192 байт. AsymmetricSignatureSize зависит от количества битов в открытом ключе SenderCertificate. Int32FieldLength — длина закодированного значения Int32, которая всегда равна 4 байтам.

Заголовок обеспечения безопасности используется для симметричных алгоритмов, приведенных в таблице 48. Симметричные алгоритмы применяются для защиты всех сообщений, кроме сообщений OpenSecureChannel.

Таблица 48 — Заголовок безопасности симметричного алгоритма

Имя	Тип данных	Описание полей
TokenId	UInt32	Уникальный идентификатор SecureChannel SecurityToken, используемый для защиты сообщения. Данный идентификатор возвращается сервером в ответном сообщении OpenSecureChannel. Если сервер получает TokenId, который он не распознает, он должен возвратить соответствующую ошибку транспортного уровня

6.6.2.4 Sequence header

За заголовком безопасности неизменно следует заголовок последовательности, приведенный в таблице 49. Заголовок последовательности гарантирует, что первый зашифрованный блок каждого сообщения, отправляемого по каналу, будет начинаться с разных данных.

Таблица 49 — Заголовок последовательности

Имя	Тип данных	Описание полей
SequenceNumber	UInt32	Монотонно увеличивающийся порядковый номер, присваиваемый отправителем каждому. MessageChunk отправляется через SecureChannel
RequestId	UInt32	Идентификатор, назначенный клиентом сообщению запроса OPC UA. Все MessageChunk для запроса и связанного ответа используют один и тот же идентификатор

SequenceNumber не может быть повторно использован для любого TokenId. Для исключения возможности повторного использования действие SecurityToken должно быть достаточно коротким. Если повторное использование произойдет, получатель должен рассмотреть его как ошибку транспорта и принудительно выполнить повторное соединение.

SequenceNumber должен начинаться с 1023 и монотонно увеличиваться для всех сообщений и должен повторяться, когда он равен 4 294 966 271 (UInt32.MaxValue — 1024). Первое число после переноса должно быть менее 1024. Данное требование означает, что SequenceNumber не сбрасывается при выдаче нового Token Id. Порядковый номер должен увеличиваться ровно на единицу для каждого отправленного MessageChunk. В целях обратной совместимости получатели должны принимать порядковые номера менее 1023 и более 4 294 966 271 при условии, что они расположены последовательно. Администраторы должны иметь возможность отключить эту обратную совместимость. Получатели должны регистрировать предупреждение, когда ролловер не соответствует текущей спецификации.

За заголовком последовательности следует тело сообщения, которое закодировано с помощью двоичной кодировки OPC UA, как описано в 5.2.9. Тело может быть разделено на несколько MessageChunks.

6.6.2.5 Message footer

Каждый MessageChunk также имеет нижний колонтитул с полями, приведенными в таблице 50.

Таблица 50 — Нижний колонтитул сообщения безопасного диалога OPC UA

Имя	Тип данных	Описание полей
PaddingSize	Byte	Количество байтов заполнения (не включая байт для PaddingSize)
Padding	Byte	Заполнение добавляется в конец сообщения, чтобы гарантировать, что длина шифруемых данных является целым числом, кратным размеру блока шифрования. Значение каждого байта заполнения равно PaddingSize
ExtraPaddingSize	Byte	Старший байт 2-байтового целого числа, применяемый для указания размера заполнения, должен содержать ключ для шифрования фрагмента сообщения, превышающий 2048 бит. Данное поле опускается, если длина ключа менее или равна 2048 бит

Окончание таблицы 50

Имя	Тип данных	Описание полей
Signature	Byte	Подпись для MessageChunk. Подпись включает в себя все заголовки, все данные сообщения, PaddingSize и Padding

Формула для расчета объема заполнения зависит от объема данных, которые необходимо отправить (так называемая BytesToWrite). Отправитель должен сначала рассчитать максимальный объем доступного места в MessageChunk (называемый MaxBodySize), используя следующую формулу:

$$MaxBodySize = PlainTextBlockSize * Floor ((MessageChunkSize — HeaderSize — 1)/CipherTextBlockSize) — SequenceHeaderSize — SignatureSize. \quad (1)$$

HeaderSize включает MessageHeader и SecurityHeader. SequenceHeaderSize всегда имеет размер 8 байт. Во время шифрования блок размером, равным PlainTextBlockSize, обрабатывается для создания блока размером, равным CipherTextBlockSize. Данные значения зависят от алгоритма шифрования и могут совпадать.

Сообщение OPC UA может поместиться в один фрагмент, если значение BytesToWrite меньше или равно MaxBodySize. В этом случае PaddingSize рассчитывают по формуле

$$PaddingSize = PlainTextBlockSize — ((BytesToWrite + SignatureSize + 1) \% PlainTextBlockSize). \quad (2)$$

Если BytesToWrite больше MaxBodySize, отправитель должен записать байты MaxBodySize с PaddingSize, равным 0. Остальные байты BytesToWrite — MaxBodySize, должны быть отправлены в последующих MessageChunks.

Поля PaddingSize и Padding отсутствуют, если MessageChunk не зашифрован.

Поле «Подпись» отсутствует, если MessageChunk не подписан.

6.6.3 MessageChunks и обработка ошибок

MessageChunks отправляются в том виде, в котором они закодированы. Части сообщения, принадлежащие одному и тому же сообщению, должны отправляться последовательно. Если при создании MessageChunk возникает ошибка, отправитель должен отправить окончательный MessageChunk получателю, который сообщает получателю, что произошла ошибка и что он должен отбросить предыдущие фрагменты. Отправитель указывает, что MessageChunk содержит ошибку, устанавливая флаг IsFinal в значение «A» (для прерывания). В таблице 51 определено содержимое Message Abort MessageChunk.

Таблица 51 — Тело прерывания сообщения безопасного диалога OPC UA

Имя	Тип данных	Описание Message Abort MessageChunk
Error	UInt32	Числовой код ошибки. Это должно быть одно из значений, перечисленных в 7.1.5
Reason	String	Более подробное описание ошибки. Клиент должен игнорировать строки, длина которых превышает 4096 байт

Получатель должен проверить безопасность прерванного MessageChunk перед его обработкой. Если все в порядке, то получателю следует проигнорировать сообщение, но не закрывать SecureChannel. Клиент должен сообщить об ошибке приложению как StatusCode для запроса. Если отправителем является клиент, то он должен сообщить об ошибке, не дожидаясь ответа от сервера.

6.6.4 Установление SecureChannel

Для большинства сообщений требуется установка SecureChannel. Клиент делает это, отправляя запрос OpenSecureChannel на сервер. Сервер должен проверить сообщение и ClientCertificate и возвратить ответ OpenSecureChannel. Некоторые параметры, определенные для службы OpenSecureChannel, указаны в заголовке безопасности (см. 6.7.2), а не в теле сообщения. В таблице 52 перечислены параметры, которые появляются в теле сообщения.

Таблица 52 — Служба безопасного диалога OPC UA OpenSecureChannel

Имя	Тип данных
Запрос	
RequestHeader	RequestHeader
ClientProtocolVersion	UInt32
RequestType	SecurityTokenRequestType
SecurityMode	MessageSecurityMode
ClientNonce	ByteString
RequestedLifetime	UInt32
Ответ	
ResponseHeader	ResponseHeader
ServerProtocolVersion	UInt32
SecurityToken	ChannelSecurityToken
SecureChannelId	UInt32
TokenId	UInt32
CreatedAt	UtcTime
RevisedLifetime	UInt32
ServerNonce	ByteString

Параметры ClientProtocolVersion и ServerProtocolVersion добавляются в сообщение для обеспечения обратной совместимости, если UA SC потребуется обновить в дальнейшем. Получатели неизменно принимают числа, превышающие последнюю версию, которую они поддерживают. Предполагается, что приемник с более высоким номером версии обеспечит обратную совместимость.

Если UA SC используют с протоколом TCP OPC UA (см. 7.1), то номера версий, указанные в сообщениях OpenSecureChannel, должны быть такими же, как номера версий, указанные в сообщениях приветствия/подтверждения протокола TCP OPC UA. Получатель должен закрыть канал и сообщить об ошибке Bad_ProtocolVersionUnsupported при наличии несоответствия.

Сервер должен возвратить ответ об ошибке в соответствии с ГОСТ Р 71809, если допущены какие-либо ошибки с параметрами, указанными клиентом.

RevisedLifetime сообщает клиенту, когда он должен обновить SecurityToken, отправив еще один запрос OpenSecureChannel. Клиент должен продолжать раннюю версию SecurityToken до тех пор, пока не получит ответ OpenSecureChannel. Сервер должен принимать запросы, защищенные необновленным SecurityToken, до истечения срока действия SecurityToken или до тех пор, пока сервер не получит сообщение от клиента, защищенное новым SecurityToken. Сервер должен отклонять запросы на обновление, если SenderCertificate не совпадает с тем, который применялся для создания SecureChannel, или если существует проблема с расшифровкой или проверкой подписи. Клиент должен отказаться от SecureChannel, если сертификат, использованный для подписи ответа, не совпадает с сертификатом для шифрования запроса. Следует иметь в виду, что тип данных — это значение UInt32, представляющее количество миллисекунд вместо Double (длительность), определенного согласно ГОСТ Р 71809.

Сообщения OpenSecureChannel подписываются и шифруются, если SecurityMode не имеет значения None (в том числе, если SecurityMode имеет значение Sign).

Nonces должны быть криптографическими случайными числами, длина которых указана параметром SecureChannelNonceLength в SecurityPolicy.

Дополнительная информация о требованиях к генераторам случайных чисел приведена в [9]. Сообщения OpenSecureChannel не подписываются и не шифруются. Если Nonce имеет значение NULL, то SecurityMode требуется игнорировать. SecureChannelId и TokenId назначаются к сообщениям, которыми обмениваются через канал, безопасность не применяется. SecurityToken должен быть продлен

до истечения срока действия RevisedLifetime. Получатели должны игнорировать недействительные или просроченные идентификаторы TokenId.

Если канал связи выходит из строя, сервер должен поддерживать SecureChannel достаточно долго, чтобы позволить клиенту повторно подключиться. Параметр RevisedLifetime также сообщает клиенту, как долго сервер будет находиться в режиме ожидания. Если клиент не может повторно подключиться в течение этого периода, он должен считать, что SecureChannel закрыт. AuthenticationToken в RequestHeader должен иметь значение NULL.

Если ошибка возникает после того, как сервер проверил безопасность сообщения, он должен возвратить ServiceFault вместо ответа OpenSecureChannel. Сообщение ServiceFault описано в ГОСТ Р 71809.

Если значение SecurityMode не равно значению None, то сервер должен проверить, что SenderCertificate и ReceiverCertificateThumbprint указаны в SecurityHeader.

6.6.5 Получение ключей

После установки SecureChannel сообщения подписываются и шифруются с помощью ключей, полученных из одноразовых номеров, которыми обмениваются при вызове OpenSecureChannel. Эти ключи получаются путем передачи одноразовых номеров PRF, которая создает последовательность байтов из набора входных данных. PRF представлена следующим объявлением функции:

Byte[] PRF(
Byte[] secret,
Byte[] seed,
Int32 length,
Int32 offset)

где длина — это количество возвращаемых байтов, а смещение — это количество байтов от начала последовательности. Длина ключей, которые необходимо сгенерировать, зависит от политики безопасности, используемой для канала. Следующая информация определяется SecurityPolicy:

а) SigningKeyLength (из DerivedSignatureKeyLength);

б) EncryptingKeyLength (предполагаемый алгоритмом SymmetricEncryptionAlgorithm);

в) EncryptingBlockSize (предполагаемый алгоритмом SymmetricEncryptionAlgorithm).

PRF требует наличия секретного ключа и начального числа. Эти значения извлекаются из одноразовых номеров, которыми обмениваются в запросе и ответе OpenSecureChannel. В таблице 53 указано, как получить секреты и начальные значения при использовании политик безопасности на основе RSA.

Таблица 53 — Входные данные PRF для политик безопасности на основе RSA

Имя	Вывод
ClientSecret	Значение ClientNonce, указанное в запросе OpenSecureChannel
ClientSeed	Значение ClientNonce, указанное в запросе OpenSecureChannel
ServerSecret	Значение ServerNonce, указанное в ответе OpenSecureChannel
ServerSeed	Значение ServerNonce, указанное в ответе OpenSecureChannel

Параметры, передаваемые в PRF, представлены в таблице 54.

Таблица 54 — Параметры генерации криптографических ключей

Ключ	Безопасность	Seed	Длина	Смещение
ClientSigningKey	ServerSecret	ClientSeed	SigningKeyLength	0
ClientEncryptingKey	ServerSecret	ClientSeed	EncryptingKeyLength	SigningKeyLength
ClientInitialization Vector	ServerSecret	ClientSeed	EncryptingBlockSize	SigningKeyLength+ EncryptingKeyLength
ServerSigningKey	ClientSecret	ServerSeed	SigningKeyLength	0
ServerEncryptingKey	ClientSecret	ServerSeed	EncryptingKeyLength	SigningKeyLength
ServerInitialization Vector	ClientSecret	ServerSeed	EncryptingBlockSize	SigningKeyLength+ EncryptingKeyLength

Ключи клиента используют для защиты сообщений, отправляемых клиентом; ключи сервера — для защиты сообщений, отправляемых сервером. Спецификация SSL/TLS определяет PRF P_HASH, которая применяется для этой цели. Функция повторяется до тех пор, пока не выдаст достаточно данных для всех необходимых ключей. Смещение в таблице 54 относится к смещению от начала сгенерированных данных.

Алгоритм хеширования P_ определен следующим образом:

P_HASH(secret, seed) = HMAC_HASH(secret, A(1) + seed) +

+ HMAC_HASH(secret, A(2) + seed) + HMAC_HASH(secret, A(3) + seed) + ...

Where A(n) is defined as: A(0) = seed

A(n) = HMAC_HASH(secret, A(n–1))

+ указывает, что результаты добавляются к предыдущим результатам,

где HASH — хеш-функция, например SHA256. Используемая хеш-функция зависит от SecurityPolicyUri.

6.6.6 Проверка безопасности сообщения

Содержимое MessageChunk не должно интерпретироваться до тех пор, пока сообщение не будет расшифровано и не будут проверены подпись и порядковый номер.

Если при проверке сообщения возникает ошибка, получатель закрывает канал связи. Если получателем является сервер, он также должен отправить сообщение об ошибке транспорта перед закрытием канала. После закрытия канала клиент попытается повторно открыть канал и запросить новый SecurityToken, отправив запрос OpenSecureChannel. Механизм отправки ошибок транспорта клиенту зависит от канала связи.

Получатель сначала должен проверить SecureChannelId. Это значение может быть равно 0, если сообщение является запросом OpenSecureChannel. Для других сообщений он должен сообщить об ошибке Bad_SecureChannelUnknown, если SecureChannelId не распознан. Если сообщение — это запрос OpenSecureChannel и SecureChannelId не равен 0, тогда SenderCertificate должен быть таким же, как SenderCertificate, использованный для создания канала.

Если сообщение защищено асимметричными алгоритмами, получатель должен убедиться в том, что оно поддерживает запрошенную политику безопасности. Если сообщение является ответом, отправленным клиенту, то SecurityPolicy должна быть таким же, как тот, который указан в запросе. На сервере SecurityPolicy должна быть такой же, как и та, которая использовалась при первоначальном создании SecureChannel. Получатель должен сначала проверить, что сертификату доверяют, и в случае ошибки возвратить Bad_CertificateUntrusted. Затем получатель должен проверить SenderCertificate, используя правила, определенные согласно ГОСТ Р 71809. Получатель должен сообщить о соответствующей ошибке, если проверка сертификата не удалась. Получатель должен проверить ReceiverCertificateThumbprint и сообщить Bad_CertificateInvalid, если он ее не распознает.

Если сообщение защищено симметричными алгоритмами, то Bad_SecureChannel об ошибке TokenUnknown должно быть сообщено, если TokenId относится к SecurityToken, срок действия которого истек или не распознан. Если расшифровка или проверка подписи завершается неудачей, сообщается об ошибке Bad_SecurityChecksFailed.

Если реализация допускает использование нескольких режимов безопасности, получатель также должен проверить, что сообщение было защищено должным образом в соответствии с требованиями режима безопасности, указанного в запросе OpenSecureChannel.

После завершения проверки безопасности получатель должен проверить RequestId и SequenceNumber. Если проверки завершаются неудачно, сообщается об ошибке Bad_SecurityChecksFailed. RequestId должен быть проверен только клиентом, так как только клиент знает, действителен ли он. Если SequenceNumber недействителен, получатель должен зарегистрировать Bad_SequenceNumberInvalid и сообщить об ошибке.

На этом этапе SecureChannel должен подтвердить, что проверенное сообщение не подделано или отправлено повторно. Это означает, что SecureChannel может возвращать защищенные ответы об ошибках, если возникнут какие-либо проблемы.

Стеки, реализующие UA SC, должны иметь механизм регистрации ошибок, при получении недействительных сообщений они должны игнорироваться.

7 Транспортные протоколы

7.1 Протокол подключения OPC UA

7.1.1 Общие положения

Протокол соединения OPC UA (UA CP) — абстрактный протокол, который устанавливает полнодуплексный канал между клиентом и сервером. Конкретные реализации UA CP могут быть построены с использованием любого промежуточного программного обеспечения, поддерживающего полнодуплексный обмен сообщениями, включая TCP/IP и веб-сокеты. Термин «TransportConnection» описывает соединение, специфичное для промежуточного программного обеспечения и используемое для обмена сообщениями. Например, сокет — это TransportConnection для TCP/IP. TransportConnection позволяет возвращать ответы в любом порядке, а также ответы на другое физическое соединение TransportConnection, если сбои связи вызывают временные перерывы.

Протокол соединения OPC UA предназначен для работы с SecureChannel, реализованным на более высоком уровне стека. По этой причине протокол соединения OPC UA определяет взаимодействие с SecureChannel в дополнение к проводному протоколу.

7.1.2 Структура сообщения

7.1.2.1 Общие положения

На рисунке 12 проиллюстрирована структура сообщения, передаваемого по проводу, а также то, как элементы сообщения, определенные отображением двоичного кодирования OPC UA (см. 5.2) и отображением безопасного разговора OPC UA (см. 6.7), связаны с сообщениями протокола соединения OPC UA.

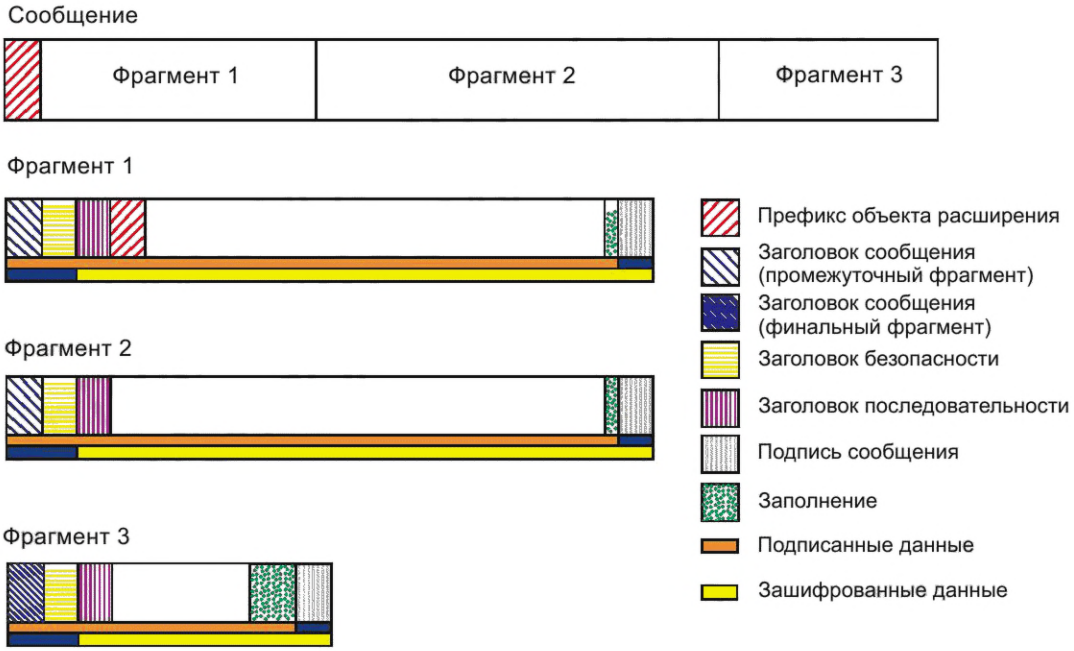

Рисунок 12 — Структура сообщения протокола соединения OPC UA

7.1.2.2 Заголовок сообщения

Каждое сообщение протокола соединения OPC UA имеет заголовок с полями, определенными в таблице 55.

Таблица 55 — Заголовок сообщения протокола соединения OPC UA

Имя	Тип	Описание полей
MessageType	Byte	Трехбайтовый код ASCII, определяющий тип сообщения. На данный момент определены следующие значения: HEL — приветственное сообщение; ACK — сообщение подтверждения; ERR — сообщение об ошибке; RHE — сообщение ReverseHello. Уровень SecureChannel определяет дополнительные значения, которые должен принимать уровень протокола соединения OPC UA
Reserved	Byte	Игнорируется. Должен быть установлен в коды ASCII для «F», если MessageType является одним из значений, поддерживаемых протоколом соединения OPC UA
MessageSize	UInt32	Длина сообщения в байтах, включающая 8 байтов заголовка сообщения

Структура заголовка сообщения протокола соединения OPC UA намеренно идентична первым 8 байтам заголовка сообщения безопасного диалога OPC UA (см. таблицу 46). Это позволяет уровню протокола соединения OPC UA извлекать сообщения Secure Channel из входящего потока, даже если он не понимает их содержимое.

На уровне протокола соединения OPC UA необходимо проверить MessageType и убедиться в том, что MessageSize меньше согласованного ReceiveBufferSize, прежде чем передавать любое сообщение на уровень SecureChannel.

7.1.2.3 Hello Message

Hello Message имеет дополнительные поля, представленные в таблице 56.

Таблица 56 — Приветственное сообщение протокола соединения OPC UA

Имя	Тип данных	Описание полей
ProtocolVersion	UInt32	Последняя версия протокола UA CP, поддерживаемая клиентом. Сервер может отклонить клиента, возвратив Bad_ProtocolVersionUnsupported. Если сервер принимает соединение, он несет ответственность за возврат сообщения, соответствующего этой версии протокола. Сервер неизменно должен принимать версии, превышающие те, которые он поддерживает
ReceiveBufferSize	UInt32	Самый большой MessageChunk, который может получить отправитель
SendBufferSize	UInt32	Самый большой MessageChunk, который отправит отправитель
MaxMessageSize	UInt32	Максимальный размер любого ответного сообщения. Сервер должен прервать передачу сообщения с сообщением об ошибке Bad_ResponseTooLarge, если ответное сообщение превышает это значение. Механизм прерывания сообщений в полном объеме описан в 6.7.3. Размер сообщения рассчитывают с использованием незашифрованного тела сообщения. NULL указывает на то, что у клиента отсутствует лимит
MaxChunkCount	UInt32	Максимальное количество фрагментов в любом ответном сообщении. Сервер должен прервать передачу сообщения с сообщением об ошибке Bad_ResponseTooLarge, если ответное сообщение превышает это значение. Описание механизма прерывания сообщений в полном объеме приведено в 6.7.3. NULL указывает на то, что у клиента отсутствует лимит

Окончание таблицы 56

Имя	Тип данных	Описание полей
EndpointUrl	String	URL-адрес конечной точки, к которой хотел подключиться клиент. Закодированное значение должно быть менее 4096 байтов. Серверы должны возвратить сообщение об ошибке Bad_TcpEndpointUrlInvalid и закрыть соединение, если длина превышает 4096 байтов или если он не распознает ресурс, идентифицированный URL-адресом

Параметр EndpointUrl позволяет нескольким серверам использовать одну и ту же конечную точку на компьютере. На конечной точке прокси-процесс должен подключиться к серверу, указанному в EndpointUrl, и пересылать все сообщения на сервер через этот сокет. Если один сокет закрывается, прокси-сервер должен закрыть другой сокет.

Если у сервера недостаточно ресурсов для создания нового SecureChannel, он должен немедленно возвратить сообщение об ошибке Bad_TcpNotEnoughResources и закрыть сокет.

7.1.2.4 Сообщение подтверждения

Сообщение подтверждения имеет дополнительные поля, показанные в таблице 57.

Таблица 57 — Сообщение подтверждения протокола соединения OPC UA

Имя	Тип	Описание полей
ProtocolVersion	UInt32	Последняя версия протокола UA CP, поддерживаемая сервером. Если клиент принимает соединение, он несет ответственность за отправку сообщений, которые соответствуют этой версии протокола. Клиент непременно должен принимать версии, превышающие те, которые он поддерживает
ReceiveBufferSize	UInt32	Самый большой MessageChunk, который может получить отправитель. Данное значение не должно превышать значения, запрошенного клиентом в приветственном сообщении
SendBufferSize	UInt32	Самый большой MessageChunk, который отправит отправитель. Данное значение не должно превышать значения, запрошенного клиентом в приветственном сообщении
MaxMessageSize	UInt32	Максимальный размер любого сообщения запроса. Клиент должен прервать сообщение с помощью Bad_RequestTooLarge StatusCode, если сообщение запроса превышает это значение. Механизм прерывания сообщений в полном объеме описан в 6.7.3. Размер сообщения рассчитывают с использованием незашифрованного тела сообщения. NULL указывает, что у сервера отсутствуют ограничения
MaxChunkCount	UInt32	Максимальное количество фрагментов в любом сообщении запроса. Клиент должен прервать передачу сообщения с кодом состояния Bad_RequestTooLarge, если запрос сообщения превышает это значение. Механизм прерывания сообщений в полном объеме описан в 6.7.3. NULL указывает, что у сервера отсутствуют ограничения

7.1.2.5 Сообщение об ошибке

Сообщение об ошибке имеет дополнительные поля, представленные в таблице 58.

Таблица 58 — Сообщение об ошибке протокола соединения OPC UA

Имя	Тип	Описание полей
Error	UInt32	Числовой код ошибки. Данное значение должно быть одним из перечисленных в 7.1.5
Reason	String	Подробное описание ошибки. Клиент должен игнорировать строки, длина которых превышает 4096 байт

Сокет неизменно корректно закрывается клиентом после получения сообщения об ошибке.

7.1.2.6 Сообщение ReverseHello

Сообщение ReverseHello имеет дополнительные поля, представленные в таблице 59.

Таблица 59 — Сообщение ReverseHello протокола соединения OPC UA

Имя	Тип данных	Описание полей
ServerUri	String	ApplicationUri сервера, отправившего сообщение. Закодированное значение должно быть менее 4096 байт. Клиент должен возвратить ошибку Bad_TcpEndpointUrlInvalid и закрыть соединение, если длина превышает 4096 байт или если он не распознает сервер, идентифицированный URI
EndpointUrl	String	URL-адрес конечной точки, которую клиент использует при установлении SecureChannel. Данное значение должно быть передано обратно на сервер в сообщении приветствия. Закодированное значение должно быть менее 4096 байт. Клиенты должны возвратить ошибку Bad_TcpEndpointUrlInvalid и закрыть соединение, если длина превышает 4096 байт или если не распознается ресурс, идентифицированный URL-адресом. Данное значение является уникальным идентификатором сервера, который клиент может использовать для поиска информации о конфигурации. Это должен быть один из URL-адресов, возвращаемых службой GetEndpoints

Для протоколов на основе сообщений сообщение ReverseHello позволяет серверам сообщать клиенту о своем присутствии. В данном сообщении EndpointUrl указывает адрес сервера, зависящий от протокола, и все токены, необходимые для доступа к нему.

7.1.3 Установление соединения

Соединения могут быть инициированы клиентом или сервером, когда они создают TransportConnection и устанавливают связь со своим одноранговым узлом. Если клиент создает TransportConnection, первое отправленное сообщение должно быть приветствием, в котором указаны размеры буфера, которые поддерживает клиент. Сервер должен ответить сообщением подтверждения, которое завершает согласование буфера. Согласованный размер буфера должен быть сообщен на уровень SecureChannel. Согласованный параметр SendBufferSize определяет размер MessageChunks, который будет использоваться для сообщений, отправляемых через соединение.

Если сервер создает TransportConnection, первым сообщением будет ReverseHello, отправленное клиенту. Если клиент принимает соединение, он отправляет сообщение Hello обратно на сервер, который запускает согласование буфера, описанное для соединения, инициированного клиентом.

Сообщения приветствия/подтверждения могут быть отправлены только один раз. Если они получены повторно, получатель должен сообщить об ошибке и закрыть TransportConnection. Приложения, принимающие входящие соединения, должны закрыть TransportConnection по истечении периода времени, если оно не получит сообщение Hello или ReverseHello. Этот период времени должен быть настраиваемым и иметь значение по умолчанию, не превышающее 2 мин.

Клиент отправляет запрос OpenSecureChannel после получения подтверждения от сервера. Если сервер принимает новый канал, он должен связать TransportConnection с SecureChannelId. Сервер использует принимаемую ассоциацию, чтобы определить, какой TransportConnection использовать, когда ему необходимо отправить ответ клиенту. Клиент делает то же самое, когда получает ответ OpenSecureChannel.

Последовательность сообщений при установлении протокола соединения OPC UA, соединение по которому инициировал клиент, показано на рисунке 13.

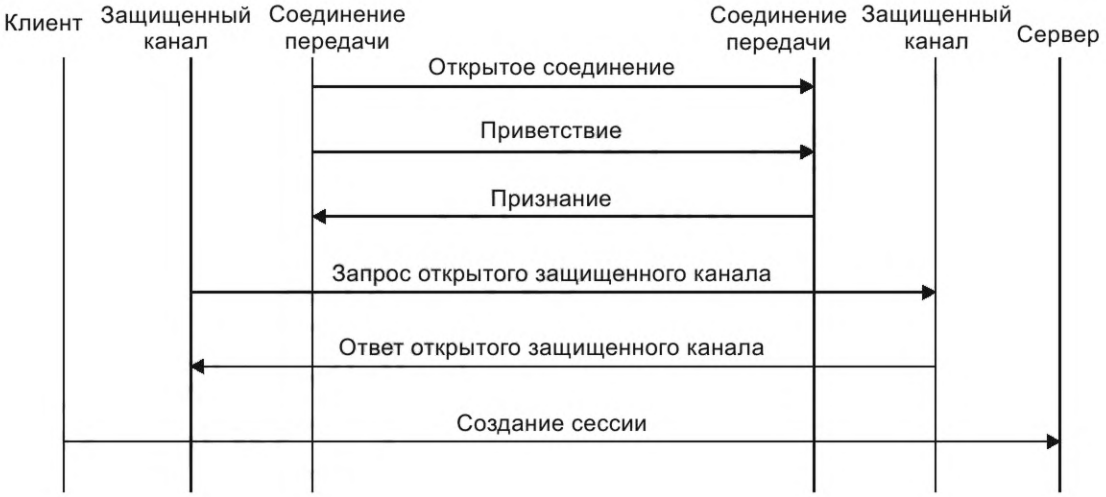

Рисунок 13 — Клиент инициировал соединение по протоколу соединения OPC UA

Последовательность сообщений при установлении протокола соединения OPC UA, соединение по которому инициировал сервер, показана на рисунке 14.

Рисунок 14 — Инициированное сервером соединение по протоколу соединения OPC UA

Приложение сервера не выполняет обработки во время согласования SecureChannel; однако серверное приложение должно предоставить стеку список доверенных сертификатов. Стек должен направлять уведомления серверному приложению каждый раз, когда оно получает запрос OpenSecureChannel. Уведомления должны включать ответ OpenSecureChannel или ошибку, возвращаемые клиенту.

Сервер должен быть настроен и включен администратором для подключения к одному или нескольким клиентам. Для каждого клиента администратор должен предоставить ApplicationUri и EndpointUrl, если EndpointUrl клиента не указан, администратор может предоставить EndpointUrl для GDS (см. [10]), которая знает о клиенте. Сервер должен находиться в режиме ожидания, так как клиенту потребуется некоторое время, чтобы ответить на ReverseHello. Клиент закрывает SecureChannel, или, если сокет закрывается без установления SecureChannel, сервер должен создать новый сокет и отправить новое сообщение ReverseHello. Администраторы могут ограничить количество одновременных сокетов, создаваемых сервером.

7.1.4 Закрытие соединения

Клиент закрывает соединение, отправляя запрос CloseSecureChannel и корректно закрывая сокет. Когда сервер получает сообщение, он должен освободить все ресурсы, выделенные для канала. Тело запроса CloseSecureChannel пустое. Сервер не отправляет ответ CloseSecureChannel.

Если проверка безопасности для сообщения CloseSecureChannel не удалась, сервер должен сообщить об ошибке и закрыть сокет. Последовательность сообщений при закрытии соединения по протоколу соединения OPC UA показана на рисунке 15.

Рисунок 15 — Закрытие соединения по протоколу соединения OPC UA

Серверное приложение не выполняет обработки, когда SecureChannel закрыт; однако стек должен предоставлять уведомления серверному приложению каждый раз, когда получен запрос CloseSecureChannel или когда стек очищает заброшенный SecureChannel.

7.1.5 Обработка ошибок

При возникновении ошибки, приводящей к необратимым последствиям, сервер отправляет клиенту сообщение об ошибке и корректно закрывает TransportConnection. Когда клиент получает сообщение об ошибке, он сообщает об ошибке приложению и корректно закрывает TransportConnection. Если клиент сталкивается с ошибкой, приводящей к необратимым последствиям, он должен сообщить об ошибке приложению и отправить CloseSecureChannel сообщение. Сервер должен корректно закрыть TransportConnection при получении сообщения CloseSecureChannel.

Возможные ошибки протокола соединения OPC UA определены в таблице 60.

Таблица 60 — Коды ошибок протокола соединения OPC UA

Имя	Описание полей
Bad_TcpServerTooBusy	Сервер не может обработать запрос, поскольку он чересчур занят. Сервер должен определить, когда ему необходимо возвратить сообщение. Сервер может контролировать частоту повторного подключения клиента, ожидая возврата этой ошибки
Bad_TcpMessageTypeInvalid	Недопустимый тип сообщения, указанный в заголовке. Каждое сообщение начинается с 4-байтовой последовательности значений ASCII, которая идентифицирует тип сообщения. Сервер возвращает ошибку, если тип сообщения не принят. Ряд типов сообщений определен уровнем SecureChannel
Bad_TcpSecureChannelUnknown	SecureChannelId и/или TokenId в настоящее время не используются. Об этой ошибке сообщает уровень SecureChannel
Bad_TcpMessageTooLarge	Размер сообщения, указанного в заголовке, чересчур велик. Сервер возвращает эту ошибку, если размер сообщения превышает максимальный размер буфера или размер буфера приема, согласованный во время обмена приветствием/подтверждением

Окончание таблицы 60

Имя	Описание полей
Bad_Timeout	При доступе к ресурсу произошел тайм-аут. Сервер должен определить, когда наступит тайм-аут
Bad_TcpNotEnoughResources	Недостаточно ресурсов для обработки запроса. Сервер возвращает эту ошибку, когда ему не хватает памяти или возникают аналогичные проблемы с ресурсами. Сервер может контролировать частоту повторного подключения клиента, ожидая возврата ошибки
Bad_TcpInternalError	Возникла внутренняя ошибка. Его следует возвращать только в случае возникновения непредвиденной ошибки конфигурации или программирования
Bad_TcpEndpointUrlInvalid	Сервер не распознает указанный EndpointUrl
Bad_RequestInterrupted	Запрос не удалось отправить из-за сбоя в сети
Bad_RequestTimeout	При обработке запроса произошел тайм-аут
Bad_SecureChannelClosed	Защищенный канал закрыт
Bad_SecureChannelTokenUnknown	Срок действия SecurityToken истек или он не распознан
Bad_CertificateUntrusted	Сертификат отправителя не пользуется доверием получателя
Bad_CertificateTimeInvalid	Срок действия сертификата отправителя истек или не действителен
Bad_CertificateIssuerTimeInvalid	Срок действия сертификата отправителя истек или не действителен
Bad_CertificateUseNotAllowed	Сертификат отправителя не может быть использован для установления защищенного канала
Bad_CertificateIssuerUseNotAllowed	Сертификат эмитента не может использоваться в качестве ЦС
Bad_CertificateRevocationUnknown	Не удалось проверить статус отзыва сертификата отправителя
Bad_CertificateIssuerRevocationUnknown	Не удалось проверить статус отзыва сертификата эмитента
Bad_CertificateRevoked	Сертификат отправителя отозван эмитентом
Bad_IssuerCertificateRevoked	Сертификат эмитента отозван его эмитентом
Bad_SequenceNumberInvalid	Порядковый номер в сообщении недействителен

Числовые значения этих кодов ошибок определены в А.2.

7.2 OPC UA

TCP TCP/IP — универсальный протокол, обеспечивающий полнодуплексную связь между двумя приложениями. Сокет — это TransportConnection в реализации TCP/IP протокола соединения OPC UA.

Схема URL-адресов для конечных точек, использующих OPC UA TCP, — opc.tcp.

TransportProfileUri должен быть URI для транспорта TCP (см. [1]).

7.3 OPC UA HTTPS

7.3.1 Общие положения

HTTPS относится к HTTP-сообщениям, которыми обмениваются через соединение SSL/TLS (см. HTTPS). Синтаксис HTTP-сообщений не меняется, и единственное отличие состоит в том, что вместо соединения TCP/IP создается соединение TLS. Это означает, что профили, использующие этот транспорт, также могут использоваться HTTP, когда безопасность не имеет значения.

HTTPS — это протокол, обеспечивающий транспортную безопасность. Это означает, что все байты защищены при отправке без учета границ сообщения. Транспортная безопасность может работать

только для связи «точка-точка» и не допускает использования ненадежных посредников или прокси-серверов для обработки трафика.

SecurityPolicy должен быть указан, однако он влияет только на алгоритмы, используемые для подписи одноразовых номеров во время рукопожатия CreateSession/ActivateSession. Политика безопасности: слово «нет» указывает, что одноразовые номера не нужно подписывать. SecurityMode имеет значение Sign, если SecurityPolicy имеет значение «нет»; в этом случае SecurityMode должен быть установлен на «нет». Если UserIdentityToken подлежит шифрованию, он должен быть явно указан в UserTokenPolicy.

Если доступно несколько вариантов политики безопасности, заголовок HTTP под наименованием «OPCUA-SecurityPolicy» применяется клиентом, чтобы сообщить серверу, какую политику данный клиент использует. Значение заголовка — это URI для SecurityPolicy. Если клиент опускает заголовок, то сервер должен предположить, что SecurityPolicy имеет значение None. Все соединения HTTPS через URL-адрес должны рассматриваться как один SecureChannel, который используется несколькими клиентами. Стеки должны предоставлять уникальный идентификатор SecureChannel, который позволяет приложениям сопоставлять запрос с SecureChannel. Это означает, что сеансы можно считать безопасными только в том случае, если AuthenticationToken (см. ГОСТ Р 71809) длиной более 20 байт и включено шифрование HTTPS.

Алгоритмы шифрования, используемые HTTPS, не относятся к EndpointDescription. SecurityPolicy определяются применением политик, установленных для HTTPS, и выходят за рамки OPC UA.

На рисунке 16 показано несколько сценариев применения транспорта HTTPS.

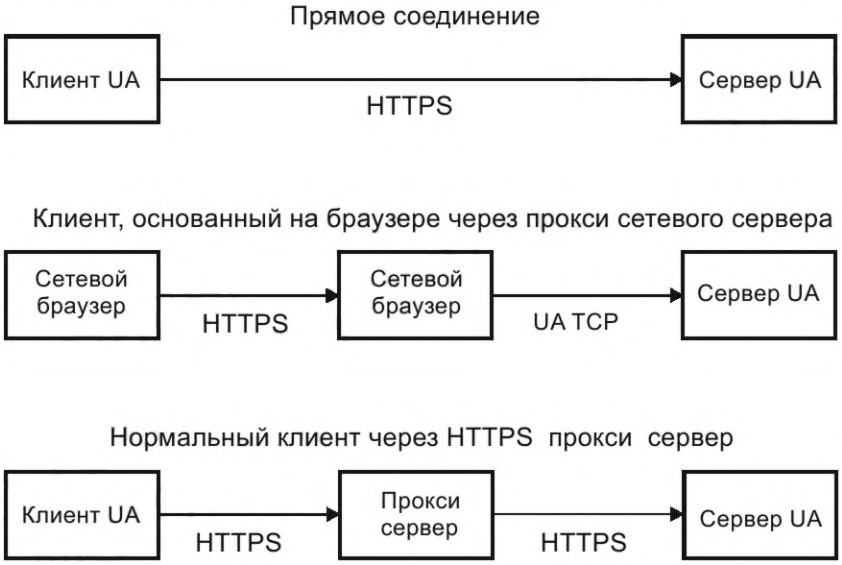

Рисунок 16 — Сценарии применения транспорта HTTPS

В некоторых сценариях связь HTTPS зависит от посредника, которому приложения не доверяют. В этом случае для обеспечения безопасности транспорт HTTPS не может использоваться. Прежде чем устанавливать связь с OPC UA, приложениям необходимо установить безопасный туннель, например VPN.

Приложения, поддерживающие транспорт HTTPS, должны поддерживать HTTP 1.1 и SSL/TLS 1.0.

Ряд реализаций HTTPS требует, чтобы все серверы имели сертификат с общим именем (CN), совпадающим с именем сервера DNS. Это означает, что серверу с несколькими именами DNS потребуется несколько сертификатов HTTPS. В связи с различиями сертификатов HTTPS и сертификатов приложений допускается использование общих сертификатов HTTPS, если несколько серверов находятся на одном компьютере. Приложения, которые используют транспорт HTTPS и требуют аутентификации приложений, должны проверять сертификаты приложений во время транзакций CreateSession/ ActivateSession.

Допускается автоматическое генерирование сертификатов HTTPS, однако данная опция может быть причиной ошибок клиентов, работающих в ограниченной среде, например в веб-браузере. Сер-

тификаты HTTPS должны выдаваться органом, который принимается всеми веб-браузерами, которым необходим доступ к серверу. Набор ЦС, принимаемых веб-браузерами, определяется организацией, управляющей клиентскими компьютерами. Клиентские приложения, которые не работают в сети, могут использовать список доверия, используемый для сертификатов приложений.

HTTPS-соединения имеют непрогнозируемое время жизни. Таким образом, серверам необходимо полагаться на AuthenticationToken, переданный в RequestHeader, для определения личности клиента. Это означает, что AuthenticationToken должен представлять собой случайно сгенерированное значение, содержащее не менее 32 байтов данных, и неизменно должен использоваться HTTPS с подписью и шифрованием.

HTTPS позволяет клиентам иметь сертификаты; однако они не требуются для транспорта HTTPS. Сервер должен позволять клиентам подключаться без предоставления сертификата во время согласования HTTPS-соединения.

HTTP 1.1 поддерживает фрагментирование сообщений, где для заголовка Content-Length в ответе на запрос установлено значение chunked, а перед каждым фрагментом указывается его размер в байтах. Все приложения, поддерживающие транспорт HTTPS, должны поддерживать фрагментирование HTTP.

Схема URL-адресов применяется для конечных точек, использующих транспорт HTTPS, — opc.https, где «https» — общая схема URL-адресов для базового транспорта. Префикс «opc» указывает, что конечная точка принимает сообщения OPC UA, как определено в 7.3.

7.3.2 Сервисы без сеансов

Службы без сеансов в соответствии с ГОСТ Р 71809 можно вызывать через HTTPS POST. Заголовок авторизации HTTP в запросе должен иметь токен носителя, который является токеном доступа и предоставляется службой авторизации.

Тип контента для тел, закодированных в двоичном формате OPC UA, для служб без сеансов отличается от типа контента для служб на основе сеансов, указанных в 7.3.4. Тип контента протокола HTTP должен указывать кодировку тела. Если типом контента является application/opcua+uabinary, то тело кодируется с использованием двоичного кодирования OPC UA (см. 7.3.4). Если тип контента — application/opcua+uajson, то тело кодируется с использованием обратимой формы кодировки JSON (см. 7.3.5).

7.3.3 XML-кодирование

TransportProtocol реализует службы OPC UA с использованием шаблона сообщений запроса-ответа SOAP через соединение HTTPS.

Телом HTTP-сообщений должно быть сообщение SOAP часть 1 (см. SOAP 1.2). Заголовки WS-Addressing не являются обязательными (см. «адресация WS»).

Кодирование XML OPC UA определяет способ представления сообщения OPC UA в виде элемента XML. Данный элемент добавляется в сообщение SOAP как единственный дочерний элемент тела SOAP. Если при анализе тела запроса на сервере возникает ошибка, сервер может возвратить ошибку SOAP или ответ об ошибке OPC UA.

Действие SOAP, связанное с сообщением запроса в кодировке XML, неизменно имеет форму, включающую имя вызываемой службы OPC UA:

http://opcfoundation.org/UA/2008/02/Services.wsdl/

Действие SOAP, связанное с ответным сообщением в формате XML, также неизменно имеет форму:

http://opcfoundation.org/UA/2008/02/Services.wsdl/

Все запросы должны быть запросами HTTP POST. Тип контента должен быть «application/soap+xml», при этом необходимо указать кодировку и параметры действия. Параметр charset должен быть «utf-8», а параметр действия должен быть URI для действия SOAP.

Пример заголовка протокола HTTP:

POST/UA/SampleServer HTTP/1.1

Content-Type: application/soap+xml; charset=”utf-8”; action=”http://opcfoundation.org/UA/2008/02/Services.wsdl/Read”

Content-Length: nnnn

Параметр действия отображается в той же строке, что и объявление Content-Type. Пример сообщения запроса:

<s:Envelope xmlns:s=”http://www.w3.org/2003/05/soap-envelope”>

<s:Body>

<ReadRequest xmlns=”http://opcfoundation.org/UA/2008/02/Types.xsd”>

…

</ReadRequest>
</s:Body>
</s:Envelope>
Пример заголовка ответа HTTP:
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"; action="http://opcfoundation.org/UA/2008/02/Services.wsdl/ReadResponse"
Content-Length: nnnn
Параметр действия отображается в той же строке, что и объявление Content-Type. Пример ответного сообщения:
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
<s:Body>
<ReadResponse xmlns="http://opcfoundation.org/UA/2008/02/Types.xsd">
…
</ReadResponse>
</s:Body>
</s:Envelope>

7.3.4 Двоичное кодирование OPC UA

Данный транспортный протокол реализует службы OPC UA с использованием двоичного кода OPC UA, которым обмениваются с помощью HTTPS-соединения.

Приложения, поддерживающие профиль HTTPS, должны поддерживать HTTP 1.1.

Тело HTTP-сообщений должно представлять собой двоичный объект OPC UA в двоичном кодировании. Тип контента должен быть типом «приложение/октет-поток».

Пример заголовка протокола HTTP:
POST/UA/SampleServer HTTP/1.1
Content-Type: application/octet-stream; Content-Length: nnnn
Пример заголовка ответа HTTP:
HTTP/1.1 200 OK
Content-Type: application/octet-stream; Content-Length: nnnn

Тело сообщения представляет собой структуру запроса или ответа, закодированную как ExtensionObject в двоичном формате OPC UA. Заголовок авторизации используют только для вызовов службы без сеанса (см. 7.3.2).

Если двоичное кодирование OPC UA применяют для службы без сеанса, заголовок протокола HTTP имеет вид:
POST/UA/SampleServer HTTP/1.1
Authorization: Bearer <base64-encoded-token-data>
Content-Type: application/opcua+uabinary;
Content-Length: nnnn

7.3.5 Кодирование JSON

Данный транспортный протокол реализует службы OPC UA с использованием сообщений в кодировке JSON, которыми обмениваются через соединение HTTPS.

Приложения, поддерживающие профиль HTTPS, должны поддерживать HTTP 1.1. Тело HTTP-сообщений должно быть закодировано OPC UAJSON. Тип контента должен быть типом «приложение/OPC UA+UAJSON».

Пример заголовка протокола HTTP:
POST/UA/Сервер выборок HTTP/1.1
Разрешение: на предъявителя <base64-encoded-token-data> Content-Type: приложение/OPC UA+UAJSON;
Content-Length: nnnn
Пример заголовка ответа HTTP:
HTTP/1.1 200 OK
Content-Type: приложение/OPC UA+UAJSON; Content-Length: nnnn

7.4 WebSockets

7.4.1 Общие положения

Данный транспортный протокол отправляет сообщения протокола соединения OPC UA через WebSockets.

WebSockets — двунаправленный протокол для связи через веб-сервер, который обычно используется приложениями на основе браузера, чтобы позволить веб-серверу асинхронно отправлять информацию клиенту. WebSockets применяет тот же порт по умолчанию, что и HTTP или HTTPS и инициирует связь с помощью протокола HTTP. Применение данного порта делает его наиболее востребованным в средах, где брандмауэры ограничивают трафик портами, используемыми HTTP или HTTPS.

WebSockets использует HTTP, однако на практике соединение WebSocket инициируется только с помощью запроса HTTP GET, а веб-сервер предоставляет ответ HTTP. После этого обмена весь трафик использует протокол двоичного кодирования.

Сервер, поддерживающий транспорт WebSockets, должен опубликовать одну или несколько конечных точек со схемой opc.wss. TransportProfileUri должен быть одним из URI для транспорта WebSockets (см. [1]). TransportProfileUri указывает протокол кодирования и безопасности, используемый для создания сообщений OPC UA, отправляемых через WebSocket.

SecurityMode и SecurityPolicyUri конечной точки управляют безопасностью, применяемой к сообщениям, отправленным через WebSocket. Данная технология обеспечивает безопасность сообщений, даже если соединение WebSocket установлено через ненадежные прокси-серверы HTTPS.

На рисунке 17 представлено отображение протокола opcua+uacp (см. 7.4.2) в виде полного процесса установления связи через WebSocket (см. [11]).

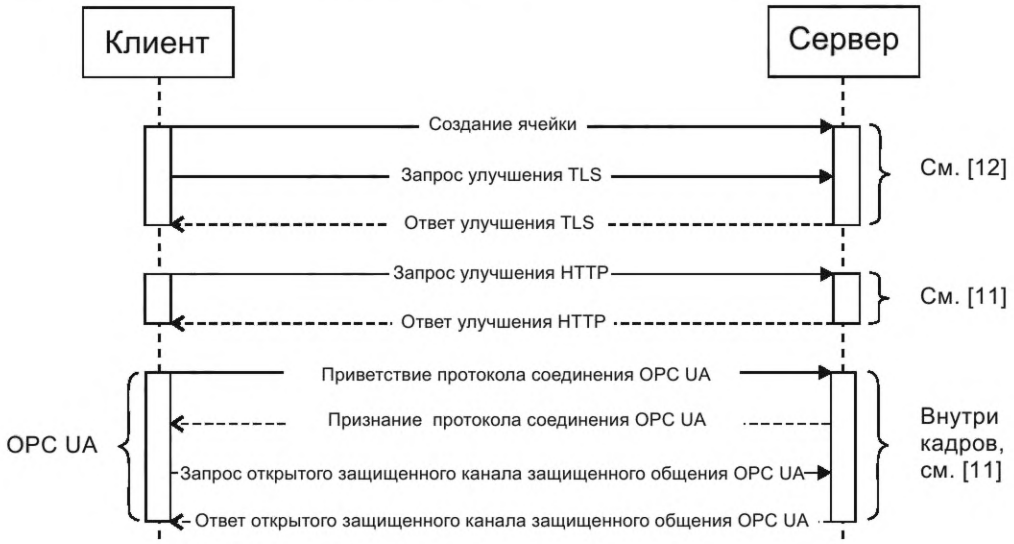

Рисунок 17 — Настройка связи через WebSocket

7.4.2 Сопоставление протоколов

Протокол WebSocket позволяет клиентам запрашивать, чтобы серверы использовали определенные подпротоколы с заголовком «Sec-WebSocket-Protocol» в подтверждении WebSocket (см. [11]). Протоколы в настоящем стандарте приведены в таблице 61.

Таблица 61 — Сопоставления протоколов WebSocket

Протокол	Описание протоколов
opcua+uacp	Каждый кадр WebSocket представляет собой MessageChunk, как определено в 6.7.2. После создания WebSocket «рукопожатие», описанное в 7.1.3, используется для согласования максимального размера MessageChunk. Максимальный размер буфера, необходимый для приема кадра WebSocket, равен максимальной длине MessageChunk плюс максимальный размер заголовка кадра WebSocket. При использовании этого протокола необходимая нагрузка в каждом кадре является двоичной (OpCode 0x2 см. в [11])
opcua+uajson	Каждый кадр WebSocket представляет собой сообщение, закодированное с использованием кодировки JSON, описанной в 5.4.9. Не существует механизма согласования максимального размера кадра. Если получатель обнаруживает кадр, превышающий его внутренние ограничения, он должен закрыть соединение WebSocket и предоставить код состояния 1009, как описано в [11]. При использовании этого протокола необходимая нагрузка в каждом кадре представляет собой текст (OpCode 0x1 см. в [11])

Каждое определенное сопоставление протокола WebSocket имеет TransportProfileUri (см. [1]). Клиент запрашивает протокол. Если сервер не поддерживает запрошенный клиентом протокол, клиент должен закрыть соединение и сообщить об ошибке.

7.4.3 Безопасность

WebSockets требует, чтобы у сервера был сертификат, однако у клиента может быть сертификат. Сертификат сервера должен содержать имя домена как компонент общего имени субъекта; однако клиенты могут переопределить сертификат.

Процедура проверки допускает решение о принятии сертификатов с несоответствием домена.

При использовании транспорта WebSockets из веб-браузера среда браузера может устанавливать дополнительные ограничения. Например, веб-браузер может потребовать у сервера действительный сертификат TLS, выданный ЦС и установленный в списке доверия для веб-браузера.

Для поддержки клиентов сервер может использовать сертификат TLS, который не соответствует требованиям для сертификата ApplicationInstance. В этих случаях сертификат TLS применяется только для согласования TLS, а сервер должен использовать действительный сертификат ApplicationInstance при другом взаимодействии, при необходимости. Серверы должны позволять администраторам указывать сертификат для использования с TLS, отличным от сертификата ApplicationInstance.

Клиенты, работающие в среде браузера, указывают HTTP-заголовок Origin во время подтверждения обновления WebSocket.

Серверы должны возвращать Access-Control-Allow-Origin, чтобы указать, что соединение разрешено.

Любой клиент, который не работает в среде веб-браузера и поддерживает транспорт WebSockets, должен принимать сертификат экземпляра приложения OPC UA в качестве сертификата TLS при условии, что в поле subjectAltName указан правильный домен.

Клиенту дано право использовать свой сертификат экземпляра приложения в качестве сертификата TLS, и серверы должны принимать эти сертификаты, если они действительны в соответствии с правилами проверки сертификата OPC UA.

Некоторые операционные системы не предоставляют приложению контроль над набором алгоритмов, с которыми будет согласовываться TLS. В таких случаях этот набор будет основан на потребностях веб-браузеров и не будет соответствовать потребностям приложения OPC UA. В случае необходимости приложениям следует использовать безопасный диалог OPC UA в дополнение к TLS.

Клиенты, поддерживающие транспорт WebSockets, должны поддерживать явную настройку прокси-сервера HTTPS. При использовании прокси-сервера HTTPS клиент должен сначала отправить сообщение HTTP CONNECT перед началом установления связи по протоколу WebSockets. Следует иметь в виду, что явные прокси HTTPS допускают атаки по типу «человек посередине». Эту угрозу можно смягчить, используя безопасный диалог OPC UA в дополнение к TLS.

7.5 Известные адреса

Локальный сервер обнаружения (LDS) — сервер OPC UA, который реализует набор служб обнаружения согласно ГОСТ Р 71809. Если LDS установлен на машине, он должен использовать один или несколько общеизвестных адресов, определенных в таблице 62.

Таблица 62 — Известные адреса для локальных серверов обнаружения

Транспортное сопоставление	URL	Замечания
OPC UA TCP	opc.tcp://localhost:4840/UADiscovery	—
OPC UA WebSockets	opc.wss://localhost:443/UADiscovery	—
OPC UA HTTPS	https://localhost:443/UADiscovery	—

Приложения OPC UA, использующие LDS, должны позволять администраторам изменять общеизвестные адреса, используемые в системе.

Конечная точка, используемая серверами для регистрации в LDS, должна представлять собой базовый адрес с добавленным к нему путем «/registration» (например, http://localhost/UADiscovery/registration). Серверы OPC UA должны позволять администраторам настраивать адрес, используемый для регистрации.

Каждое приложение серверов OPC UA реализует набор служб обнаружения. Если серверу OPC UA требуется другой адрес для этой конечной точки, он должен создать адрес, добавив путь «/discovery» к своему базовому адресу.

8 Нормативные контракты

8.1 Бинарная схема OPC

Нормативным контрактом для сообщений, закодированных в двоичном формате OPC UA, является двоичная схема OPC. Данный файл определяет структуру всех типов и сообщений. Синтаксис схемы двоичного типа OPC описан в [7]. Бинарная схема фиксирует нормативные имена типов и их полей, а также порядок появления полей при кодировании. Также фиксируется тип данных каждого поля.

8.2 XML-схема и WSDL

Нормативным контрактом для сообщений, закодированных в формате OPC UA XML, является схема XML. Данный файл определяет структуру всех типов и сообщений. Схема XML фиксирует нормативные имена типов и их полей, а также порядок появления полей при кодировании. Также фиксируется тип данных каждого поля.

Нормативный контракт для сообщения, отправляемого через транспортный протокол SOAP/HTTP, представляет собой WSDL, который включает схему XML для сообщений, закодированных в формате XML OPC UA. Он определяет типы портов для серверов OPC UA и DiscoveryServers.

8.3 Константы

В приложении А представлены константы для идентификаторов атрибутов, кодов состояния и числовых полей узлов.

8.4 Конфигурация безопасности

В приложении Б представлено описание схемы для настройки безопасности.

8.5 Схема информационной модели

В приложении В определена схема на основе XML, которая используется для информационных моделей.

Приложение А
(обязательное)

Константы

А.1 Идентификаторы атрибутов

В таблице А.1 приведены идентификаторы, присвоенные атрибутам.

Таблица А.1 — Идентификаторы, присвоенные атрибутам

Атрибуты	Идентификатор
NodeId	1
NodeClass	2
BrowseName	3
DisplayName	4
Description	5
WriteMask	6
UserWriteMask	7
IsAbstract	8
Symmetric	9
InverseName	10
ContainsNoLoops	11
EventNotifier	12
Value	13
DataType	14
ValueRank	15
ArrayDimensions	16
AccessLevel	17
UserAccessLevel	18
MinimumSamplingInterval	19
Historizing	20
Executable	21
UserExecutable	22
DataTypeDefinition	23
RolePermissions	24
UserRolePermissions	25
AccessRestrictions	26
AccessLevelEx	27

А.2 Коды состояния

В настоящем разделе определены числовые идентификаторы для кодов состояния, установленных спецификацией OPC UA. Идентификаторы указывают в файле CSV со следующим синтаксисом:

<SymbolName>, <Code>, <Description>,

где SymbolName — буквальное имя кода ошибки, который появляется в спецификации, а Code — это шестнадцатеричное значение кода статуса согласно ГОСТ Р 71809. Если LDS установлен на машине, он должен использовать один или несколько кодов. Значимость, связанная с конкретным кодом, определена префиксом («хорошо», «неопределенно» или «плохо»). Description — описание кодов, установленных спецификацией OPC UA.

А.3 Числовые идентификаторы узлов

В настоящем разделе определены числовые идентификаторы для числовых идентификаторов NodeId, установленных спецификацией OPC UA. В файле CSV идентификаторы указывают со следующим синтаксисом:

<SymbolName>, <Identifier>, <NodeClass>,

где SymbolName — это либо имя браузера узла типа, либо путь браузера для узла экземпляра, который указан в спецификации, Identifier — числовое значение для NodeId, а NodeClass — класс узла.

BrowsePath для узла экземпляра создается путем добавления BrowserName узла экземпляра к BrowseName для содержащего его экземпляра или типа. Символ «_» используется для разделения каждого имени браузера в пути. Имя символа для объявления экземпляра NamespaceArray в объявлении ServerType: ServerType_NamespaceArray. В ГОСТ Р 71810 определен стандартный экземпляр типа объекта ServerType с именем браузера «Сервер». BrowseName для свойства NamespaceArray стандартного объекта сервера: Server_NamespaceArray.

А.4 Набор узлов OPC UA

Набор узлов OPC UA включает полную информационную модель и соответствует синтаксису схемы информационной модели XML, приведенной в А.6, и может быть прочитан и обработан компьютерной программой.

А.5 Объявления типов для собственного сопоставления OPC UA

В настоящем разделе определяется двоичное кодирование OPC UA для всех типов данных и сообщений, приведенных в настоящем стандарте. Схема, используемая для описания типа, установлена в ГОСТ Р 71808.

А.6 XML-схема

В настоящем разделе определена XML-схема для всех типов данных и сообщений, установленных в настоящем стандарте.

А.7 Типы портов WDSL

В настоящем разделе определены операции WSDL и типы портов для служб в соответствии с ГОСТ Р 71809. WSDL импортирует XML-схему, представленную в А.6.

А.8 Привязки WSDL

В настоящем разделе определены привязки WSDL для служб в соответствии с ГОСТ Р 71809.

Приложение Б
(обязательное)

Управление настройками безопасности

Б.1 Общие сведения

Все приложения OPC UA должны поддерживать безопасность, однако это требование означает, что администраторам необходимо настроить параметры безопасности для приложения OPC UA в соответствии с ГОСТ Р 59799. В данном приложении представлено описание XML-схемы, которую можно использовать для чтения и обновления настроек безопасности для приложения OPC UA. Все приложения OPC UA могут поддерживать настройки с помощью импорта/экспорта документов, соответствующих схеме SecuredApplication.

Схема SecuredApplication может поддерживаться двумя способами:

а) предоставление файла конфигурации XML, который можно редактировать напрямую;

б) предоставление утилиты импорта/экспорта, которую можно применять по мере необходимости.

Если приложение поддерживает прямое редактирование файла конфигурации XML, этот файл должен содержать ровно один элемент с локальным именем схемы SecuredApplication и URI, равным URI схемы SecuredApplication. Сторонняя утилита конфигурации должна иметь возможность анализировать XML, читать и обновлять элемент схемы SecuredApplication. Администратор должен гарантировать, что только авторизованные администраторы могут обновлять этот файл.

Например, конфигурация, которую допустимо редактировать напрямую имеет вид:

```
<s1:SampleConfiguration xmlns:s1=”http://acme.com/UA/Sample/Configuration.xsd”>
<ApplicationName>ACME UA Server</ApplicationName>
<ApplicationUri>urn:myfactory.com:Machine54:ACME UA Server</ApplicationUri>
<!-- any number of app<SecuredApplication
xmlns=”http://opcfoundation.org/UA/2011/03/SecuredApplication.xsd”>
<ApplicationName>ACME UA Server</ApplicationName>
<ApplicationUri>urn:myfactory.com:Machine54:ACME UA Server</ApplicationUri>
<ApplicationType>Server_0</ApplicationType>
<ApplicationCertificate>
<StoreType>Windows</StoreType>
<StorePath>LocalMachine\My</StorePath>
<SubjectName>ACME UA Server</SubjectName>
</ApplicationCertificate>
</SecuredApplication>

<!-- any number of application specific elements -->

<DisableHiResClock>true</DisableHiResClock>
</s1:SampleConfiguration>
```

Если приложение предоставляет утилиту импорта/экспорта, то файл импорта/экспорта должен соответствовать схеме SecuredApplication. Администратор должен гарантировать, что только авторизованные администраторы могут запускать утилиту. Например, файл, используемый утилитой импорта/экспорта, имеет вид:

```
<?xml version=»1.0» encoding=»utf-8»?>
<SecuredApplication xmlns=”http://opcfoundation.org/UA/2011/03/SecuredApplication.xsd”>
<ApplicationName>ACME UA Server</ApplicationName>
<ApplicationUri>urn:myfactory.com:Machine54:ACME UA Server</ApplicationUri>
<ApplicationType>Server_0</ApplicationType>
<ConfigurationMode>urn:acme.com:ACME Configuration Tool</ConfigurationMode>
<LastExportTime>2011-03-04T13:34:12Z</LastExportTime>
<ExecutableFile>%ProgramFiles%\ACME\Bin\ACME UA Server.exe</ExecutableFile>
<ApplicationCertificate>
<StoreType>Windows</StoreType>
<StorePath>LocalMachine\My</StorePath>
<SubjectName>ACME UA Server</SubjectName>
</ApplicationCertificate>
<TrustedCertificateStore>
<StoreType>Windows</StoreType>
<StorePath>LocalMachine\UA applications</StorePath>
<!-- Offline CRL Checks by Default -->
<ValidationOptions>16</ValidationOptions>
```

```
</TrustedCertificateStore>
<TrustedCertificates>
<Certificates>
<CertificateIdentifier>
<SubjectName>CN=MyFactory CA</SubjectName>
<!-- Online CRL Check for this CA -->
<ValidationOptions>32</ValidationOptions>
</CertificateIdentifier>
</Certificates>
</TrustedCertificates>
<RejectedCertificatesStore>
<StoreType>Directory</StoreType>
<StorePath>%CommonApplicationData%\OPC Foundation\RejectedCertificates</StorePath>
</RejectedCertificatesStore>
</SecuredApplication>
```

Б.2 Элемент схемы SecuredApplication

Параметры защищенности приложения определяют элементы схемы SecuredApplication. Элементы, содержащиеся в схеме SecuredApplication, описаны в таблице Б.1.

Таблица Б.1 — Защищенное приложение в схеме SecuredApplication

Элемент	Тип	Описание элементов приложения
ApplicationName	String	Имя приложения, понимаемое идентификатором. Приложения должны позволять читать или изменять это значение
ApplicationUri	String	Глобальный уникальный идентификатор экземпляра приложения. Приложения должны позволять читать или изменять это значение
ApplicationType	ApplicationType	Тип приложения. Может быть один из следующих: - сервер_0; - клиент_1; - клиентИСервер_2; - дискавериСервер_3. Приложение должно предоставить это значение. Приложения не позволяют изменять это значение
ProductName	String	Наименование продукта. Приложение должно предоставить это значение. Приложения не позволяют изменять это значение
ConfigurationMode	String	Параметр указывает, как должно быть настроено приложение. Пустое или отсутствующее значение указывает на то, что файл конфигурации можно редактировать напрямую. В этом случае необходимо указать местоположение файла конфигурации. Другое значение представляет собой URI, идентифицирующий утилиту настройки. В документации поставщика должно быть объяснено, как использовать данную утилиту. Приложение должно предоставить это значение. Приложения не позволяют изменять это значение
LastExportTime	UtcTime	Когда конфигурация была экспортирована утилитой импорта/экспорта. Данный элемент можно не учитывать, если приложения допускают прямое редактирование конфигурации безопасности
ConfigurationFile	String	Полный путь к файлу конфигурации, используемому приложением. Приложения не предоставляют данное значение, если используется утилита импорта/экспорта. Приложения не позволяют изменять данное значение. Разрешения, установленные для этого файла, определяют, кто имеет права изменять конфигурацию приложения

Продолжение таблицы Б.1

Элемент	Тип	Описание элементов приложения
ExecutableFile	String	Полный путь к исполняемому файлу приложения. Приложения могут не предоставлять это значение. Приложения не позволяют изменять это значение. Разрешения, установленные для этого файла, определяют, кто имеет права на запуск приложения
ApplicationCertificate	CertificateIdentifier	Идентификатор сертификата экземпляра приложения. Приложения должны позволять читать или изменять это значение. Данный идентификатор может ссылаться на хранилище сертификатов, содержащее закрытый ключ. Если закрытый ключ недоступен внешним приложениям, данное значение должно содержать сертификат X.509 v3 для приложения. Если утилита настройки назначает новый закрытый ключ, это значение должно ссылаться на хранилище, в котором размещен закрытый ключ. Утилита импорта/экспорта может удалить этот закрытый ключ, если переместит его в безопасное место, доступное приложению. Приложения должны позволять вводить пароль, необходимый для доступа к закрытому ключу во время операции импорта. Приложения должны сообщать об ошибке, если ApplicationCertificate недействителен
TrustedCertificateStore	CertificateStore Identifier	Расположение хранилища сертификатов, содержащего сертификаты приложений, или ЦС, которым можно доверять. Приложения должны позволять читать или изменять это значение. Данное значение должно быть ссылкой на физическое хранилище, которым можно управлять отдельно от приложения. Приложения, поддерживающие общие физические хранилища, должны проверять это хранилище на наличие изменений при каждой проверке сертификата. Администратор несет ответственность за проверку подписи на всех сертификатах, размещенных в данном хранилище. Это означает, что приложение может доверять сертификатам в этом хранилище, даже если их невозможно проверить до доверенного корня. Администраторы должны размещать любые сертификаты ЦС, используемые для проверки подписи, в IssuerStore или IssuerList. Приложение должно проверить статус отзыва сертификатов, если сертификат был выдан ЦС. Приложение должно искать CRL в хранилище, где оно обнаружило сертификат ЦС. Местоположение онлайнового CRL для ЦС должно быть указано с помощью расширения сертификата CRLDistributionPoints (OID= 2.5.29.31) X.509 v3. Параметр ValidationOptions используется для указания списка отзывов, применяемых для ЦС в этом хранилище

Продолжение таблицы Б.1

Элемент	Тип	Описание элементов приложения
TrustedCertificates	CertificateList	Список сертификатов приложений для ЦС, которым можно доверять. Значение представляет собой явный список сертификатов, который является личным для приложения. Он используется, когда приложение не поддерживает общие физические хранилища сертификатов или когда администраторам необходимо указать ValidationOptions для сертификатов. Если указаны TrustedCertificateStore и TrustedCertificates, то приложение должно использовать TrustedCertificateStore для проверки доверительных отношений. Элемент TrustedCertificates применяется только для поиска ValidationOptions для отдельных сертификатов. Его также можно использовать для предоставления CRL для сертификатов ЦС. Если TrustedCertificateStore не указан, элемент TrustedCertificates должен содержать полный сертификат X.509 v3 для каждой записи
IssuerStore	CertificateStore Identifier	Расположение хранилища сертификатов, содержащего сертификаты ЦС, которые не являются доверенными, но необходимы для проверки подписей на сертификатах. Приложения должны позволять читать или изменять данные значения. Данное значение должно быть ссылкой на физическое хранилище, которым можно управлять отдельно от приложения. Приложения должны проверять хранилище на наличие изменений при каждой проверке сертификата. Это хранилище также может содержать CRL для ЦС
IssuerCertificates	CertificateList	Список сертификатов для ЦС, которым не доверяют, но которые необходимы для проверки подписей на сертификатах. Приложения должны позволять читать или изменять это значение. Данное значение представляет собой явный список сертификатов, который является личным для приложения. Он используется, когда приложение не поддерживает общие физические хранилища сертификатов или когда администраторам необходимо указать ValidationOptions для отдельных сертификатов. Если указаны оба элемента IssuerStore и IssuerCertificates, то приложение должно использовать IssuerStore для проверки подписей. Параметр IssuerCertificates применяется только для поиска ValidationOptions для отдельных сертификатов. Его также можно использовать для предоставления CRL для сертификатов ЦС
RejectedCertificates Store	CertificateStore Identifier	Расположение общего хранилища сертификатов, содержащего сертификаты отклоненных приложений. Приложения должны позволять читать или изменять это значение. Приложения должны добавлять сертификат, закодированный в DER, в это хранилище каждый раз, когда оно отклоняет сертификат, поскольку он не является доверенным или если он не соответствует одному из правил проверки, которые могут быть подавлены (см. Б.6). Приложения не должны добавлять сертификат в это хранилище, если он был отклонен по причине, которую невозможно скрыть (например, сертификат отозван)

Окончание таблицы Б.1

Элемент	Тип	Описание элементов приложения
BaseAddresses	String	Список URL-адресов конечных точек, поддерживаемых сервером. Приложения должны позволять читать или изменять эти значения. Если сервер не поддерживает схему URL-адреса, он должен ее игнорировать. Этот список может содержать несколько записей для одной и той же схемы URL-адресов. Первой записью схемы является базовый URL-адрес. Остальные считаются псевдонимами DNS, указывающими на первый URL-адрес. Администратор несет ответственность за настройку сети для правильной маршрутизации этих псевдонимов
SecurityProfileUris	SecurityProfile	Список SecurityPolicyUris, поддерживаемых сервером. URI определены как профили безопасности (см. [1]). Приложения должны позволять читать или изменять эти значения, а также позволять изменять флаг «Включено» для каждого SecurityProfile, который он поддерживает. Если флаг Enabled имеет значение false, сервер не должен разрешать соединения с использованием SecurityProfile. Если сервер не поддерживает SecurityProfile, он должен его игнорировать
Extensions	xs:any	Список расширений, определенных поставщиком, прикрепленных к настройкам безопасности. Приложения должны игнорировать расширения, которые они не распознают. Приложения, обновляющие файл, содержащий расширения, не должны удалять или изменять расширения, которые они не распознают

Когда элемент схемы SecuredApplication импортируется в приложение, приложение обновляет свою конфигурацию на основе содержащейся в ней информации. Если при импорте возникают неисправимые ошибки, приложение не должно вносить какие-либо изменения в свою конфигурацию и информировать о причине ошибки.

Механизм, используемый для импорта или экспорта конфигурации, зависит от приложения. Приложения должны гарантировать, что только авторизованные пользователи смогут получить доступ к этой функции.

Элемент схемы SecuredApplication может ссылаться на сертификаты X.509 v3, которые содержатся в физических хранилищах. Относительно каждого приложения следует определить, какие хранилища будут использованы: общие физические хранилища, которыми администратор может управлять напрямую, изменяя местоположение, или частные хранилища, доступ к которым возможен только через утилиту импорта/экспорта. Если в приложении использованы частные хранилища, то при экспорте содержимое этих частных хранилищ копируется в файл экспорта. Если файл импорта ссылается на общие физические хранилища, то утилита импорта/экспорта должна скопировать содержимое этих хранилищ в частные хранилища.

Утилита импорта/экспорта не должна экспортировать закрытые ключи. Если администратор решает назначить приложению новый общедоступный закрытый ключ, он должен поместить личный ключ в хранилище, где к нему сможет получить доступ утилита импорта/экспорта. Затем утилита импорта/экспорта отвечает за обеспечение безопасного перемещения файла в то место, где приложение может получить к нему доступ.

Б.3 CertificateIdentifier

Элемент CertificateIdentifier описывает сертификат X.509 v3. Сертификат может быть предоставлен явно внутри элемента, или элемент может указать расположение хранилища сертификатов, содержащего сертификат. Элементы, содержащиеся в CertificateIdentifier, описаны в таблице Б.2.

Таблица Б.2 — Структура элемента CertificateIdentifier

Элемент	Тип	Описание
StoreType	String	Тип хранилища сертификатов, содержащего сертификат. Предопределенные значения: «Окна» и «Каталог». Если отсутствует ссылка на элемент StoreType, должен быть указан элемент RawData

Окончание таблицы Б.2

Элемент	Тип	Описание
StorePath	String	Путь к хранилищу сертификатов. Синтаксис зависит от элемента StoreType. Если отсутствует ссылка на элемент StorePath, должен быть указан элемент RawData
SubjectName	String	Имя субъекта сертификата. Компонент общего имени (CN) имени субъекта. Имя субъекта представлено в виде строки (см. [13], раздел 3). Значения, не содержащие символов «=», считают компонентом общего имени
Thumbprint	String	Сертифицированный дайджест для сертификата в формате шестнадцатеричной строки. Случай не существенный
RawData	ByteString	Сертификат в кодировке DER. CertificateIdentifier является недействительным, если информация в сертификате DER конфликтует с информацией, указанной в других полях. Утилиты импорта должны отклонять конфигурации, содержащие недействительные сертификаты. Если указаны StoreType и StorePath, на данное поле не должно быть ссылки
OfflineRevocationList	ByteString	CRL, связанный с сертификатом эмитента. Формат CRL приведен в [4]. Данное поле имеет значение только для сертификатов эмитента
OnlineRevocationList	String	URL-адрес онлайн-списка отзыва, связанного с сертификатом эмитента. Данное поле имеет значение только для сертификатов эмитента

Элемент StoreType, представляемый в Windows, указывает хранилище сертификатов Windows. Синтаксис StorePath имеет следующую форму:

[\\HostName\]StoreLocation[\(ServiceName | UserSid)]\StoreName, где:

HostName — имя компьютера, на котором находится хранилище.

StoreLocation — одно из значений LocalMachine,

CurrentUser, User или Service ServiceName — имя службы Windows.

UserSid — идентификатор безопасности для учетной записи пользователя Windows.

StoreName — название магазина (например, My, Root, Trust, CA и т. д.).

Примеры элемента StoreType со значением Windows:

\\MYPC\LocalMachine\My

\CurrentUser\Trust

\\MYPC\Service\My UA Server\UA applications

\User\S-1-5-25\Root

Элемент StoreType со значением Каталог указывает хранилище на диске, которое содержит файлы с сертификатами в кодировке DER. Имя файла — сертифицированный дайджест для сертификата. В хранилище Каталог могут быть размещены только открытые ключи. Для работы с Каталогом применяется элемент StorePath — абсолютный путь к файловой системе, синтаксис которого зависит от операционной системы.

Если хранилище Каталог включает в себя подкаталог «сертификаты», то он является структурированным хранилищем с подкаталогами, приведенными в таблице Б.3.

Таблица Б.3 — Структурированное представление каталога

Подкаталог	Описание каталога
certs	Содержит сертификаты X.509 v3 в кодировке DER. Файлы должны иметь расширение .der

Окончание таблицы Б.3

Подкаталог	Описание каталога
private	Содержит закрытые ключи. CSV может зависеть от приложения. Файлы в кодировке PEM должны иметь расширение .pem. Файлы в кодировке PKCS#12 должны иметь расширение .pfx. Имя корневого файла должно совпадать с именем соответствующего файла открытого ключа в каталоге сертификатов
crl	Содержит CRL в кодировке DER для всех сертификатов CA, найденных в каталогах certs или ca. Файлы должны иметь расширение .crl

Каждый сертификат уникально идентифицируется элементом отпечаток ИмяСубъекта. Данный элемент может быть использован для идентификации сертификата пользователем и может быть указан вместе с параметром отпечаток или с данными, не входящими в структурированное хранилище. Идентификаторы сертификатов обрабатываются в зависимости от того, где они используются. В общем случае рекомендуется указывать элемент ИмяСубъекта.

CRL содержит список сертификатов, выданных ЦС. Эти списки следует проверить, прежде чем приложение сможет доверять сертификату, выданному доверенным ЦС. Требуется проверка имеющихся списков на наличие в CRL. Формат таких списков приведен в [4].

Автономные списки отзыва сертификатов размещаются в локальном хранилище сертификатов вместе с сертификатом эмитента. Данные списки идентифицируют по URL-адресу.

Б.4 CertificateStoreIdentifier

Элемент CertificateStoreIdentifier описывает физическое хранилище, содержащее сертификаты X.509 v3. Элементы, содержащиеся в CertificateStoreIdentifier, приведены в таблице Б.4.

Таблица Б.4 — Структура элементов CertificateStoreIdentifier

Элемент	Тип	Описание
StoreType	String	Тип хранилища сертификатов, содержащего сертификат. Предопределенные значения: «Окна» и «Каталог»
StorePath	String	Путь к хранилищу сертификатов. Синтаксис зависит от элемента StoreType. См. Б.3 для описания синтаксиса различных элементов StoreTypes
ValidationOptions	Int32	Параметры, используемые при проверке сертификатов, содержащихся в хранилище. Возможные варианты описаны в Б.6

Все сертификаты размещаются в физическом хранилище, которое может быть защищено от несанкционированного доступа. Реализация такого хранилища может различаться и зависеть от приложения, инструмента разработки или операционной системы. Хранилище сертификатов может быть использовано многими приложениями на одном компьютере.

Каждое хранилище сертификатов идентифицируется элементами StoreType и StorePath. Тот же путь на разные машины идентифицируют разные хранилища.

Б.5 CertificateList

Элемент CertificateList представляет собой список сертификатов. Элементы, содержащиеся в списке сертификатов, приведены в таблице Б.5.

Таблица Б.5 — Список сертификатов

Элемент	Тип	Описание элементов
Certificates	CertificateIdentifier	Список сертификатов, содержащихся в списке доверия
ValidationOptions	Int32	Элементы, используемые при проверке сертификатов, содержащихся в хранилище. Данные элементы применяют только в отношении сертификатов, имеющих ValidationOptions. Установлен бит UseDefaultOptions. Возможные варианты описаны в Б.6.

Б.6 CertificateValidationOptions

Элементы CertificateValidationOptions управляют процессом проверки сертификата. Параметры проверки могут использоваться в отношении любого сертификата, если для хранилища или списка, содержащего сертификат, нет ограничений ValidationOptions. Возможные варианты проверки приведены в таблице Б.6. Особым ограничением является необоснованное проведение проверок с возможностью создать риски сохранения конфиденциальности информации. Подробно такие риски рассмотрены в [9].

Таблица Б.6 — Параметры проверки сертификата

Поле	Описание проверок
SuppressCertificateExpired	Игнорируют ошибки, связанные со сроком действия сертификата или его эмитентами
SuppressHostNameInvalid	Игнорируют несоответствия между именем хоста или ApplicationUri
SuppressRevocationStatusUnknown	Игнорируют ошибки, если список отзыва эмитента не найден
CheckRevocationStatusOnline	Проверяют статус отзыва онлайн. Если этот параметр установлен, валидатор будет искать URL-адрес точки распространения CRL в сертификате и использовать OCSP (см. [14]), чтобы определить, был ли сертификат отозван. Если точка распространения CRL недоступна, то валидатор будет искать автономные CRL, если установлен бит CheckRedictionStatusOffine. В противном случае проверка не будет выполнена. Данный параметр указан для сертификатов эмитента и используется при проверке сертификатов, выданных данным эмитентом
CheckRevocationStatusOffline	Проверяют статус отзыва в автономном режиме. Если установлено, то валидатор будет искать CRL в хранилище сертификатов, где находится ЦС. Сертификат найден. Проверка не завершена, если CRL не найден. Данный параметр указан для сертификатов эмитента и используется при проверке. Сертификаты, выданные этим эмитентом
UseDefaultOptions	Если установлено, будут использованы параметры CertificateValidationOptions из списка сертификатов. Если сертификат не принадлежит к списку сертификатов, то значение по умолчанию равно 0 для всех битов

Приложение В
(обязательное)

XML-схема информационной модели

В.1 Общие положения

Разработчики информационной модели определяют стандартные адресные пространства, которые реализованы большинством серверов. Существует потребность в стандартном синтаксисе, который разработчики информационных моделей могут использовать для формального определения своих моделей в форме, которую может прочитать компьютерная программа. Для этой цели в настоящем приложении определена схема на основе XML.

Представление схемы является формальным определением. В настоящем приложении рассмотрены только детали семантики, а не типы, которые описывают сами себя.

Применение данной схемы допустимо для сериализации (т. е. импорта или экспорта) произвольного набора узлов в адресном пространстве сервера. Сериализованную форму можно применять для сохранения состояния сервера для последующего использования сервером или для обмена с другими приложениями (например, для поддержки автономной конфигурации клиентом).

Данная схема определяет способ представления структуры узлов, который не предназначен для отражения многочисленных семантических правил, определенных в других частях системы стандартов OPC UA. Потребители данных, сериализованных с помощью этой схемы, должны обрабатывать входные данные, которые соответствуют схеме, однако не соответствуют спецификации OPC UA из-за одного или нескольких нарушений семантических правил.

Таблицы, определяющие типы данных в спецификации, имеют имена полей, начинающиеся со строчной буквы. Первая буква должна быть преобразована в верхний регистр, когда имена полей формально определены в элементе UANodeSet.

В.2 UANodeSet

Данный элемент является корнем документа и определяет набор узлов, их атрибутов и ссылок. Допускаются ссылки на узлы за пределами документа.

Структура элемента UANodeSet приведена в таблице В.1.

Таблица В.1 — UANodeSet

Элемент	Тип	Описание корня UANodeSet
NamespaceUris	UriTable	Список NamespaceUris, используемых в элементе UANodeSet
ServerUris	UriTable	Список ServerUris, используемых в элементе UANodeSet
Models	ModelTableEntry	Список моделей, определенных в элементе UANodeSet, а также любые зависимости, которые эти модели имеют
ModelUri	String	URI модели. URI должен быть одной из записей в таблице элемента NamespaceUris
Version	String	Версия модели, определенная в элементе UANodeSet, является удобочитаемой строкой, не предназначенной для программного сравнения
PublicationDate	DateTime	Когда модель определена в нескольких файлах элемента UANodeSet, значения параметров используются для сравнения
RolePermissions	RolePermissions	Список RolePermissions по умолчанию для всех узлов в модели
AccessRestrictions	AccessRestrictions	Ограничения доступа по умолчанию, которые применяются ко всем узлам в модели
RequiredModels	ModelTableEntry	Список зависимостей для модели. Если модель требует минимальной версии, необходимо указать элемент PublicationDate. Инструменты, которые пытаются разрешить эти зависимости, могут принимать любой элемент PublicationDate после установленной даты
Aliases	AliasTable	Список псевдонимов, используемых в элементе UANodeSet

Окончание таблицы В.1

Элемент	Тип	Описание корня UANodeSet
Extensions	xs:any	Элемент, содержащий любые определенные поставщиком расширения для элемента UANodeSet
LastModified	DateTime	Время последнего изменения документа
<choice>	UAObject, UAVariable, UAMethod UAView, UAObjectType, UAVariableType, UADataType, UAReferenceType	Узлы в элементе UANodeSet

Элемент NamespaceUris — список URI для пространств имен, используемых в элементе UANodeSet. Индексы пространства имен, применяемые в элементах NodeId, ExpandedNodeIds и QualifiedNames, идентифицируют элемент в данном списке. Первый индекс неизменно равен 1 (0 — это исключительно пространство имен OPC UA).

Элемент ServerUris — список URI для серверов, на которые приведены ссылки в элементе UANodeSet. Индекс ServerIndex в элементе ExpandedNodeIds идентифицирует элемент в данном списке. Первый индекс неизменно равен 1 (0 — исключительно текущий сервер).

Элемент Models определяет модели, которые формально определены элементом UANodeSet. Он включает информацию о версии, а также информацию о любых зависимостях, которые может иметь модель. Если модель определена в элементе UANodeSet, то файл также должен определять экземпляр ObjectType NamespaceMetadataType. Дополнительная информация приведена в ГОСТ Р 71810.

Псевдонимы представляют собой список строковых замен элементов NodeIds. Псевдонимы можно использовать, чтобы сделать файл более читабельным, разрешив использовать строку типа HasProperty вместо числового NodeId (i = 46). Псевдонимы не являются обязательными.

Расширения представляют собой XML-данные свободной формы, которые допустимо использовать для прикрепления данных, определенных поставщиком, к элементу UANodeSet.

B.3 UANode

UANode — абстрактный базовый тип для всех узлов, определяющий базовый набор атрибутов и ссылок. Для каждого NodeClass указаны подтипы согласно ГОСТ Р 71809. Каждый из данных подтипов устанавливает элементы XML и атрибуты для атрибутов OPC UA, специфичных для NodeClass. Поля Типа UANode приведены в таблице В.2.

Таблица В.2 — UANode

Элемент	Тип	Описание полей
NodeId	NodeId	NodeId, сериализованный как строка. Синтаксис сериализованной строки определен в 5.3.1.10
BrowseName	QualifiedName	QualifiedName, сериализованный как строка в форме: <индекс пространства имен>:<имя>, где NamespaceIndex относится к таблице элемента NamespaceUris
SymbolicName	String	Символическое имя узла, которое можно использовать в качестве имени класса/поля в автоматически создаваемом коде. Его следует указывать только в том случае, если BrowseName не представляется возможным использовать для этой цели. Это поле не отображается в AddressSpace и предназначено для использования инструментами проектирования. Разрешены только буквы, цифры или знак подчеркивания («_»)
WriteMask	WriteMask	Значение атрибута элемента WriteMask
UserWriteMask	WriteMask	Находится в схеме, но не используется
AccessRestrictions	AccessRestrictions	Элементы AccessRestrictions, применимые к Node
DisplayName	LocalizedText	Список отображаемых имен для узла в разных локалях. Для каждой локали должна быть только одна запись

Окончание таблицы В.2

Элемент	Тип	Описание полей
Description	LocalizedText	Список описаний узла в разных локалях. Для каждой локали должна быть только одна запись
Category	String	Список идентификаторов, примененных для группировки связанных элементом UANode для пользования инструментами, которые создают/редактируют файлы элемента UANodeSet
Documentation	String	Дополнительная нелокализованная документация для пользования инструментами, создающими/редактирующими файлы элемента UANodeSet
References	Reference	Список ссылок на узле Node
RolePermissions	RolePermissions	Список элементов RolePermissions для узла Node
Extensions	xs:any	Элемент, содержащий любые расширения элемента UANode, определенные поставщиком

Расширения представляют собой XML-данные свободной формы, которые можно использовать для прикрепления данных, определенных поставщиком, к элементу UANode.

Значения массива обозначают символом «[]», однако в схеме XML массивы отображаются в сложный тип, начинающийся с префикса ListOf.

Ожидается, что UANodeSet будет содержать множество UANode, которые ссылаются друг на друга. Инструменты, создающие UANodeSets, не должны добавлять элементы Reference для обоих направлений, чтобы минимизировать размер XML-файла. Инструменты, читающие UANodeSets, должны автоматически добавлять обратные ссылки, если только обратные ссылки не подходят с учетом ReferenceType. HasTypeDefinition и HasModellingRule — это два примера того, что нецелесообразно добавлять обратные ссылки.

Следует иметь в виду, что UANodeSet представляет собой коллекцию узлов в адресном пространстве. Это означает, что любые экземпляры должны включать полностью унаследованную InstanceDeclarationHierarchy, как определено в ГОСТ Р 71808.

В.4 Элемент ReferenceType

Элемент ReferenceType — указывает ссылку на узел. Задание может быть прямым или обратным. В UANodeSet должно быть только одно направление для каждой ссылки. Другое направление будет добавлено автоматически во время операции импорта. Поля элемента ReferenceType приведены в таблице В.3.

Таблица В.3 — Элемент ReferenceType

Элемент	Тип	Описание полей
NodeId	NodeId	Элемент NodeId цели ссылки, сериализованной в виде строки. Синтаксис сериализованной строки определен в 5.3.1.11 (ExpandedNodeId). Это значение можно заменить псевдонимом
ReferenceType	NodeId	NodeId ReferenceType, сериализованный как строка. Синтаксис сериализованной строки определен в 5.3.1.10 (NodeId). Это значение можно заменить псевдонимом
IsForward	Boolean	Если TRUE, ссылка является прямой ссылкой

В.5 RolePermission

Элемент RolePermission указывает разрешения и предоставленные роли для узла. Поля элемента RolePermission определены в таблице В.4.

Таблица В.4 — Разрешение роли

Элемент	Тип	Описание полей
NodeId	NodeId	Элемент NodeId роли, имеющей разрешения
Permissions	UInt32	Битовая маска, определяющая разрешения и предоставленные роли. Битовая маска определяет биты разрешений, определенные в ГОСТ Р 71808

B.6 UAType

Элемент UAType является подтипом UANode, определенного в B.3 как базовый тип для подтипов, определенных в таблице B.5.

Таблица B.5 — Узлы типа UANodeSet

Подтип	Описание полей
UAObjectType	Определяет узел ObjectType, как описано в ГОСТ Р 71808
UAVariableType	Определяет узел VariableType Node, как описано в ГОСТ Р 71808
UADataType	Определяет узел DataType Node, как описано в ГОСТ Р 71808
UAReferenceType	Определяет узел ReferenceType Node, как описано в ГОСТ Р 71808

B.7 UAInstance

Элемент UAInstance — подтип UANode, определенный в B.3 как базовый тип для подтипов, приведенных в таблице B.6. Поля типа UAInstance перечислены в таблице B.7. Подтипы UAInstance, которые имеют поля в дополнение к полям, приведенным в ГОСТ Р 71808, подробно описаны далее.

Таблица B.6 — Узлы экземпляра UANodeSet

Подтип	Описание подтипов
UAObject	Определяет Object Node, как описано в ГОСТ Р 71808
UAVariable	Определяет Variable Node, как описано в ГОСТ Р 71808
UAMethod	Определяет Method Node, как описано в ГОСТ Р 71808
UAView	Определяет View Node, как описано в ГОСТ Р 71808

Таблица B.7 — UAInstance

Элемент	Тип	Описание полей
Все поля типа UANode описаны в B.3		
ParentNodeId	NodeId	Элемент NodeId узла, который является родительским для узла в информационной модели. Это поле используется для указания того, что между узлом и его родителем существует тесная связь (например, когда родительский узел удаляется, дочерний узел также удаляется). Данная информация не отображается в AddressSpace и предназначена для использования инструментами проектирования

B.8 UAVariable

Элемент UAVariable — это подтип UAInstance. Он представляет собой узел переменной. Поля типа элемента UAVariable приведены в таблице B.8.

Таблица B.8 — Переменная UAVariable

Элемент	Тип	Описание полей
Все поля типа UAInstance описаны в B.7		
Value	Variant	Значение кодировки узла с использованием проводной кодировки XML UA
Translation	TranslationType []	Список переводов значения, если значение представляет собой LocalizedText или структуру, содержащую LocalizedTexts. Это поле может быть опущено. Если значение представляет собой массив, количество элементов в этом массиве должно соответствовать количеству элементов в значении. Лишние элементы игнорируют. Если значение является скаляром, то в этом массиве есть один элемент. Если Value является структурой, то каждый элемент содержит переводы для одного или нескольких полей, идентифицируемых по имени. Дополнительная информация приведена в строке TranslationType

Окончание таблицы В.8

Элемент	Тип	Описание полей
DataType	NodeId	Тип данных значения
ValueRank	ValueRank	Ценный ранг. Если не указано, значение по умолчанию — −1 (скаляр)
ArrayDimensions	ArrayDimensions	Количество измерений в значении массива
AccessLevel	AccessLevel	Уровень доступа
UserAccessLevel	AccessLevel	Находится в схеме, но не используется
MinimumSamplingInterval	Duration	Минимальный интервал выборки
Historizing	Boolean	Информация относительно архивирования истории

В.9 UAMethod

UAMethod является подтипом UAInstance, определенного в В.7. Он представляет собой узел метода. Поля типа UAMethod приведены в таблице В.9.

Таблица В.9 — UAMethod

Элемент	Тип	Описание полей
Поля типа UAInstance описаны в В.7		
MethodDeclarationId	NodeId	Имя поля может быть указано для узлов методов, которые применяются с целью использования ссылки HasComponent из одного узла объекта, например: NodeId UAMethod с тем же именем BrowseName, которое содержится в TypeDefinition, связанном с узлом объекта. Если TypeDefinition переопределяет метод, унаследованный от базового ObjectType, тогда этот атрибут должен ссылаться на узел метода в подтипе
UserExecutable	Boolean	Находится в схеме, но не используется
ArgumentDescription	UAMethodArgument []	Список описаний аргументов узла метода. Каждая запись имеет имя, которое однозначно идентифицирует аргумент. Для каждого имени должна быть только одна запись. Каждая запись содержит список описаний аргумента в разных локалях. Для каждого аргумента должна быть только одна запись для каждой локали

В.10 TranslationType

Элемент TranslationType содержит дополнительные переводы для Localized Texts, используемых в значении переменной. Поля элемента TranslationType приведены в таблице В.10. Если в наличии несколько аргументов, существует элемент перевода для каждого аргумента.

Тип может иметь две формы в зависимости от того, является ли значение LocalizedText или структурой, содержащей LocalizedTexts. Если это LocalizedText, он содержит простой список переводов. Если это структура, она содержит список полей, каждое из которых содержит список переводов. Каждое поле идентифицируется именем, уникальным в пределах структуры. Сопоставление имени и структуры требует понимания структуры кодировки. Если поле закодировано как LocalizedText с помощью UA XML, то имя представляет собой неполный путь к элементу XML, где имена в пути разделяются символом «/». Например, структура с вложенной структурой, содержащей LocalizedText, может иметь путь типа Server/ApplicationName.

В следующем примере показан алгоритм перевода поля «Описание» в аргументе. Структура представлена в схеме XML:

```
<Value>
<ListOfExtensionObject xmlns="http://opcfoundation.org/UA/2008/02/Types.xsd">
<ExtensionObject>
<TypeId>
<Identifier>i=297</Identifier>
```

```
</TypeId>
<Body>
<Argument>
<Name>ConfigData</Name>
<DataType>
<Identifier>i=15</Identifier>
</DataType>
<ValueRank>-1</ValueRank>
<ArrayDimensions />
<Description>
<Text>[English Translation for Description]</Text>
</Description>
</Argument>
</Body>
</ExtensionObject>
</ListOfExtensionObject>
</Value>
<Translation>
<Field Name="Description">
<Text Locale="de-DE">[German Translation for Description]</Text>
<Text Locale="fr-FR">[French Translation for Description]</Text>
</Field>
</Translation>
```

Для каждого структурного аргумента должна быть реализована своя локаль.

Таблица В.10 — Тип перевода

Элемент	Тип	Описание типа
Text	LocalizedText []	Массив переводов значения. Он появляется только в том случае, если значением является LocalizedText или массив LocalizedText
Field	StructureTranslationType []	Массив структурных полей, имеющих переводы. Оно появляется только в том случае, если значением является структура или массив структур
Name	String	Имя поля. Это однозначно идентифицирует поле внутри структуры. Точное отображение зависит от кодировки структуры
Text	LocalizedText []	Массив переводов поля структуры

В.11 Элемент UADataType

Элемент UADataType — это подтип UAType, определенный в В.6. Он определяет узел типа данных. Поля типа UADataType приведены в таблице В.11.

Таблица В.11 — UADataType

Элемент	Тип	Описание полей подтипа
Поля из UANode type описаны в В.3		
Definition	DataTypeDefinition	Абстрактное определение типа данных, которое может быть использовано инструментами проектирования для создания кода, который может сериализовать тип данных в XML и/или двоичных формах. Он не отображается в AddressSpace. Это применяется только для определения подтипов типов данных структуры или перечисления

В.12 Элемент DataTypeDefinition

Элемент DataTypeDefinition определяет абстрактное представление UADataType, которое может быть использовано инструментами проектирования для автоматического создания кода сериализации. Поля в элементе DataTypeDefinition приведены в таблице В.12.

Таблица В.12 — Определение типа данных

Элемент	Тип	Описание полей типа
Name	QualifiedName	Уникальное имя типа данных должно совпадать с именем браузера или содержащим его типом данных
SymbolicName	String	Символическое имя типа данных, которое можно применять в качестве имени класса/структуры в автоматически создаваемом коде. Его следует указывать только в том случае, если имя не может быть использовано для этой цели. Разрешены только буквы, цифры или знак подчеркивания («_»), причем первым символом должна быть буква. Это поле указывают только для вложенных определений типа данных. В противном случае используют символическое имя узла типа данных
BaseType	QualifiedName	Не используется. Сохраняется в схеме для обратной совместимости
IsUnion	Boolean	Этот флаг указывает, представляет ли тип данных объединение. Только одно из полей, определенных для типа данных, кодируется в значение. Это поле является необязательным. Значение по умолчанию неверно. Если это значение истинное, первое поле является значением переключателя
IsOptionSet	Boolean	Этот флаг указывает, что тип данных определяет свойство OptionSetValues. Данное поле является необязательным. Значение по умолчанию ложь
Fields	DataTypeField []	Список полей, составляющих тип данных. Это определение предполагает, что структура имеет последовательную компоновку. Для перечислений поля представляют собой просто список значений. Данный список не включает поля, унаследованные от базового типа данных

В.13 Элемент DataTypeField

Элемент DataTypeField определяет абстрактное представление поля внутри UADataType, которое может быть использовано инструментами проектирования для автоматического создания кода сериализации. Поля Типа DataTypeField приведены в таблице В.13.

Таблица В.13 — Поля типа данных

Элемент	Тип	Описание полей типа
Name	String	Имя поля, уникальное в элементе DataTypeDefinition
SymbolicName	String	Символическое имя поля, которое можно применять в автоматически создаваемом коде. Его следует указывать только в том случае, если имя не может быть использовано для этой цели. Разрешены только буквы, цифры или знак подчеркивания («_»)
DisplayName	LocalizedText []	Имя поля, отображаемое с использованием различных языков
DataType	NodeId	NodeId DataType для поля. NodeId может ссылаться на другой узел со своим собственным DataTypeDefinition. Данное поле не указывается для подтипов перечисления
ValueRank	Int32	Ранг значения для поля. Это должен быть скаляр (—1) или массив фиксированного ранга (≥1). Для подтипов перечисления это поле не указывается

Окончание таблицы В.13

Элемент	Тип	Описание полей типа
ArrayDimensions	String	Максимальная длина массива. Это поле представляет собой список целочисленных значений без знака, разделенных запятыми. Список содержит количество элементов, равное ValueRank. Значение равно 0, если максимум для измерения неизвестен. Это поле не указывается, если ValueRank ≤ 0. Это поле не указывается для подтипов перечисления или для типов данных со свойством OptionSetValues
MaxStringLength	UInt32	Максимальная длина значения String или ByteString. Если неизвестно, значение равно 0. Значение равно 0, если DataType не является String или ByteString. Если ValueRank > 0, максимум применяется к каждому элементу массива. Для подтипов перечисления или для типов данных со свойством OptionSetValues это поле не указывается
Description	LocalizedText []	Описание поля в нескольких локалях
Value	Int32	Значение, связанное с полем. Поле указывается только для подтипов типов данных Enumeration и OptionSet. Для OptionSets значением является номер бита, связанного с полем
IsOptional	Boolean	Поле указывает, что поле типа данных в структуре является необязательным. Значение по умолчанию ложь. Это поле не указано для подтипов Enumeration и Union

B.14 Variant

Тип Variant определяет значение для узла Variable или VariableType. Этот тип аналогичен типу, определенному в 5.3.1.17. В результате функции, применяемые для сериализации вариантов во время вызовов службы, можно использовать для сериализации варианта в этом синтаксисе файла.

Варианты могут содержать элементы NodeIds, ExpandedNodeIds и QualifiedNames, которые должны быть изменены таким образом, чтобы NamespaceIndexes и ServerIndexes ссылались на NamespaceUri и ServerUri (см. таблицы в UANodeSet).

Варианты также могут содержать объекты ExtensionObject, которые содержат EncodingId и структуру с полями, которые могут быть NodeIds, ExpandedNodeIds или QualifiedNames. NamespaceIndexes и ServerIndexes в этих полях также должны ссылаться на таблицы, приведенные в UANodeSet.

B.15 Example

Пример представления комплекса некоторых систем с UA (UANodeSet). В данном комплексе определены узлы для информационной модели с URI, представленной на адресе http://sample.com/Instances. Данные об узлах для базовой информационной модели OPC UA и информационной модели с URI приведены по адресу: http://sample.com/Types.

Пространства имен XML, объявленные ранее, включают URI для пространств имен, упомянутых в UANodeSet, поскольку данный комплекс построен на архитектуре данных с несколькими уровнями. Такие комплексы при отсутствии в них многоуровневой архитектуры данных не будут иметь этих объявлений.

<UANodeSet xmlns:s1="http://sample.com/Instances" xmlns:s0="http://sample.com/Types"
xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd">

В таблицу NamespaceUris включены все пространства имен, упомянутых в UANodeSet, за исключением базовой информационной модели OPC UA. NamespaceIndex, равный 1, относится к URI по адресу: http://sample.com/Instances.

<NamespaceUris>
<Uri>http://sample.com/Instances</Uri>
<Uri>http://sample.com/Types</Uri>
</NamespaceUris>

Приведенная таблица псевдонимов предназначена для удобства чтения. Правила относительно того, что включено, отсутствуют. Полезное руководство может включать стандартные ReferenceTypes и DataTypes при наличии ссылки в представлении UANodeSet.

```
<Aliases>
<Alias Alias="HasComponent">i=47</Alias>
<Alias Alias="HasProperty">i=46</Alias>
<Alias Alias="HasSubtype">i=45</Alias>
<Alias Alias="HasTypeDefinition">i=40</Alias>
</Aliases>
```

BicycleType — это узел типа данных, который наследуется от типа данных, определенного в другой информационной модели (*ns* = 2; *i* = 314). Предполагается, что любое приложение импортирует этот файл при наличии информации указанной информационной модели, представляющей UANodeSet. Сервер может сопоставить ссылки с другим сервером OPC UA, добавив ServerIndex в TargetNode NodeIds. Структура DataType определена элементом Definition. Эта информация может быть использована генераторами кода для автоматического создания сериализаторов для DataType.

```
<UADataType NodeId="ns=1;i=365" BrowseName="1:BicycleType">
<DisplayName>BicycleType</DisplayName>
<References>
<Reference ReferenceType="HasSubtype" IsForward="false">ns=2;i=314</Reference>
</References>
<Definition Name="BicycleType">
<Field Name="NoOfGears" DataType="UInt32"/>
<Field Name="ManufacterName" DataType="QualifiedName"/>
</Definition>
</UADataType>
```

Данный узел является экземпляром Object TypeDefinition Node, определенного в другой InformationModel (*ns* = 2; *i* = 341). В рассматриваемом комплексе есть единственное Property, которое объявлено в описании UANodeSet.

```
<UAObject NodeId="ns=1;i=375" BrowseName="1:DriverOfTheMonth" ParentNodeId="ns=1;i=281">
<DisplayName>DriverOfTheMonth</DisplayName>
<References>
<Reference ReferenceType="HasProperty">ns=1;i=376</Reference>
<Reference ReferenceType="HasTypeDefinition">ns=2;i=341</Reference>
<Reference ReferenceType="HasComponent" IsForward="false">ns=1;i=281</Reference>
</References>
</UAObject>
```

Данный узел является экземпляром узла определения типа переменной, определенного в базовой информационной модели OPC UA (*i* = 68). DataType является базовым типом для DataType BicycleType. AccessLevels объявляет переменную как доступную для чтения и записи. ParentNodeId указывает, что этот узел тесно связан с родительским узлом (DriverOfTheMonth) и будет удален.

```
<UAVariable NodeId="ns=1;i=376" BrowseName="2:PrimaryVehicle" ParentNodeId="ns=1;i=375"
DataType="ns=2;i=314" AccessLevel="3" UserAccessLevel="3">
<DisplayName>PrimaryVehicle</DisplayName>
<References>
<Reference ReferenceType="HasTypeDefinition">i=68</Reference>
<Reference ReferenceType="HasProperty" IsForward="false">ns=1;i=375</Reference>
</References>
```

Указанное значение является экземпляром типа данных BicycleType. Он заключен в ExtensionObject, который фиксирует, что значение сериализуется с использованием кодировки DataTypeEncoding XML по умолчанию для DataType. Значение можно сериализовать и путем двоичного кодирования типа данных по умолчанию, но в результате представление комплекса невозможно будет редактировать вручную. Независимо от того, какой DataTypeEncoding применяется, NamespaceIndex, используемый в поле ManufactureName, ссылается на таблицу NamespaceUris в UANodeSet. Описание комплекса загружается приложением только при условии, если данное приложение отвечает за изменение любого значения, которое оно должно иметь.

```
<Value>
<ExtensionObject xmlns="http://opcfoundation.org/UA/2008/02/Types.xsd">
<TypeId>
<Identifier>ns=1;i=366</Identifier>
</TypeId>
<Body>
<s1:BicycleType>
```

<s0:Make>Trek</s0:Make>
<s0:Model>Compact</s0:Model>
<s1:NoOfGears>10</s1:NoOfGears>
<s1:ManufactureName>
<uax:NamespaceIndex>1</uax:NamespaceIndex>
<uax:Name>Hello</uax:Name>
</s1:ManufactureName>
</s1:BicycleType>
</Body>
</ExtensionObject>
</Value>
</UAVariable>

Далее приведены узлы DataTypeEncoding для типа данных BicycleType.

<UAObject NodeId="ns=1;i=366" BrowseName="Default XML">
<DisplayName>Default XML</DisplayName>
<References>
<Reference ReferenceType="HasEncoding" IsForward="false">ns=1;i=365</Reference>
<Reference ReferenceType="HasDescription">ns=1;i=367</Reference>
<Reference ReferenceType="HasTypeDefinition">i=76</Reference>
</References>
</UAObject>
<UAObject NodeId="ns=1;i=370" BrowseName="Default Binary">
<DisplayName>Default Binary</DisplayName>
<References>
<Reference ReferenceType="HasEncoding" IsForward="false">ns=1;i=365</Reference>
<Reference ReferenceType="HasDescription">ns=1;i=371</Reference>
<Reference ReferenceType="HasTypeDefinition">i=76</Reference>
</References>
</UAObject>

Это узел DataTypeDescription для XML-кодирования типа данных по умолчанию для типа данных BicycleType. Value — один из встроенных типов.

<UAVariable NodeId="ns=1;i=367" BrowseName="1:BicycleType" DataType="String">
<DisplayName>BicycleType</DisplayName>
<References>
<Reference ReferenceType="HasTypeDefinition">i=69</Reference>
<Reference ReferenceType="HasComponent" IsForward="false">ns=1;i=341</Reference>
</References>
<Value>
<uax:String>//xs:element[@name='BicycleType']</uax:String>
</Value>
</UAVariable>

Далее показан узел DataTypeDictionary для DataTypeDescription. Запись схемы XML — это документ UTF-8, хранящийся как значение xs:base64Binary, что позволяет клиентам читать схему для всех типов данных, принадлежащих DataTypeDictionary. Значение узла DataTypeDescription для каждого типа данных содержит запрос XPath, который находит правильное определение внутри документа схемы.

<UAVariable NodeId="ns=1;i=341" BrowseName="1:Quickstarts.DataTypes.Instances" DataType="ByteString">
<DisplayName>Quickstarts.DataTypes.Instances</DisplayName>
<References>
<Reference ReferenceType="HasProperty">ns=1;i=343</Reference>
<Reference ReferenceType="HasComponent">ns=1;i=367</Reference>
<Reference ReferenceType="HasComponent" IsForward="false">i=92</Reference>
<Reference ReferenceType="HasTypeDefinition">i=72</Reference>
</References>
<Value>
<uax:ByteString>PHhz...W1hPg==</uax:ByteString>
</Value>
</UAVariable>>

B.16 UANodeSetChanges

UANodeSetChanges — корень документа, содержащий набор изменений в AddressSpace. Предполагается, что один файл будет содержать либо элемент UANodeSet, либо элемент UANodeSetChanges в корне, и предоставлять

список узлов/ссылок для добавления или список узлов/ссылок для удаления. Структура UANodeSetChangesStatus, определенная в В.22, создается, когда документ UANodeSetChanges применяется к AddressSpace. Элементы типа приведены в таблице В.14.

Таблица В.14 — Корень набора изменений в AddressSpace

Элемент	Тип	Описание UANodeSetChanges
NamespaceUris	UriTable	Описано в таблице В.1
ServerUris	UriTable	
Models	ModelTableEntry []	
Aliases	AliasTable	
Extensions	xs:any	Описано в таблице В.1
LastModified	DateTime	
NodesToAdd	NodesToAdd	Список новых узлов для добавления в AddressSpace
ReferencesToAdd	ReferencesToChange	Список новых ссылок для добавления в AddressSpace
NodesToDelete	NodesToDelete	Список узлов для удаления из AddressSpace
ReferencesToDelete	ReferencesToChange	Список ссылок для удаления из AddressSpace

Элемент Models указывает версию одной или нескольких моделей, которые создаст файл UANodeSetChanges при его применении к существующему адресному пространству. UANodeSetChanges не представляется возможным применить, если текущая версия модели в адресном пространстве выше. Подэлемент RequiredModels (см. таблицу В.1) определяет версии. При проверке зависимостей версия модели в существующем адресном пространстве должна точно соответствовать требуемой версии.

Если файл UANodeSetChanges изменяет типы и в AddressSpace существуют действующие экземпляры типов, то сервер должен автоматически изменить экземпляры, чтобы они соответствовали новому типу, или сгенерировать ошибку.

Файл UANodeSetChanges обрабатывается как одна операция. Это позволяет единовременно указать адреса для обязательного удаления узлов и необходимой замены ссылок.

B.17 NodesToAdd

Тип NodesToAdd указывает список узлов, которые нужно добавить в AddressSpace. Структура этих узлов определяется типом UANodeSet в таблице В.1. Элементы типа приведены в таблице В.15.

Таблица В.15 — NodesToAdd

Элемент	Тип	Описание элементов NodesToAdd
<choice>	UAObject, UAVariable, UAMethod UAView, UAObjectType, UAVariableType, UADataType, UAReferenceType	Узлы, добавляемые в AddressSpace

При добавлении узлов ссылки можно указать как часть определения узла или как отдельный элемент ReferencesToAdd. Следует иметь в виду, что ссылки на узлы, которые могут существовать, разрешены. Другими словами, узел не отклоняется потому, что он имеет ссылку на неизвестный узел. Обратные ссылки добавляются автоматически, если процессор считает это целесообразным.

B.18 ReferencesToChange

ReferencesToChange указывает список ссылок, которые необходимо добавить или удалить из AddressSpace. Элементы типа приведены в таблице В.16.

Таблица В.16 — Ссылки на изменение

Элемент	Тип	Описание элементов ReferencesToChange
Reference	ReferenceToChange	Ссылка для добавления в AddressSpace

B.19 ReferenceToChange

Тип ReferenceToChange указывает одну ссылку, которую нужно добавить или удалить из AddressSpace. Элементы типа приведены в таблице В.17.

Таблица В.17 — Ссылки на изменение

Элемент	Тип	Описание полей ReferencesToChange
Source	NodeId	Идентификатор исходного узла ссылки
ReferenceType	NodeId	Идентификатор типа ссылки
IsForward	Boolean	Истина, если ссылка является прямой ссылкой
Target	NodeId	Идентификатор целевого узла ссылки

Ссылки на узлы, которые могут существовать, разрешены. Другими словами, ссылка не отклоняется потому, что целью является неизвестный узел.

Источник ссылки должен существовать в AddressSpace или в UANodeSetChanges, когда представление комплекса находится в обработке.

Обратные ссылки добавляются, когда процессор считает это целесообразным.

B.20 NodesToDelete

Тип NodesToDelete указывает список узлов, которые необходимо удалить из AddressSpace. Элементы типа приведены в таблице В.18.

Таблица В.18 — NodesToDelete

Элемент	Тип	Описание узлов
Node	NodeToDelete	Узел, который нужно удалить из AddressSpace

B.21 NodeToDelete

Тип NodeToDelete указывает узел, который необходимо удалить из AddressSpace. Элементы типа приведены в таблице В.19.

Таблица В.19 — Ссылки на изменение

Элемент	Тип	Описание узлов
Node	NodeId	Идентификатор узла, который необходимо удалить
DeleteReverseReferences	Boolean	Если истина, то ссылки на узел также удаляются

B.22 UANodeSetChangesStatus

UANodeSetChangesStatus — это корень документа, который создается при обработке документа UANodeSetChanges. Элементы типа приведены в таблице В.20.

Таблица B.20 — UANodeSetChangesStatus

Элемент	Тип	Описание элементов типа
NamespaceUris	UriTable	См. в таблице B.1
ServerUris	UriTable	
Aliases	AliasTable	
Extensions	xs:any	
Version	String	
LastModified	DateTime	См. в таблице B.1
TransactionId	String	Глобальный уникальный идентификатор из исходного документа UANodeSetChanges
NodesToAdd	NodeSetStatusList	Список результатов для NodesToAdd, указанного в исходном документе. Список пуст, если все элементы обработаны успешно
ReferencesToAdd	NodeSetStatusList	Список результатов для ReferencesToAdd, указанного в исходном документе. Список пуст, если все элементы обработаны успешно
NodesToDelete	NodeSetStatusList	Список результатов для NodesToDelete, указанного в исходном документе. Список пуст, если все элементы обработаны успешно
ReferencesToDelete	NodeSetStatusList	Список результатов для ReferencesToDelete, указанного в исходном документе. Список пуст, если все элементы обработаны успешно

B.23 NodeSetStatusList

Тип NodeSetStatusList определяет список результатов, полученных при применении документа UANodeSetChanges к AddressSpace. Если ошибок не произошло, этот список пуст. Если возникает одна или несколько ошибок, то этот список содержит по одному элементу для каждой операции, указанной в исходном документе. Элементы типа приведены в таблице B.21.

Таблица B.21 — NodeSetStatusList

Элемент	Тип	Описание элементов типа
Result	NodeSetStatus	Результат операции

B.24 NodeSetStatus

Тип NodeSetStatus указывает один результат, полученный при применении операции, указанной в документе UANodeSetChanges, к AddressSpace.

Элементы типа приведены в таблице B.22.

Таблица B.22 — NodeSetStatus

Элемент	Тип	Описание элементов типа
Code	StatusCode	Результат операции. Возможные коды состояния определены согласно ГОСТ Р 71809
Details	String	Строка, предоставляющая информацию, которая не передается с помощью StatusCode является нечитаемой для StatusCode

Библиография

[1]	МЭК 62541-7	«Унифицированная архитектура OPC. Часть 7. Профили»
[2]	RFC 7159	«Формат обмена данными JavaScript Object Notation (JSON)»
[3]	ИСО 8601-1:2019	«Дата и время. Представление для обмена информацией. Часть 1. Основные правила»
[4]	RFC 3280	«Сертификат инфраструктуры открытых ключей Internet X.509 и список отзыва сертификатов (CRL) Профиль»
[5]	RFC 1738	«Унифицированные средства поиска ресурсов (URL)»
[6]	RFC 8141	«Унифицированные имена ресурсов (URNs)»
[7]	RFC 6749	«Платформа авторизации OAuth 2.0»
[8]	RFC 7523	«Профиль JSON Web Token (JWT) для аутентификации и авторизации клиента OAuth 2.0»
[9]	IEC/TR 62541-2:2020	«Унифицированная архитектура OPC. Часть 2. Модель безопасности»
[10]	ИСО/МЭК 7498-1 (Рекомендация МСЭ-Т X.200)	«Информационные технологии. Взаимосвязь открытых систем. Базовая эталонная модель: Базовая Модель. АдресацияWS»
[11]	RFC 6455	«Протокол WebSocket»
[12]	RFC 5246	«Протокол безопасности транспортного уровня (TLS)»
[13]	RFC 4514	«Легкий протокол доступа к каталогам (LDAP): строковое представление отличительных имен»
[14]	RFC 6960	«Протокол состояния онлайн-сертификата (OCSP)»

УДК 621.3.049.75:006.354

ОКС 31.180
31.190

Ключевые слова: цифровая промышленность, управление цифровой трансформацией производств, мета-модель унифицированной архитектуры, открытая распределенная система, адресное пространство

Редактор *Л.С. Зимилова*
Технический редактор *И.Е. Черепкова*
Корректор *Р.А. Ментова*
Компьютерная верстка *М.В. Малеевой*

Сдано в набор 24.12.2024. Подписано в печать 16.01.2025. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 10,23. Уч.-изд. л. 8,70.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Создано в единичном исполнении в ФГБУ «Институт стандартизации»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.
www.gostinfo.ru info@gostinfo.ru