**СЕРВЕР:**  
**Simplex** - это класс веб-службы, который наследуется от WebService. Он предоставляет несколько методов, которые можно вызывать удаленно.

Атрибут [WebService] используется для указания описания и пространства имен веб-службы.

[ScriptService] указывает, что этот класс может быть использован из скрипта с использованием ASP.NET AJAX.

**Параметр Description атрибута WebService:**

* **Описание:** Этот параметр предназначен для предоставления краткой информации о веб-службе. Это описание может быть использовано различными инструментами, такими как генераторы документации или инструменты разработки, чтобы предоставить пользователям понятное описание функциональности веб-службы.

**Параметр Description атрибута WebMethod:**

* **Описание:** Этот параметр предназначен для предоставления описания конкретного метода веб-службы. Он может использоваться, например, для пояснения того, что делает метод, какие параметры принимает и какой результат возвращает.

**Параметр MessageName атрибута WebMethod:**

* **Описание:** Этот параметр позволяет задать имя, которое будет использовано для метода в сообщениях SOAP (Simple Object Access Protocol), когда клиент взаимодействует с веб-службой. Это имя может быть полезным в тех случаях, когда имя метода в коде отличается от имени, используемого в протоколе взаимодействия.

**12 пункт методы:**

1. **Собственное пространство имен:**
   * Обычно, в WSDL-документе, собственное пространство имен указывается в разделе **wsdl:definitions** как атрибут **targetNamespace**.
2. **Описание типов данных для всех методов сервиса:**
   * В разделе **wsdl:types** вы можете найти описания типов данных, используемых в веб-службе. Эти описания будут включать в себя типы параметров методов, типы возвращаемых значений и другие.
3. **Описание входных и выходных сообщений всех методов сервиса:**
   * В разделе **wsdl:message** будут описаны сообщения, используемые в методах веб-службы.
4. **Описание методов сервиса:**
   * Раздел **wsdl:portType** содержит описания всех методов веб-службы.
5. **Описание протоколов обмена данных:**
   * В разделе **wsdl:binding** содержится информация о протоколе обмена данных (например, SOAP). Пример:
6. **Описание сервиса:**
   * Раздел **wsdl:service** содержит конечную точку (endpoint) веб-службы, к которой клиенты могут обращаться. Пример:

**13 пункт методы:**

1. **Envelope (Оболочка):** Оболочка содержит всю информацию SOAP-сообщения и представляет начало и конец сообщения.
2. **Header (Заголовок):** Необязательный элемент, который содержит метаданные, необходимые для обработки сообщения.
3. **Body (Тело):** Содержит основную информацию сообщения, такую как вызов метода и его параметры.
4. **Fault (Ошибка):** Элемент, используемый для передачи информации об ошибках, если таковые возникли.

**<soap:Envelope**

**xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"**

**soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">**

**<soap:Header>**

**<!-- Заголовочная информация -->**

**</soap:Header>**

**<soap:Body>**

**<Add xmlns="http://BVA/">**

**<x>5</x>**

**<y>10</y>**

**</Add>**

**</soap:Body>**

**</soap:Envelope>**

**15 пункт методы:**

1. **Создание Proxy-классов:**
   * WSDL (Web Services Description Language) предоставляет метаданные о веб-службе, включая информацию о методах, типах данных и протоколе обмена данными (например, SOAP). Сгенерированный proxy-класс представляет клиентскую сторону веб-службы в виде классов, которые можно использовать в коде клиента для вызова методов веб-службы.
2. **Как использовать wsdl.exe для генерации Proxy-классов:**
   * Откройте командную строку (Command Prompt) и перейдите в каталог, в который вы хотите сохранить сгенерированный код.
   * Выполните команду **wsdl.exe** с URL-адресом WSDL-документа вашей веб-службы.
   * После выполнения команды будет создан файл с расширением **.cs**, который содержит сгенерированный код proxy-класса.
3. **Использование сгенерированного Proxy-класса:**
   * После генерации proxy-класса вы можете включить его в свой проект и использовать для взаимодействия с веб-службой.
   * Создайте экземпляр proxy-класса и вызывайте его методы, как если бы это были локальные методы:
4. **Зачем это нужно:**
   * Создание proxy-класса упрощает взаимодействие с веб-службой, так как клиентскому приложению не нужно руководствоваться структурой SOAP-запросов и парсить SOAP-ответы. Proxy-класс обрабатывает эти детали за кулисами.
   * Программистам необходимо использовать proxy-классы для вызова методов веб-службы, что упрощает процесс взаимодействия с удаленными сервисами.

**Использование прокси-класса в клиентском коде:** Клиентский код может использовать созданный прокси-класс так, как если бы он работал с локальным объектом. Этот класс обрабатывает детали взаимодействия с веб-службой, включая формирование SOAP-запросов, отправку запросов по сети и обработку SOAP-ответов.

**Вызов методов веб-службы:** Клиентский код вызывает методы прокси-класса, которые в свою очередь генерируют соответствующие SOAP-запросы и обрабатывают SOAP-ответы от веб-службы.

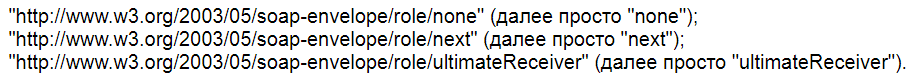
**SOAP** (от англ. Simple Object Access Protocol — простой протокол доступа к объектам) — протокол обмена структурированными сообщениями в распределённой вычислительной среде. Первоначально SOAP предназначался в основном для реализации удалённого вызова процедур (RPC). Сейчас протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур.

SOAP может использоваться с любым протоколом прикладного уровня: SMTP, FTP, HTTP, HTTPS и др. Однако его взаимодействие с каждым из этих протоколов имеет свои особенности, которые должны быть определены отдельно. Чаще всего SOAP используется поверх HTTP.

**WSDL** (англ. Web Services Description Language) — язык описания веб-сервисов и доступа к ним, основанный на языке XML.

**Роль** – набор правил, определяющих поведение узла, задается атрибутом env:role.

**SOAP:** тристандартные роли для узла (нет роли, промежуточный узел, конечный получатель), роль может быть пользовательской (задаваться собственным URI).



**Роль** может быть указана во всех блоках и указывает, кому предназначена информация блока. Обычно, обработанный блок извлекается из конверта, если не указано другого.

**Узел –** компонент SOA архитектуры, который может получать и отправлять сообщения.

**Пространства имён:** <https://www.informit.com/articles/article.aspx?p=169106&seqNum=2>

*Пространства* имен *XML* играют важную роль в сообщениях *SOAP*.  *SOAP* сообщение может включать в себя несколько различных *XML* - элементов в Header и Body, и, чтобы избежать столкновения имен, каждый из этих элементов должно быть идентифицировано уникальным пространством имен.

Использование пространств имен *XML* делает *SOAP* гибким и расширяемым протоколом.

Envelope, определяет пространство имен стандартных элементов *SOAP -*Envelope, Header и Body.

Каждый блок заголовка в элементе Header должен иметь собственное пространство имен. Это особенно важно, поскольку пространства имен помогают приложениям *SOAP* идентифицировать блоки заголовков и обрабатывать их отдельно.

Реальная сила пространств имен *XML* выходит за рамки простого предотвращения конфликтов имен, а также правильного управления версиями и обработкой. Использование полных имен для *SOAP* и данных приложения сообщает получателю *SOAP,* как обрабатывать сообщение, и какие схемы *XML* применять для проверки его содержимого. Например, различия в конкретной версии блока заголовка могут повлиять на то, как получатель обрабатывает сообщения, поэтому определение версии блока заголовка по его пространству имен позволяет получателю переключать модели обработки или отклонять сообщения, если он не поддерживает указанная версия.

**Ключевые аспекты:**

При работе с SOAP стоит учитывать следующие моменты

* Элемент Header является необязательным.
* Этот элемент используется для добавления новых возможностей и функциональности.
* Элемент header можно использовать несколько раз в файле.
* Header является первым к прочтению файлом внутри элемента envelope.

**Атрибуты элемента Header**

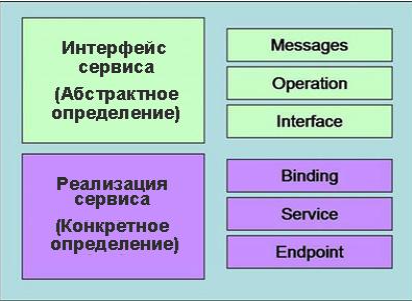
Элемент Header может иметь следующие два атрибута:

* **Actor** Протокол SOAP определяет расположение сообщения, как список сервисов SOAP. Каждый из этих промежуточных сервисов может выполнять определённую работу, а затем передавать сообщение следующему сервису по цепочке. С помощью атрибута Actor, клиент может определить получателя элемента header.
* **MustUnderstand** Этот атрибут показывает, является ли элемент обязательным или опциональным (**true –**обязательный, **false**– нет). Если элемент обязательный, то получатель уведомляется об этом и обрабатывает элемент **header** соответственно указанной семантике, либо возвращает ошибку.

**Прокси-класс с помощью wsdl.exe**

* Утилита wsdl.exe является соответствующей для asmx техникой потребления SOAP веб-сервисов. По wsdl файлу или ссылке она генерирует прокси-класс – специальной класс, максимально упрощающий обращение к данному веб-сервису. Разумеется, не важно на какой технологии реализован сам веб-сервис, это может быть что угодно — ASMX, WCF, JAX-WS или NuSOAP. Кстати, у WCF аналогичная утилита называется SvcUtil.exe.
* Утилита расположена в папке C:\Program Files (x86)\Microsoft SDKs\Windows, более того, она там представлена в разных версиях, в зависимости от версии .net, разрядности, версии windows и visual studio.
* Как известно, wsdl описание веб-сервиса в технологии ASMX генерируется автоматически. Однако иногда возникает обратная задача: по данному wsdl файлу разработать соответствующий ему веб-сервис. Решается она с помощью той же утилиты wsdl.exe. Она может создать необходимый скелет из классов и вам останется только реализовать программную логику веб-методов.

**Концептуальная модель WSDL 2.0**

Описание Web-сервиса можно разделить на две части. В абстрактной части описания Web-сервис описывается в языке WSDL с помощью системы типов, обычно W3C XML-схемы, в терминах сообщений, которые этот сервис отправляет и получает. Шаблоны обмена сообщениями определяют последовательность и количество сообщений. Элемент **operation** связывает шаблоны обмена сообщениями с одним или несколькими сообщениями. Элемент **interface** группирует операции (элементы **operation**) независимо от транспорта и способа доставки.

В конкретной части описания элементы **binding** задают транспорт и формат доставки для интерфейсов (элементов **interface**). Элемент сервиса (элемента **service**) **endpoint** связывает сетевой адрес в соответствие со связыванием (элементом **binding**). Наконец, элемент **service** группирует конечные точки (элементы **endpoint**), которые реализуют общий интерфейс (элемент **interface**). На рисунке 1 изображена концептуальная модель компонентов WSDL.

**ASMX: Active Server Method Extended;** технология Microsoft для разработки web-сервисов, основанная на XML, WSDL, SOAP;

* В качестве хоста только IIS
* Модель взаимодействия только полудуплекс
* Имеет только 1 конечную точку
* Транспорт только HTTP

**Веб-сервис** — это веб-приложение, предоставляющее открытый (значит опубликованный) интерфейс, пригодный для использования другими приложениями и интерфейсами.

**Разработка ASMX сервиса:**

1. В Visual Studio добавить к проекту ASMX-службу.
2. В WebService(): Добавить пространство имен(Namespace) и описание(Description).
3. В WebMethod(): описание методов(Description), описание возвращаемого результата, добавить имя сообщения в WSDL(MessageName).
4. Добавить proxy-класс на клиент.