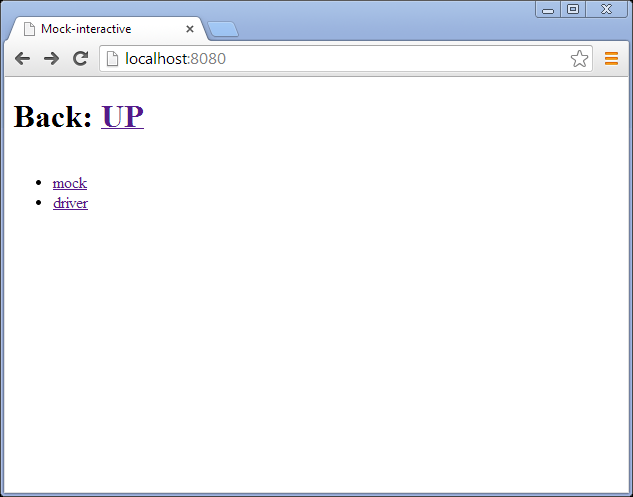
# Инструкция по работе с приложением

В приложении реализованы две основных функции:

1. Веб-сервер mock-заглушек. Предоставляет доступ к endpoint’ам. Отвечает на запросы по заранее составленным шаблонам без реализации логики бизнес процессов.
2. Драйвер отправки запросов. Предоставляет эмулировать отправку сообщений по заранее составленным шаблонам без реализации логики бизнес процессов.

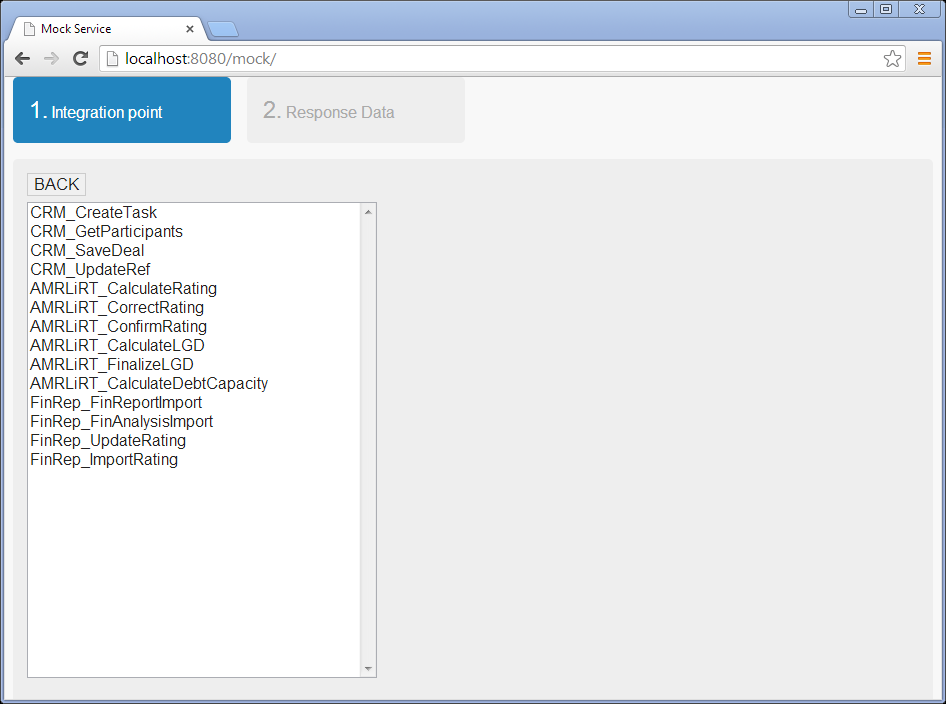
Верхний уровень веб-приложения предлагает выбор между двумя разделами – каждый отвечает за свою часть функционала приложения:



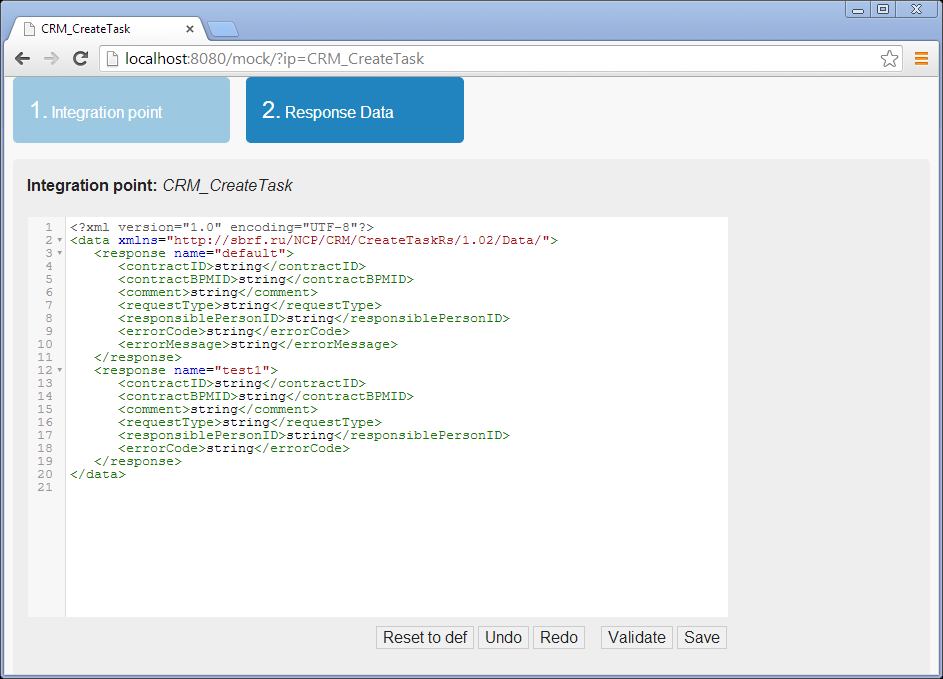
Раздел mock – за настройку шаблонов mock-заглушек.

Раздел driver – за настройку шаблонов и отправку запросов.

## Настройка mock-заглушки

При выборе раздела mock появляется страница выбора endpoint’а:  


В списке отражены все реализованные mock- серверы.

При выборе endpoint’а открывается страница редактора mock- сервера: ****

### Принцип работы mock-заглушки

При обращении к вебсервису, программа загружает файл с данными для заполнения ответа. Именно этот файл доступен для редактирования в веб-интерфейсе.

Файл представляет собой xml-контейнер элементов «response». У каждого «response» есть атрибут «name». У одного из элементов «response» атрибут «name» равен «default».

Программа способна выбирать среди элементов «response» по содержимому запроса.

У каждой точки интеграции определен элемент, по значению которого определяется имя элемента «response», из которого будет составлен ответ (Механизм генерации мок-серверов для того чтобы изменить этот элемент описан в разделе «Инструкция по доработке приложения». При изменении этого тэга веб-сервер придется передеплоить).

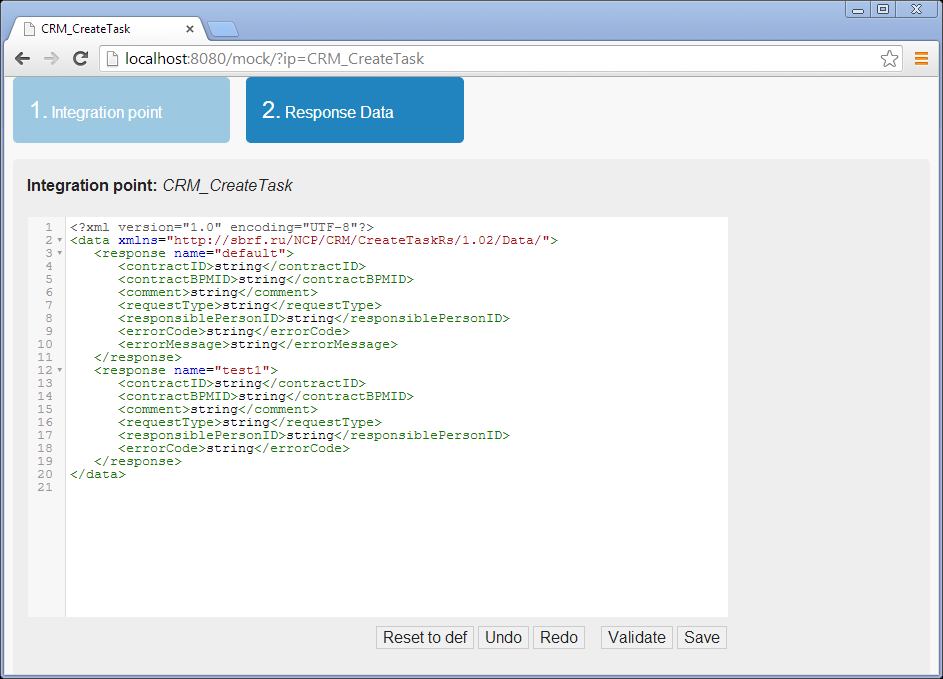
В текущей версии веб-сервера этот элемент для каждой точки интеграции определены как:

|  |  |  |
| --- | --- | --- |
| Система | Точка интеграции | Имя элементу, по которому будет определяться данные для ответа |
| CRM | CreateTask | comment |
| GetParticipants | performer |
| SaveDeal | dealType |
| UpdateRef | referenceItem |
| AMRLiRT | CalculateDebtCapacity | model |
| CalculateRating | model |
| ConfirmRating | siebelMessage |
| CorrectRating | siebelMessage |
| FinalizeLGD | type |
| CalculateLGD | comment |
| FinRep | FinAnalysisImport | dealId |
| FinReportImport | finReportType |
| ImportRating | entityType |
| UpdateRating | status |

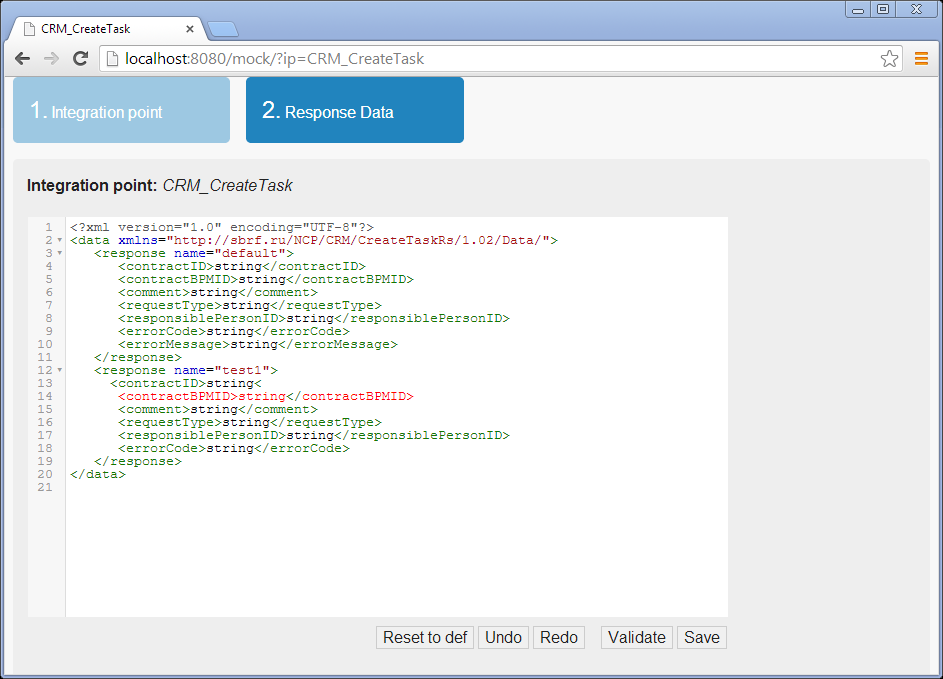
Если значение данного тэга не найдено в xml с данными, то будет использоваться default-значение.

После определения нужного имени элемента «response» из файла с данными, на его основе составляется ответное сообщение.

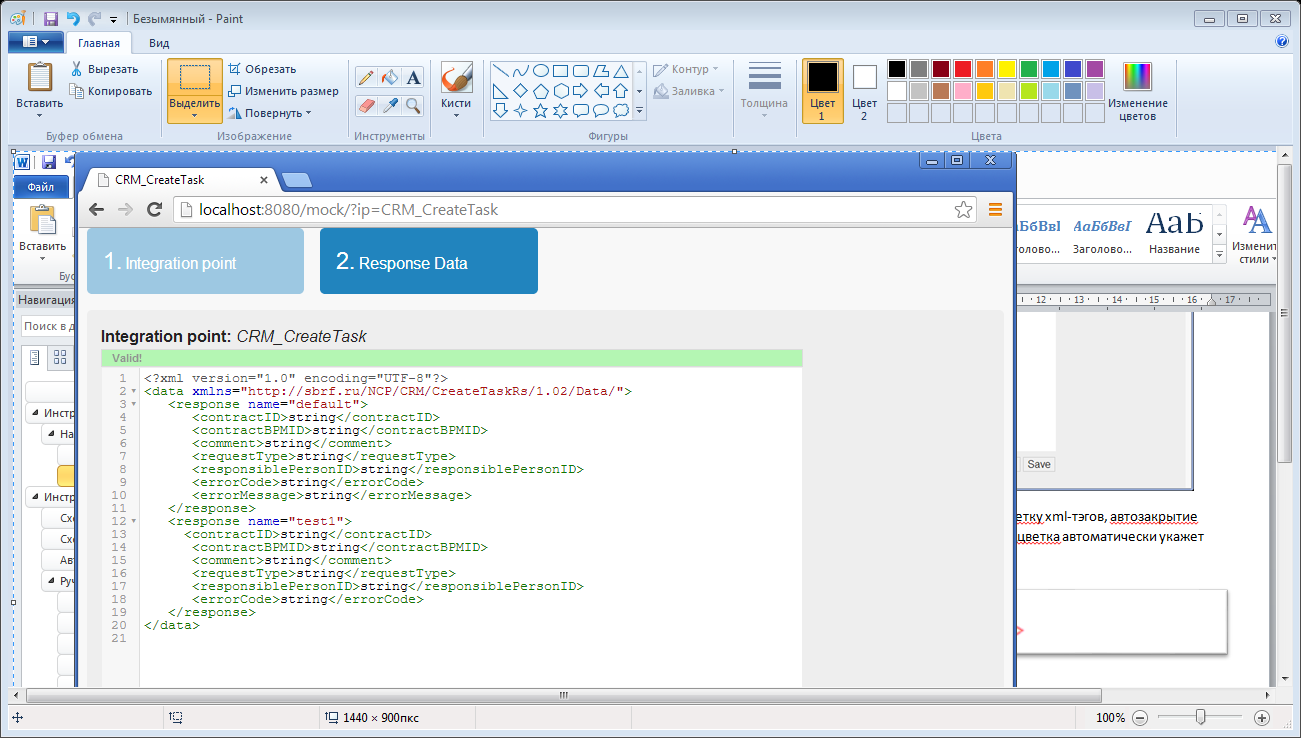
### Редактирование шаблона mock-заглушки

В окне редактора точки интеграции возможно не только просмотреть шаблон, но и редактировать его:****

Поле редактора позволяет редактировать xml, содержит автоподцветку xml-тэгов, автозакрытие открытых тэгов. Например, если нарушить xml-разметку, то автоподцветка автоматически укажет на ошибку:

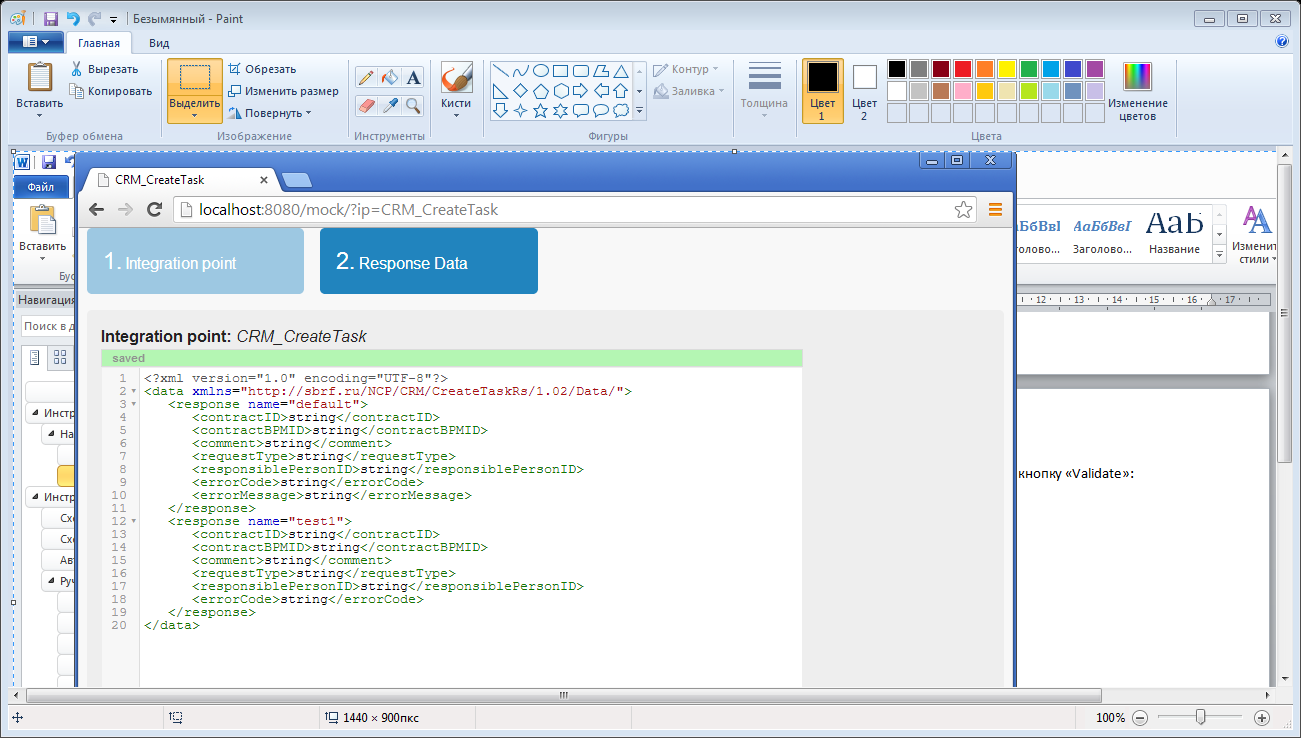


Если же в этом случае ввести символ «/», дополнив «<» до «</», то редактор автозаполнит ввод до закрывающегося тэга («</contractID>»).

После всех изменений необходимо провалидировать xml, нажав на кнопку «Validate»:

Если xml корректно и соответствует xml-схеме для шаблонов, то редактор выведет сообщение «Valid». Если же сообщение не валидно вместо этого редактор выведет указание на неправильные места в xml.

Для применения шаблона необходимо его сохранить. Для этого следует нажать кнопку «Save».

При успешном сохранении отобразится сообщение «Saved»:

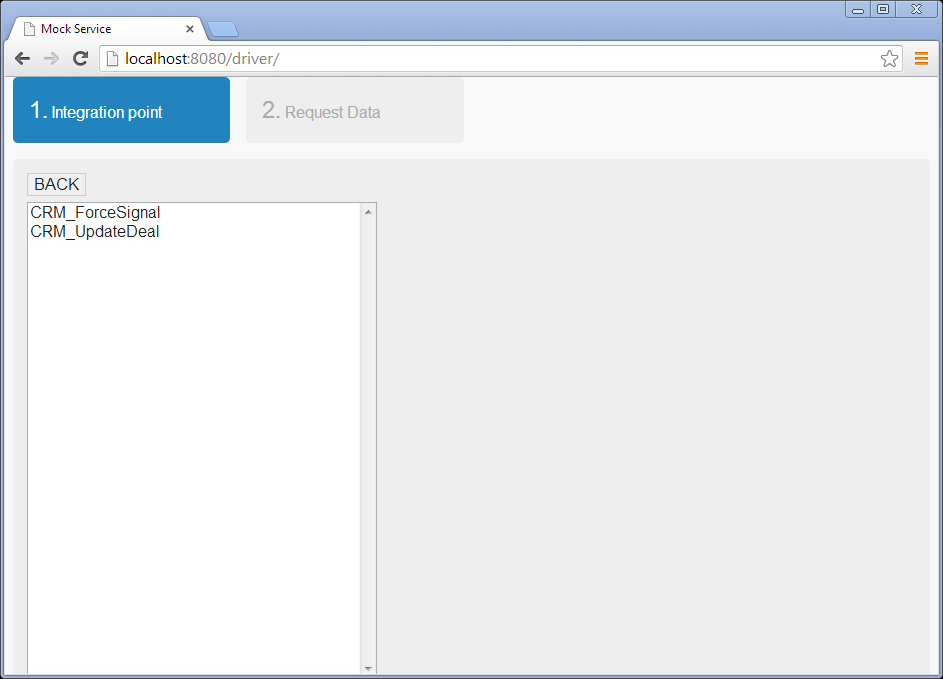
Только после этого именно измененный файл будет использоваться как шаблон для именных сообщений.

Также редактор позволяет откатывать шаблон на ранее сохраненные изменения. Для этого существуют кнопки:

* «Reset to def» – возвращает шаблон к изначальному значению.
* «Undo» – возвращает шаблон к предыдущему сохраненному значению. При этом шаблон не сохраняется – для применения необходимо нажать «Save»
* «Redu» – возвращает шаблон к следующему сохраненному значению, которое было откатано с помощью кнопки «undo». При этом шаблон не сохраняется – для применения необходимо нажать «Save»

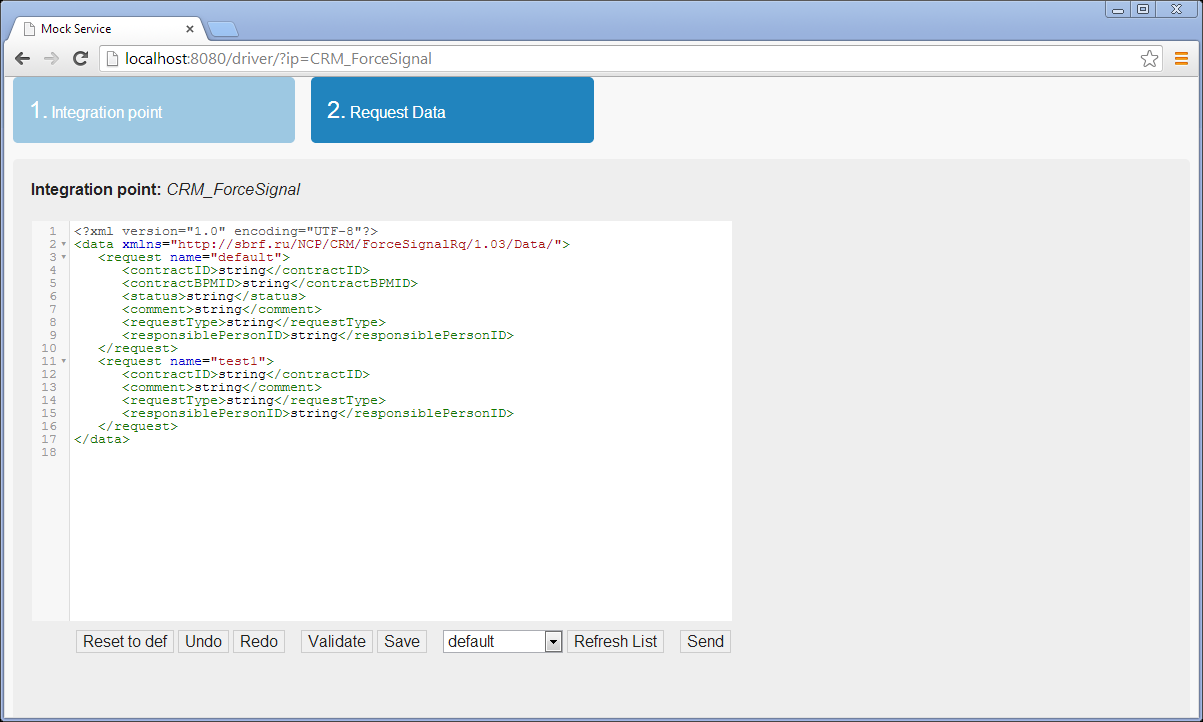
По умолчанию приложение сохраняет 4 последних сохраненных шаблона и один изначальный, самый ранний шаблон.

## Работа с драйверами

Для отправки запросов служит раздел driver:

В списке отражены все реализованные driver’ы – точки интеграции, позволяющие отправку сообщений по шаблону.

При выборе точки интеграции открывается страница редактора и отправки:



Аналогично редактированию mock-заглушек, появляется файл с шаблонами, который можно валидировать, редактировать, сохранять и восстанавливать.

В отличие, от mock-сервисов, Драйверы служат для отправки запросов. В шаблоне находятся «request» элементы с атрибутом «name». Каждый элемент «request» совпадает по структуре с запросом.

При нажатии на кнопку send приложение валидирует xml структуру шаблона, берет элемент «request» с именем из выпадающего списка под окном редактора и отправляет его.

Отправленное сообщение и ответ появятся на странице в отдельных окнах.

Выпадающий список содержит в себе все имена запросов из списка шаблонов, которые возможно отправить. При добавлении к шаблоном еще одного запроса следует нажать кнопку «Refresh List» для обновления выпадающего списка.

# Инструкция по развёртывание приложения

На данный момент существуют 2 вида сборки артефактов – для tomcat сервера и для WebSphere сервера.

### Tomcat сервер

Для создания сервера на основе tomcat 7.0 создается артефакт mock\_tomcat\_war\_exploded.

Такая сборка создана для тестовых целей – очереди а нем не крепятся к существующим MQ очередям, а драйвера замкнуты на сами себя.

### WebSphere сервер

Настройка сборки сервера для WebSphere будет описана позднее. TODO

# Инструкция по доработке приложения

## Схема эмулятора

На каждый endpoint создается структура по преобразованию запросов в ответы:

Request XSD

Request

XSLT

Response

Response XSD

Data XSD

Data xml

Обозначения на схеме:

* Request – запрос-сообщение
* Response – ответ-сообщение
* Request XSD/Response XSD – xml-схема запрос/ответ сообщений. Берется из ТЗ на интеграцию.
* Data xml - внутреннее представление данных, на основе которых создаются ответ-сообщения. Содержит контейнеры response c атрибутом name. Каждый контейнер response по структуре совпадает с корректным ответ-сообщением. Кроме того контейнер может содержать элемент header, в котором содержится заголовок ответ- сообщения.
* Data XSD – схема, которая отвечает за соответствие и корректность data xml .
* XSLT – преобразование, которое получает на вход запрос-сообщение, и выдает как результат ответ-сообщение. В своей работе xslt использует также файл data xml, получая из него значения элементов.

## Схема Драйвера

На каждый endpoint создается структура по преобразованию запросов в ответы:

Request XSD

Request

XSLT

Data XSD

Data xml

Обозначения на схеме:

* Request – запрос-сообщение
* Request XSD – xml-схема запрос- сообщений. Берется из ТЗ на интеграцию.
* Data xml - внутреннее представление данных, на основе которых создаются ответ-сообщения. Содержит контейнеры request c атрибутом name. Каждый контейнер request по структуре совпадает с корректным запрос-сообщением. Кроме того контейнер может содержать элемент header, в котором содержится заголовок запрос-сообщения.
* Data XSD – схема, которая отвечает за соответствие и корректность data xml .
* XSLT – преобразование, которое получает на вход data xml, и выдает как результат запрос-сообщение.

## Автогенерация эмуляторов и драйверов.

Для сокращения сроков создания эмуляторов и драйверов по новому ТЗ была создана процедура, которая пересоздает все текущие мок-сервисы и драйверы с текущими параметрами. Основные изменения в ТЗ (xsd-схемах запрос-ответ сообщений) не должны повлиять на работу этой процедуры.

Поэтому при изменении ТЗ (xsd-схем) следует:

1. скопировать их в папку «src\main\webapp\WEB-INF\xsd\{имя системы}\» (заменив старые).
2. запустить метод main в файле «src\main\java\ru\sbt\bpm\mock\utils\importXSD.java».
3. запустить автотесты «\src\test\java\ru\sbt\bpm\mock\tests\createMockOrDriverFromXSD.java» и «\src\test\java\ru\sbt\bpm\mock\tests\ChannelTest.java» для проверки результатов.

Успешное прохождение тестов не гарантирует успешное создание мок-сервисов и драйверов. Необходимо также вручную проверить наполнение xml данными.

Более подробно автогенерация описана в разделе «импорт XSD»

## Ручное создание эмулятора

Request XSD

Request

XSLT

Response

Response XSD

Data XSD

Data xml

### Исходные данные

Для создания эмулятора необходимо иметь XSD-схемы запросов и ответов.

### Примеры запрос-ответ сообщений

При создании сервиса необходимо получить примеры запрос-сообщения и ответ-сообщения. Примеры необходимы для:

* 1. Создания data xml
  2. Создания автотеста полученного мок-сервиса.

В текущей версии эмулятора создается по два примера ответ/запрос сообщений. Одна содержит каждый возможный элемент в возможном минимальном количестве (то есть каждого элемента по 1 экземпляру), второй пример содержит минимально возможное число элементов (опускает все необязательные элементы).

Для создания примера сообщения в автоматическом режиме используется преобразование XSDToExampleXML.xsl (src\main\webapp\WEB-INF\xsl\util\XSDToExampleXML.xsl). Преобразование получает на вход XSD со схемой, а на выходе получается промер сообщения.

Для преобразования в некоторых случаях придётся задать несколько параметров:

* *parrentXSDPath* – путь к общей xsd-схеме сообщения. По умолчанию «*../../xsd/CRM/CRM.xsd*». Относительный путь (от файла XSDToExampleXML.xsl) к объявлению общей схемы для всех сообщений системы. Для реализованных систем это:
  + «*../../xsd/CRM/CRM.xsd*» для CRM (значение по умолчанию)
  + «*../../xsd/AMRLiRT/AMRLIRT.xsd*» для AMRLIRT
  + «**../../xsd/FinRep/ASFO.xsd**» для FinRep

В случае, если этот параметр не задан нужно обязательно задать параметр rootXSD – namespace url контейнера запрос/ответ сообщения.

* entryPointName – имя типа главного элемента-контейнера операции. По умолчанию используется значение, взятое регулярным выражением из атрибута targetNamespace. Например, для создания примера запроса сервиса «CRM/CreateTask» следует использовать значение «CreateTaskRq», как значения атрибута name complexType из схемы CreateTaskRequest.xsd. В данном случае этот просмотр задавать не требуется, так как значение по умолчанию возьмёт его из неймспейса «http://sbrf.ru/NCP/CRM/CreateTaskRq/1.02/». В других случаях, например в запросе сервиса «CRM/GetParticipantsRequest» значение по умолчанию неправильно (targetNamespace - "http://sbrf.ru/NCP/CRM/GetParticipantsRq/1.03/", имя типа – «PrtspRq»).
* rootElementName – имя элемента-контейнера, в которую будет обернута операция. Значение по умолчанию некорректно и требует замены. Например для запроса сервиса «CRM/CreateTask» это значение равно «createTaskRq». Получается из файла схемы всех сообщений для системы:
  + «*../../xsd/CRM/CRM.xsd*» для CRM
  + «*../../xsd/AMRLiRT/AMRLIRT.xsd*» для AMRLIRT
  + «*../../xsd/FinRep/ASFO.xsd*» для FinRep
* parrentNS – общий неймспейс для системы. По-умолчанию получается с помощью параметра *parrentXSDPath.* Задавать следует только в том случае, если этот параметр не задан.
* systemName – имя системы. Реализованные системы:
  + «CRM» (значение по умолчанию)
  + «AMRLIRT»
  + «FinRep»
* omitComments – пропускать комментарии. Возможные значения:
  + «false» (значение по умолчанию) – в создаваемом сообщении будут проставлены xml-комментарии указывающие возможное число элементов
  + «true» - комментариев не будет
* showOptionalTags – вставлять опциональные элементы. Возможные значения:
  + «true» (значение по умолчанию) – в создаваемом сообщении будут созданы все входящие в схему элементы, по одному.
  + «false» - все элементы, вхождение которых не обязательно будут опущены.
* useLinkedTagValue – использовать тег для выбора ответа. Фактически означает, что один из тэгов будет использовать не стандартное значение, а указанное в следующих параметрах. Возможные значения:
  + «false» (значение по умолчанию) – все элементы будут иметь стандартные значения
  + «true» - элемент с именем, указанном в параметре *tagNameToTakeLinkedTag*, будет создаваться со значением указанном в параметре *linkedTagValue*. Необходимо для проверки в автотестах мок-сервисов.
* tagNameToTakeLinkedTag – имя элемента, используется в параметре useLinkedTagValue. Необходимо для проверки в автотестах мок-сервисов.
* linkedTagValue – значение элемента, используется в параметре useLinkedTagValue. Необходимо для проверки в автотестах мок-сервисов.

Результирующими файлами следует заменить файлы примеров сообщений по пути «\src\test\resources\xml\{имя системы}\{имя endpoint’а}».

В данной папке располагаются четыре файла – rq1.xml, rq2.xml, rs1.xml, rs2.xml. Разработанные автотесты проверяют что если на вход мок-сервису приходят rq1.xml, rq2.xml, то на выходе ответ соответствует сообщениям rs1.xml, rs2.xml.

С целью проверки корректности обработки в rq1.xml и rs1.xml содержаться все элементы из сообщений, а в rq2.xml и rs2.xml только обязательные.

В сообщение rq2.xml как значение в предварительно выбранном элементе вставляется значение «test1». По этому значению выбирается ответ из data-файла (см пункт 4, создание xslt).

### Создание Data XSD

Для проверки корректности xml с данными для ответа необходимо создать файл-схему для валидации этих файлов.

Результирующая схема получается по принципу:

* Изменяется неймспейс. Результирующий урл неймспейса получается из неймспейса ответа добавлением в конец «/Data/». Например для сервиса «CRM/CreateTask» namespace «http://sbrf.ru/NCP/CRM/CreateTaskRs/1.02/» преобразуется в «http://sbrf.ru/NCP/CRM/CreateTaskRs/1.02/Data/»
* Также необходимо убедиться, что все типы ссылаются на объявления внутри документа, или для типов добавлен импорт схемы. Например если в xsd тип complexType присутствует в файле, то остальные типы должны ссылаться именно на него, а не на типы из исходного неймспейса. Если же в файле присутвует ссылка на простой тип, например на c[tve CommonTypes.xsd, то в файле должен быть импорт этой схемы.

Для создания элемента в автоматическом режиме используется преобразование xsdToDataXsd.xsl (src\main\webapp\WEB-INF\xsl\util\ xsdToDataXsd.xsl ). Преобразование получает на вход XSD со схемой, а на выходе получается xsd файл.

Для преобразования в некоторых случаях придётся задать несколько параметров:

* entryPointName – имя типа главного элемента-контейнера операции. По умолчанию используется значение, взятое регулярным выражением из атрибута targetNamespace. Например, для создания примера запроса сервиса «CRM/CreateTask» следует использовать значение «CreateTaskRq», как значения атрибута name complexType из схемы CreateTaskRequest.xsd. В данном случае этот просмотр задавать не требуется, так как значение по умолчанию возьмёт его из неймспейса «http://sbrf.ru/NCP/CRM/CreateTaskRq/1.02/». В других случаях, например в запросе сервиса «CRM/GetParticipantsRequest» значение по умолчанию неправильно (targetNamespace - "http://sbrf.ru/NCP/CRM/GetParticipantsRq/1.03/", имя типа – «PrtspRq»).
* systemName – имя системы. Реализованные системы:
  + «CRM» (значение по умолчанию)
  + «AMRLIRT»
  + «FinRep»

Результатом преобразования следует заменить файл «\src\main\webapp\WEB-INF\data\{имя системы}\xsd\{имя endpoint’а}Data.xsd».

### Создание xml с данными

Для наполнения ответ-сообщения корректными и значащими данными нужно создать xml по шаблону:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <data xmlns="{неймспейс дата файла}">  <response name="default">  <… {данные из ответ-сообщения} …>  </response>  <response name="{имя ответа}">  <… {данные из ответ-сообщения} …>  </response>  <response name="{имя ответа №2}">  <… {данные из ответ-сообщения} …>  </response>  <… любое число ответов …>  </data> |

Наличие ответа с именем «default» обязательно.

Для автотестов мок-сервиса создается xml двумя ответами – «default» и «test1».

В первый вставляются данные из файла rs1.xml, во второй из файла rs2.xml (см. пункт 1, Примеры запрос-ответ сообщений). Автотест автоматически возьмет сообщения rq1.xml и rq2.xml и проверит, что ответ на них будет соответствовать сообщениям rs1.xml и rs2.xml.

Для создания элемента в автоматическом режиме используется преобразование AddExampleToData.xsl (src\main\webapp\WEB-INF\xsl\util\ AddExampleToData.xsl ). Преобразование получает на вход пример сообщения, а на выходе получается xml файл.

Параметры данного файла, которые необходимо задать:

* name – имя запроса/ответа – по умолчанию «default»
* dataFileName - путь к файлу с данными (ранее созданный файл аналогичный создаваемому)
* createEmptyData – Создать пустой data-файл. Возможные значения:
  + «false» (значение по умолчанию) – в создаваемом сообщении будут данные
  + «true» - файл будет содержать только рут-элементы
* Replace – Заменить файл. Возможные значения:
  + «false» (значение по умолчанию) – в создаваемом файле будут копированы старые значения, кроме элемента с именем name
  + «true» - файл будет содержать только одно значение – из входного файла.
* Type – тип ('response'/ 'request'). По умолчанию - 'response', которое и следует использовать. 'request' используется для драйверов, а не для мок-сервисов.

Как результат получается дата файл со вставленным/замененным примером.

Для работы автотестов следует сначала применить преобразование с заменой (Replace=true) над rs1.xml и второй раз с добавлением rs1.xml и новым именем (Replace=true, name=test1). Результат сохранить в файл «\src\main\webapp\WEB-INF\data\{имя системы}\xml\{имя endpoint’а}Data.xml».

В дальнейшем, данные файлы могут быть изменены по нуждам использования мок-сервисов через веб-интерфейс.

### Создание XSLT

Основная часть moc-сервиса – логика преобразования запроса в ответ – представляет собой xsl-файл.

Файл получает на вход запрос-сообщение, загружает data xml. По специальному заранее определенному значения элемента (для каждого endpoint’а задается отдельно) из запрос-сообщения получает имя ответа. Ищет в файле data xml ответ с таким же именем (если такого имени нет – берет ответ с именем «default»).

Например, запрос «CRM/CreateTask»:

|  |
| --- |
| <soap-env:Envelope xmlns:soap-env="http://sbrf.ru/NCP/esb/envelope/">  <soap-env:Header>  <soap-env:message-id>defaultId</soap-env:message-id>  <soap-env:request-time>2014-12-16T17:55:06.410+04:00</soap-env:request-time>  <soap-env:operation-name>createTaskRs</soap-env:operation-name>  </soap-env:Header>  <soap-env:Body>  <CRM:createTaskRs xmlns:CRM="http://sbrf.ru/NCP/CRM/"  xmlns:tns="http://sbrf.ru/NCP/CRM/CreateTaskRq/1.02/">  <tns:contractID>string</tns:contractID>  <tns:contractBPMID>string</tns:contractBPMID>  <tns:comment>test1</tns:comment>  <tns:requestType>string</tns:requestType>  <tns:responsiblePersonID>string</tns:responsiblePersonID>  </CRM:createTaskRs>  </soap-env:Body>  </soap-env:Envelope> |

В ходе генерации xsl было заранее определено, что выбор ответа будет определиться по элементу comment. Поэтому xsl загрузит data xml:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <data xmlns="http://sbrf.ru/NCP/CRM/CreateTaskRs/1.02/Data/">  <response name="default">  <contractID>string1</contractID>  <contractBPMID>string1</contractBPMID>  <comment>string1</comment>  <requestType>string1</requestType>  <responsiblePersonID>string1</responsiblePersonID>  <errorCode>string1</errorCode>  <errorMessage>string1</errorMessage>  </response>  <response name="test1">  <contractID>string2</contractID>  <contractBPMID>string2</contractBPMID>  <comment>string2</comment>  <requestType>string2</requestType>  <responsiblePersonID>string2</responsiblePersonID>  <errorCode>string2</errorCode>  </response>  </data> |

И попытается найти response с именем «test1» (если такого имени нет, то будет использовано имя «default»). По данным из элемента response и будет составлен ответ.

Для создания xsl в автоматическом режиме используется преобразование responceXSDtoXSL.xsl (src\main\webapp\WEB-INF\xsl\util\responceXSDtoXSL.xsl). Преобразование получает на вход xsd ответов, а на выходе получается xsl файл.

Для преобразования в некоторых случаях придётся задать несколько параметров:

* *entryPointName* – имя типа главного элемента-контейнера операции. По умолчанию используется значение, взятое регулярным выражением из атрибута targetNamespace. Например, для создания примера запроса сервиса «CRM/CreateTask» следует использовать значение «CreateTaskRq», как значения атрибута name complexType из схемы CreateTaskRequest.xsd. В данном случае этот просмотр задавать не требуется, так как значение по умолчанию возьмёт его из неймспейса «http://sbrf.ru/NCP/CRM/CreateTaskRq/1.02/». В других случаях, например в запросе сервиса «CRM/GetParticipantsRequest» значение по умолчанию неправильно (targetNamespace - "http://sbrf.ru/NCP/CRM/GetParticipantsRq/1.03/", имя типа – «PrtspRq»).
* *parrentXSDPath* – путь к общей xsd-схеме сообщения. По умолчанию «../../xsd/CRM/CRM.xsd». Относительный путь (от файла responceXSDtoXSL.xsl) к объявлению общей схемы для всех сообщений системы. Для реализованных систем это:
  + «../../xsd/CRM/CRM.xsd» для CRM (значение по умолчанию)
  + «../../xsd/AMRLiRT/AMRLIRT.xsd» для AMRLIRT
  + «../../xsd/FinRep/ASFO.xsd» для FinRep

В случае, если этот параметр не задан нужно обязательно задать параметр parrentNS

– namespace url контейнера запрос/ответ сообщения.

* parrentNS – имя элемента-контейнера, в которую будет обернута операция. Значение по умолчанию некорректно и требует замены. Например для запроса сервиса «CRM/CreateTask» это значение равно «createTaskRq». Получается из файла схемы всех сообщений для системы:
  + «../../xsd/CRM/CRM.xsd» для CRM
  + «../../xsd/AMRLiRT/AMRLIRT.xsd» для AMRLIRT
  + «../../xsd/FinRep/ASFO.xsd» для FinRep
* dataFileName – имя файла с данными. По умолчанию получается из параметра *entryPointName* заменой подстрок «Response» и «Rs» и добавление постфикса «Data.xml»
* systemName – имя системы. Реализованные системы:
  + «CRM» (значение по умолчанию)
  + «AMRLIRT»
  + «FinRep»
* rootElementName – имя элемента-контейнера, в которую будет обернута операция. Значение по умолчанию некорректно и требует замены. Например для запроса сервиса «CRM/CreateTask» это значение равно «createTaskRq». Получается из файла схемы всех сообщений для системы:
  + «../../xsd/CRM/CRM.xsd» для CRM
  + «../../xsd/AMRLiRT/AMRLIRT.xsd» для AMRLIRT
  + «../../xsd/FinRep/ASFO.xsd» для FinRep
* tagNameToTakeLinkedTag – имя элемента, который будет использоваться для определения, какой ответ будет взят из data xml. По умолчанию не задан – «\*»
* tagQuerryToTakeLinkedTag – xpath запроса, по которому будет определен, какой ответ будет взят из data xml. По умолчанию преобразует параметр tagNameToTakeLinkedTag в правильный запрос нахождения первого элемента по local-name. Менять следует для создания более сложных условий.

Результатом преобразования следует заменить файл «\src\main\webapp\WEB-INF\xsl\{имя системы}\{имя endpoint’а}.xsl».

## Ручное создание драйвера

Request XSD

Request

XSLT

Data XSD

Data xml

### Исходные данные

Для создания драйвера необходимо иметь XSD-схему запросов.

### Примеры запрос-ответ сообщений

При создании драйвера необходимо получить примеры запрос-сообщения. Примеры необходимы для:

1. Создания data xml
2. Создания автотеста полученного драйвера.

В текущей версии эмулятора создается два примера запрос-сообщений. Один содержит каждый возможный элемент в возможном минимальном количестве (то есть каждого элемента по 1 экземпляру), второй пример содержит минимально возможное число элементов (опускает все необязательные элементы).

Для создания примера сообщения в автоматическом режиме используется преобразование XSDToExampleXML.xsl (src\main\webapp\WEB-INF\xsl\util\XSDToExampleXML.xsl). Преобразование получает на вход XSD со схемой, а на выходе получается промер сообщения.

Для преобразования в некоторых случаях придётся задать несколько параметров:

* *parrentXSDPath* – путь к общей xsd-схеме сообщения. По умолчанию «*../../xsd/CRM/CRM.xsd*». Относительный путь (от файла XSDToExampleXML.xsl) к объявлению общей схемы для всех сообщений системы. Для реализованных систем это:
  + «*../../xsd/CRM/CRM.xsd*» для CRM (значение по умолчанию)
  + «*../../xsd/AMRLiRT/AMRLIRT.xsd*» для AMRLIRT
  + «**../../xsd/FinRep/ASFO.xsd**» для FinRep

В случае, если этот параметр не задан нужно обязательно задать параметр rootXSD – namespace url контейнера запрос/ответ сообщения.

* entryPointName – имя типа главного элемента-контейнера операции. По умолчанию используется значение, взятое регулярным выражением из атрибута targetNamespace. Например, для создания примера запроса сервиса «CRM/CreateTask» следует использовать значение «CreateTaskRq», как значения атрибута name complexType из схемы CreateTaskRequest.xsd. В данном случае этот просмотр задавать не требуется, так как значение по умолчанию возьмёт его из неймспейса «http://sbrf.ru/NCP/CRM/CreateTaskRq/1.02/». В других случаях, например в запросе сервиса «CRM/GetParticipantsRequest» значение по умолчанию неправильно (targetNamespace - "http://sbrf.ru/NCP/CRM/GetParticipantsRq/1.03/", имя типа – «PrtspRq»).
* rootElementName – имя элемента-контейнера, в которую будет обернута операция. Значение по умолчанию некорректно и требует замены. Например для запроса сервиса «CRM/CreateTask» это значение равно «createTaskRq». Получается из файла схемы всех сообщений для системы:
  + «*../../xsd/CRM/CRM.xsd*» для CRM
  + «*../../xsd/AMRLiRT/AMRLIRT.xsd*» для AMRLIRT
  + «*../../xsd/FinRep/ASFO.xsd*» для FinRep
* parrentNS – общий неймспейс для системы. По-умолчанию получается с помощью параметра *parrentXSDPath.* Задавать следует только в том случае, если этот параметр не задан.
* systemName – имя системы. Реализованные системы:
  + «CRM» (значение по умолчанию)
  + «AMRLIRT»
  + «FinRep»
* omitComments – пропускать комментарии. Возможные значения:
  + «false» (значение по умолчанию) – в создаваемом сообщении будут проставлены xml-комментарии указывающие возможное число элементов
  + «true» - комментариев не будет
* showOptionalTags – вставлять опциональные элементы. Возможные значения:
  + «true» (значение по умолчанию) – в создаваемом сообщении будут созданы все входящие в схему элементы, по одному.
  + «false» - все элементы, вхождение которых не обязательно будут опущены.
* useLinkedTagValue – использовать тег для выбора ответа. Фактически означает, что один из тэгов будет использовать не стандартное значение, а указанное в следующих параметрах. Возможные значения:
  + «false» (значение по умолчанию) – все элементы будут иметь стандартные значения
  + «true» - элемент с именем, указанном в параметре *tagNameToTakeLinkedTag*, будет создаваться со значением указанном в параметре *linkedTagValue*. Необходимо для проверки в автотестах мок-сервисов.
* tagNameToTakeLinkedTag – имя элемента, используется в параметре useLinkedTagValue. Необходимо для проверки в автотестах мок-сервисов.
* linkedTagValue – значение элемента, используется в параметре useLinkedTagValue. Необходимо для проверки в автотестах мок-сервисов.

Результирующими файлами следует заменить файлы примеров сообщений по пути «\src\test\resources\xml\{имя системы}\{имя endpoint’а}».

В данной папке располагаются два файла – rq1.xml, rq2.xml. Разработанные автотесты проверяют что на выходе ответ соответствует сообщениям rq1.xml, rq2.xml.

С целью проверки корректности обработки в rq1.xml содержаться все элементы из сообщений, а в rq2.xml только обязательные.

### Создание Data XSD

Для проверки корректности xml с данными для ответа необходимо создать файл-схему для валидации этих файлов.

Результирующая схема получается по принципу:

* Изменяется неймспейс. Результирующий урл неймспейса получается из неймспейса ответа добавлением в конец «/Data/». Например для драйвера «CRM/ForceSignal» namespace «http://sbrf.ru/NCP/CRM/ForceSignalRq/1.03/» преобразуется в «http://sbrf.ru/NCP/CRM/ForceSignalRq/1.03/ Data/»
* Также необходимо убедиться, что все типы ссылаются на объявления внутри документа, или для типов добавлен импорт схемы. Например если в xsd тип complexType присутствует в файле, то остальные типы должны ссылаться именно на него, а не на типы из исходного неймспейса. Если же в файле присутвует ссылка на простой тип, например на c[tve CommonTypes.xsd, то в файле должен быть импорт этой схемы.

Для создания элемента в автоматическом режиме используется преобразование xsdToDataXsd.xsl (src\main\webapp\WEB-INF\xsl\util\ xsdToDataXsd.xsl ). Преобразование получает на вход XSD со схемой, а на выходе получается xsd файл.

Для преобразования в некоторых случаях придётся задать несколько параметров:

* entryPointName – имя типа главного элемента-контейнера операции. По умолчанию используется значение, взятое регулярным выражением из атрибута targetNamespace. Например, для создания примера запроса сервиса «CRM/CreateTask» следует использовать значение «CreateTaskRq», как значения атрибута name complexType из схемы CreateTaskRequest.xsd. В данном случае этот просмотр задавать не требуется, так как значение по умолчанию возьмёт его из неймспейса «http://sbrf.ru/NCP/CRM/ ForceSignalRq /1.03/». В других случаях, например в запросе сервиса «CRM/GetParticipantsRequest» значение по умолчанию неправильно (targetNamespace - "http://sbrf.ru/NCP/CRM/GetParticipantsRq/1.03/", имя типа – «PrtspRq»).
* systemName – имя системы. Реализованные системы:
  + «CRM» (значение по умолчанию)
  + «AMRLIRT»
  + «FinRep»

Результатом преобразования следует заменить файл «\src\main\webapp\WEB-INF\data\{имя системы}\xsd\{имя endpoint’а}Data.xsd».

### Создание xml с данными

Для наполнения ответ-сообщения корректными и значащими данными нужно создать xml по шаблону:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <data xmlns="{неймспейс дата файла}">  <request name="default">  <… {данные из запрос-сообщения} …>  </request>  <request name="{имя запроса }">  <… {данные из запрос-сообщения} …>  </request>  <request name="{имя запроса №2}">  <… {данные из запрос-сообщения} …>  </request>  <… любое число ответов …>  </data> |

Наличие запроса с именем «default» обязательно.

Для автотестов мок-сервиса создается xml двумя ответами – «default» и «test1».

В первый вставляются данные из файла rq1.xml, во второй из файла rq2.xml (см. пункт 1, Примеры запрос-ответ сообщений). Автотест проверит, что запрос будет соответствовать сообщениям rs1.xml и rs2.xml.

Для создания элемента в автоматическом режиме используется преобразование AddExampleToData.xsl (src\main\webapp\WEB-INF\xsl\util\ AddExampleToData.xsl ). Преобразование получает на вход пример сообщения, а на выходе получается xml файл.

Параметры данного файла, которые необходимо задать:

* name – имя запроса/ответа – по умолчанию «default»
* dataFileName - путь к файлу с данными (ранее созданный файл аналогичный создаваемому)
* createEmptyData – Создать пустой data-файл. Возможные значения:
  + «false» (значение по умолчанию) – в создаваемом сообщении будут данные
  + «true» - файл будет содержать только рут-элементы
* Replace – Заменить файл. Возможные значения:
  + «false» (значение по умолчанию) – в создаваемом файле будут копированы старые значения, кроме элемента с именем name
  + «true» - файл будет содержать только одно значение – из входного файла.
* Type – тип ('response'/ 'request'). По умолчанию - 'response', которое следует использовать мок-сервисов. Для драйверов следует установить 'request'.

Как результат получается дата файл со вставленным/замененным примером.

Для работы автотестов следует сначала применить преобразование с заменой (Replace=true) над rq1.xml и второй раз с добавлением rq1.xml и новым именем (Replace=true, name=test1). Результат сохранить в файл «\src\main\webapp\WEB-INF\data\{имя системы}\xml\{имя endpoint’а}Data.xml».

В дальнейшем, данные файлы могут быть изменены по нуждам использования драйверов через веб-интерфейс.

### Создание XSLT

Основная часть драйвера – логика преобразования файла с данными в запрос – представляет собой xsl-файл.

Файл получает на вход data xml. В параметре «name» получает имя ответа. Ищет в файле data xml ответ с таким же именем (если такого имени нет – берет ответ с именем «default»).

И попытается найти response с именем из параметра name (если такого имени нет, то будет использовано имя «default»). По данным из элемента response и будет составлен ответ.

Для создания xsl в автоматическом режиме используется преобразование requestXSDtoXSL.xsl (src\main\webapp\WEB-INF\xsl\util\ requestXSDtoXSL.xsl). Преобразование получает на вход xsd запросов, а на выходе получается xsl файл.

Для преобразования в некоторых случаях придётся задать несколько параметров:

* *entryPointName* – имя типа главного элемента-контейнера операции. По умолчанию используется значение, взятое регулярным выражением из атрибута targetNamespace. Например, для создания примера запроса сервиса «CRM/CreateTask» следует использовать значение «CreateTaskRq», как значения атрибута name complexType из схемы CreateTaskRequest.xsd. В данном случае этот просмотр задавать не требуется, так как значение по умолчанию возьмёт его из неймспейса «http://sbrf.ru/NCP/CRM/CreateTaskRq/1.02/». В других случаях, например в запросе сервиса «CRM/GetParticipantsRequest» значение по умолчанию неправильно (targetNamespace - "http://sbrf.ru/NCP/CRM/GetParticipantsRq/1.03/", имя типа – «PrtspRq»).
* *parrentXSDPath* – путь к общей xsd-схеме сообщения. По умолчанию «../../xsd/CRM/CRM.xsd». Относительный путь (от файла responceXSDtoXSL.xsl) к объявлению общей схемы для всех сообщений системы. Для реализованных систем это:
  + «../../xsd/CRM/CRM.xsd» для CRM (значение по умолчанию)
  + «../../xsd/AMRLiRT/AMRLIRT.xsd» для AMRLIRT
  + «../../xsd/FinRep/ASFO.xsd» для FinRep

В случае, если этот параметр не задан нужно обязательно задать параметр parrentNS

– namespace url контейнера запрос/ответ сообщения.

* parrentNS – имя элемента-контейнера, в которую будет обернута операция. Значение по умолчанию некорректно и требует замены. Например для запроса сервиса «CRM/CreateTask» это значение равно «createTaskRq». Получается из файла схемы всех сообщений для системы:
  + «../../xsd/CRM/CRM.xsd» для CRM
  + «../../xsd/AMRLiRT/AMRLIRT.xsd» для AMRLIRT
  + «../../xsd/FinRep/ASFO.xsd» для FinRep
* dataFileName – имя файла с данными. По умолчанию получается из параметра *entryPointName* заменой подстрок «Response» и «Rs» и добавление постфикса «Data.xml»
* systemName – имя системы. Реализованные системы:
  + «CRM» (значение по умолчанию)
  + «AMRLIRT»
  + «FinRep»
* rootElementName – имя элемента-контейнера, в которую будет обернута операция. Значение по умолчанию некорректно и требует замены. Например для запроса сервиса «CRM/CreateTask» это значение равно «createTaskRq». Получается из файла схемы всех сообщений для системы:
  + «../../xsd/CRM/CRM.xsd» для CRM
  + «../../xsd/AMRLiRT/AMRLIRT.xsd» для AMRLIRT
  + «../../xsd/FinRep/ASFO.xsd» для FinRep

Результатом преобразования следует заменить файл «\src\main\webapp\WEB-INF\xsl\{имя системы}\{имя endpoint’а}.xsl».

## Импорт XSD

Для сокращения сроков создания эмуляторов и драйверов по новому ТЗ была создана процедура, которая пересоздает все текущие мок-сервисы и драйверы с текущими параметрами. Основные изменения в ТЗ (xsd-схемах запрос-ответ сообщений) не должны повлиять на работу этой процедуры.

Точкой входа для вызова процедуры является метод main в файле «src\main\java\ru\sbt\bpm\mock\utils\importXSD.java».

Процедура main в свою очередь вызывает процедуру renewData, которая производит все необходимые действия.

renewData вызывает для каждой точки интеграции функции driverCycle или mockCycle, передавая им все необходимые параметры.

Параметры используемые в данной функции:

* rootElementName – имя элемента-контейнера, в которую будет обернута операция. Значение по умолчанию некорректно и требует замены. Например для запроса сервиса «CRM/CreateTask» это значение равно «createTaskRq». Получается из файла схемы всех сообщений для системы:
  + «*../../xsd/CRM/CRM.xsd*» для CRM
  + «*../../xsd/AMRLiRT/AMRLIRT.xsd*» для AMRLIRT
  + «*../../xsd/FinRep/ASFO.xsd*» для FinRep
* RqRootElementName – аналогично rootElementName, но для тэга запроса.
* entryPointName – имя типа главного элемента-контейнера операции. По умолчанию используется значение, взятое регулярным выражением из атрибута targetNamespace из XSD. Например, для создания примера запроса сервиса «CRM/CreateTask» следует использовать значение «CreateTaskRq», как значения атрибута name complexType из схемы CreateTaskRequest.xsd. В данном случае этот просмотр задавать не требуется, так как значение по умолчанию возьмёт его из неймспейса «http://sbrf.ru/NCP/CRM/CreateTaskRq/1.02/». В других случаях, например в запросе сервиса «CRM/GetParticipantsRequest» значение по умолчанию неправильно (targetNamespace - "http://sbrf.ru/NCP/CRM/GetParticipantsRq/1.03/", имя типа – «PrtspRq»).
* RqEntryPointName – аналогично entryPointName, но для тэга запроса.
* tagNameToTakeLinkedTag – имя элемента, который будет использоваться для определения, какой ответ будет взят из data xml. По умолчанию не задан – «\*»
* dataFileName – имя файла с данными. По умолчанию получается из параметра *entryPointName* заменой подстрок «Response» и «Rs» и добавление постфикса «Data.xml»
* systemName – имя системы. Реализованные системы:
  + «CRM» (значение по умолчанию)
  + «AMRLIRT»
  + «FinRep»
* parrentXSDPath – путь к общей xsd-схеме сообщения. По умолчанию «../../xsd/CRM/CRM.xsd». Относительный путь (от файла responceXSDtoXSL.xsl) к объявлению общей схемы для всех сообщений системы. Для реализованных систем это:
  + «../../xsd/CRM/CRM.xsd» для CRM (значение по умолчанию)
  + «../../xsd/AMRLiRT/AMRLIRT.xsd» для AMRLIRT
  + «../../xsd/FinRep/ASFO.xsd» для FinRep

В случае, если этот параметр не задан нужно обязательно задать параметр parrentNS

– namespace url контейнера запрос/ответ сообщения.