[Содержание](#Содержание)

[Трасформации данных](#Трасформации_данных)

# [Отладка трансформаций](#Отладка_трансформаций)

# [Простые трансформации (ST)](#Простые_трансформации)

[Цепочки загрузки данных](#Цепочки_загрузки)

* [*Программа ведения цепочек*](#Программа_ведения_цепочек)
* [*Просмотр журнала выполнения цепочки*](#Просмотр_журнала_выполнения)
* [*Цепочки с ручным запуском*](#Ручной_запуск_цепочки)
* [*Что делать, если в журнале цепочки – красный кружок /ошибка/*](#Ошибка_загрузки)
* [*Основные цепочки*](#Основные_цепочки)

**Трасформации данных**

[Содержание](#Содержание)

***Преобразование на стороне ABAP приложения***

В основе данного подхода лежит осуществление выполнения правил преобразования, программы запуска, программы завершения и эксперт рутины силами самого ABAP приложения.

***Последовательность выполнения трансформации***

1. Сервер ABAP приложения посылает запрос на сервер БД для выборки данных из источника.
2. Сервер БД выполняет запрос и результат отправляет на Сервер ABAP приложения.
3. Внутри сервера приложения ABAP осуществляется преобразование данных (Выполнение правил, программы запуска, программы завершения и т.д.).
4. Результат выполнения ABAP приложения отправляется на сервер БД.
5. Сервер БД записывает результат в цель.

***Дополнительные факторы***

* Преобразование данных осуществляется силами Сервера приложений.
* Обработка данных в пакете осуществляется по записям.
* Процедура позволяет определять правила на основе полей, которые в дальнейшем будут выполняться последовательно (т.е. изначально выполняется программа запуска, после чего осуществляется выполнение правил присвоения и т.д.).

## 

## *BW преобразование на основе HANA*

Данная реализация позволяет выполнить стандартное преобразование BW на сервере БД. В данном случае вся логика преобразования BW преобразовывается в Calculation Scenario и хранится в виде преобразования HANA в BW.

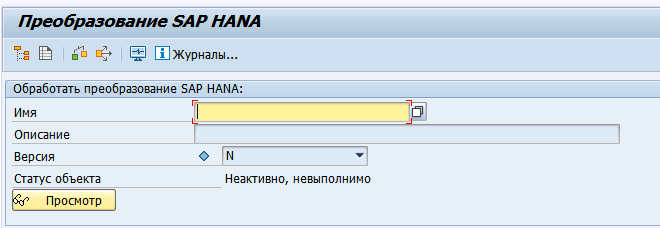
Посмотреть преобразование можно 2 способами

* Транзакция **RSDHATR (Рис 1.).**
* Через трансформацию.

# 

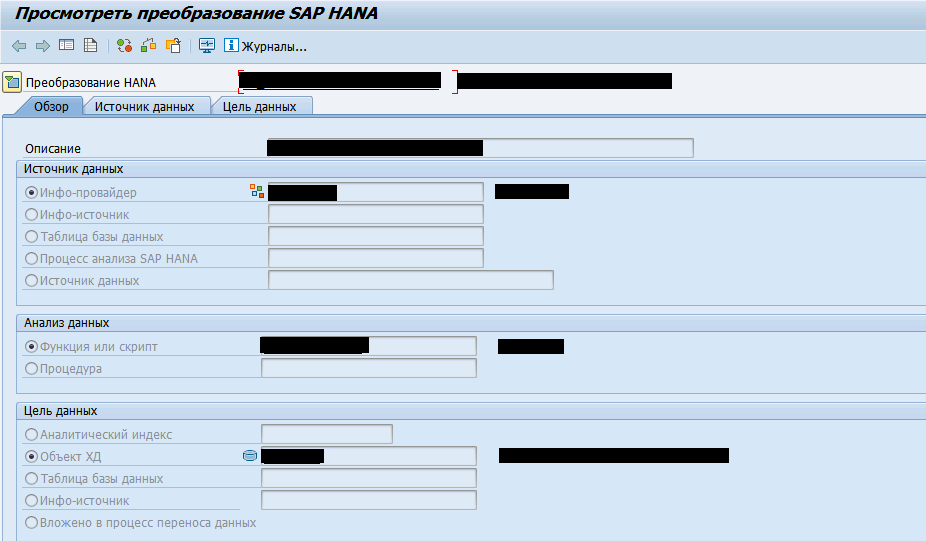
# ***Транзакция RSDHATR***

Чтобы найти нужное преобразование, необходимо в поле «Имя» через поиск (F4) указать тех. имя преобразования HANA, после чего нажать на кнопку «Просмотр». **(Рис.1.)**



**Рис 1.** - Транзакция RSDHATR

После нажатия на кнопку «Просмотр» открывается описание преобразования, в котором указаны: источник данных, цель данных и сценарий преобразования. **(Рис 2.).**

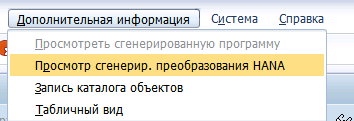


**Рис 2.** - Просмотр преобразования SAP HANA

# 

# ***Просмотр через трансформацию***

Посмотреть преобразование SAP HANA можно также через трансформацию, на которой оно используется. Для этого необходимо перейти на вкладку «Дополнительная информация» => «Просмотр сгенер. преобразования HANA» **(Рис 3.).**



**Рис 3.** - Переход на просмотр преобразования на основе HANA

В дальнейшем, Calcution Scenario встраивается в ColumnView, на основе которого DTP формирует SQL Select с учетом логики преобразования, что позволяет перенести данные из источника данных в цель за один шаг обработки, исключая передачу данных между сервером БД и сервером ABAP приложения. Технически данный процесс можно представить оператором «INSERT AS SELECT».

После того, как трансформация активирована, система проверяет возможность выполнения преобразования в SAP HANA. В случаи отсутствия противоречий, препятствующих выполнению преобразования BW на стороне SAP HANA, в DTP доступна возможность выбора параметра «Выполнения SAP HANA» (Рис 4.)

В данной ситуации в системе хранится два сценария выполнения программы: на сервере ABAP приложения и на сервере HANA, что в любой момент позволяет переключать между собой режимы по необходимости (Рис.4.).

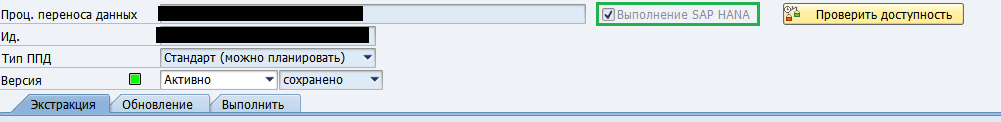


Рис 4. - Включение выполнения преобразования BW на основе HANA

***Expert Script SAP HANA***

Описанный процесс является более сложным для реализации, чем выполнение трансформации BW на HANA и не имеет возможности переключения режима обработки на ABAP, т.к. его невозможно реализовать при помощи стандартного функционала BW. Для реализации преобразования придется использовать инструменты, позволяющие моделировать SAP BW на основе HANA.

***Инструменты для разработки***

* SAP HANA Studio
* SAP BW Modeling Tools for Eclipse
* ABAP Development Tools for Eclipse

Expert Script SAP HANA похож на Expert Routine ABAP, отличие между которыми заключается в процессе обработки данных. В случаи с HANA обработка пишется на SQL Script и выполняется на сервере SAP HANA попакетно, в то время как ABAP выполняет обработку построчно.

Если рассматривать Expert Script SAP HANA с технической точки зрения, то это процедура БД SAP HANA обернутая в метод ABAP класса.

***Дополнительные факторы***

* ABAP обладает большей функциональностью, чем SQL Script.
* Не всегда Expert Script SAP HANA быстрее преобразований реализованных на ABAP.

На Рис 5.Схематически представлено отличие преобразования данных при помощи стандартных средств BW и с использованием возможностей HANA.

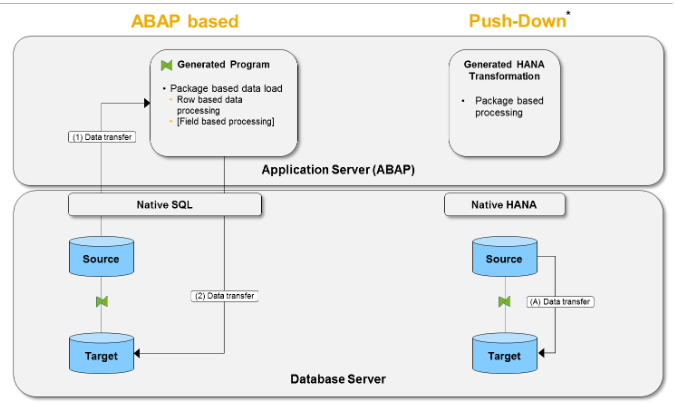


Рис 5. **-** Отличие преобразования на стороне сервера приложений и сервера БД

# 

# ***Предпосылки к использованию и ограничения***

Одним из основных аспектов разработки Expert Script SAP HANA является производительность.

## 

## *Анализ и решение*

Прежде чем реализовывать Expert Script SAP HANA, необходимо провести анализ, на основе которого принять правильное решение.

Ниже представлены соображения, которых необходимо придерживаться во время принятия решения

* Если большее количество времени занимает обработка ABAP или передача данных между сервером приложений и сервером БД, то в данном случае реализация преобразования через Expert Script SAP HANA будет эффективна.
* В ситуации, когда большее время затрачивается на обработку БД, реализация преобразования в HANA будет не эффективна.
* Необходимо учитывать, что в роли целевых объектов в версии SAP BW 7.4 SP5 могут являться, только
  + Классические DSO (Стандартная, С оптимизированной записью)
  + Расширенная DSO (ADSO)
  + Семантически-разделенная DSO
  + Open Hub
* Отсутствует возможность обработки ошибок, т.е. преобразование всех записей должно завершиться успешно, нет возможности обработать ошибочные записи повторно.

## *Объекты, не поддерживающиеся в трансформации HANA*

* Нельзя использовать запросы как инфо-провайдеры в качестве источника.
* Группы правил не поддерживаются.
* Не поддерживаются ABAP программы запуска, завершения, эксперт рутины.
* Не поддерживаются пользовательские функции в формулах.
* Семантические группы не поддерживаются.

## 

## *Рекомендации*

* Во время разработки Expert Script SAP HANA рекомендуется использовать сценарии SQL, предопределенные БД HANA и избегать обработок данных построчно, т.е. не использовать циклы и курсоры БД.
* По возможности максимально использовать стандартные возможности преобразования - в SAP BW доступна возможность связывать цель и источник данных несколькими трансформациями, используя infoSources.

# **Отладка трансформаций**

[Содержание](#Содержание)

В BW с HANA подпрограммы в трансформациях можно делать как с использованием ABAP, так и используя AMDP скрипты. Механизмы отладки у этих двух вариантов преобразований разный.

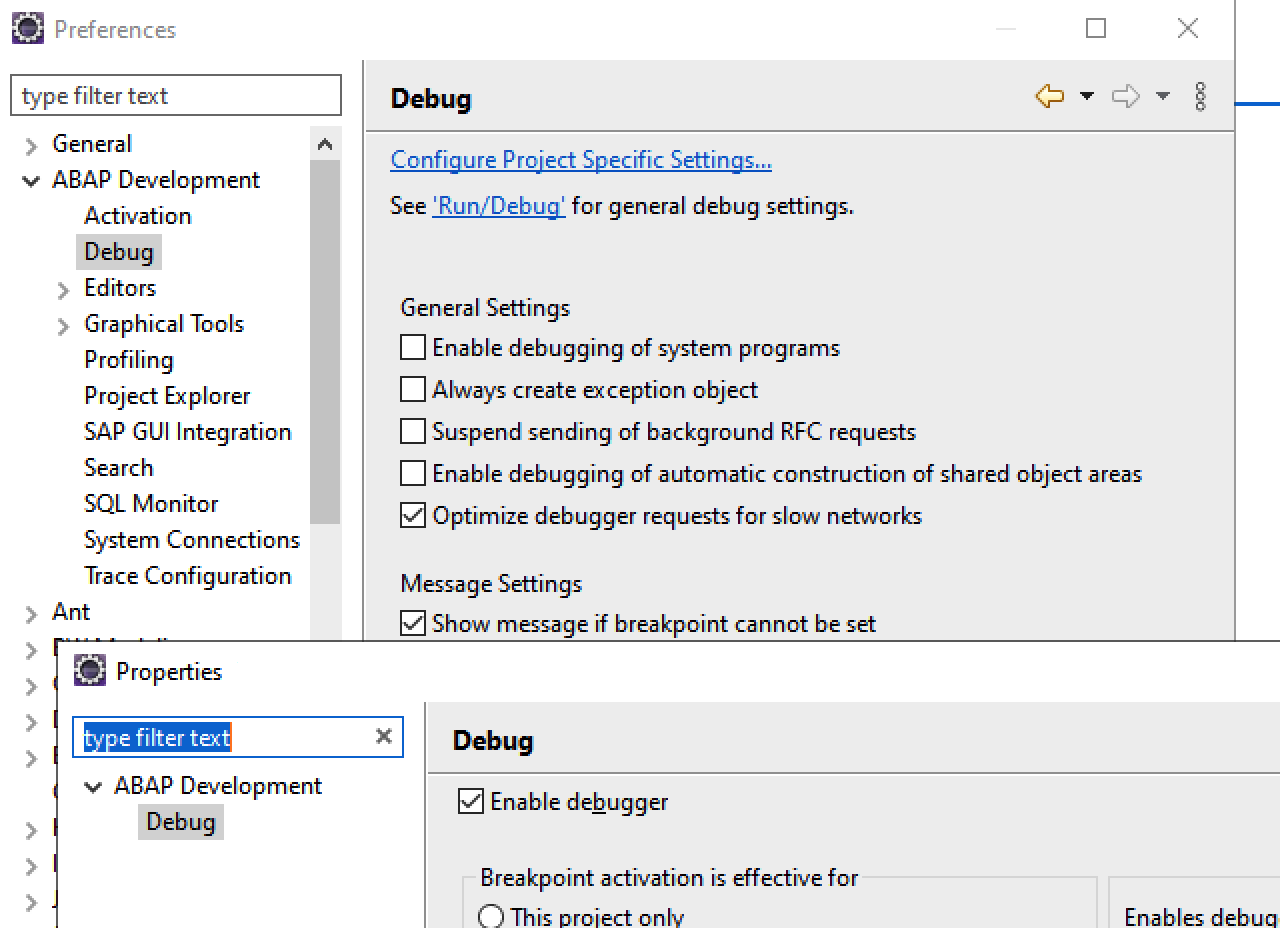
#### **Отладка ABAP преобразований**

При попытке отладить АБАП подпрограмму мы можем получить ошибку

**«Breakpoints cannot be set because of external context (–> long text»,** - в таком случае в настройках Eclipse нужно отключить дебаггер. После этого можно поставить точку останова в SAP GUI в сгенерированной программе преобразования (либо написать BREAK-POINT) и запустить ППД (DTP) в режиме отладки.  После этого успешно остановится программа в сгенерированной подпрограмме.

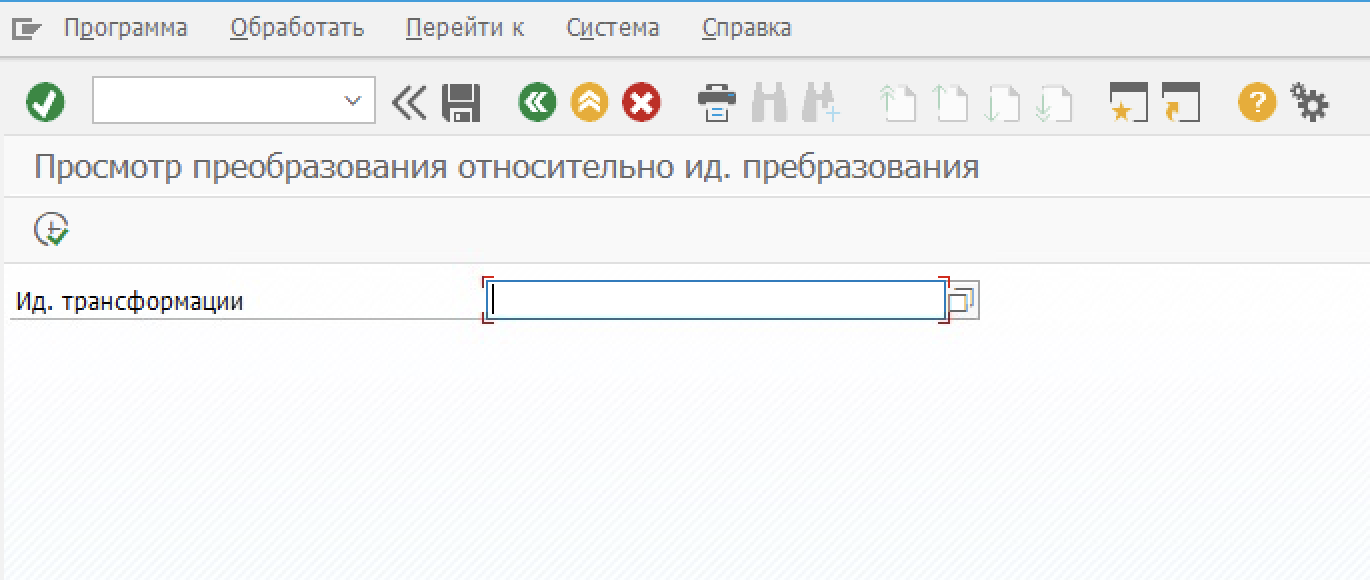
***Как отключить надстройку***

**Windows -> Preference -> ABAP Development -> Debug -> Configure Project Specific Settings -> Project D41 -> ‘Enable Debugger’ checkbox would be enabled, please disable it -> Apply and Close**



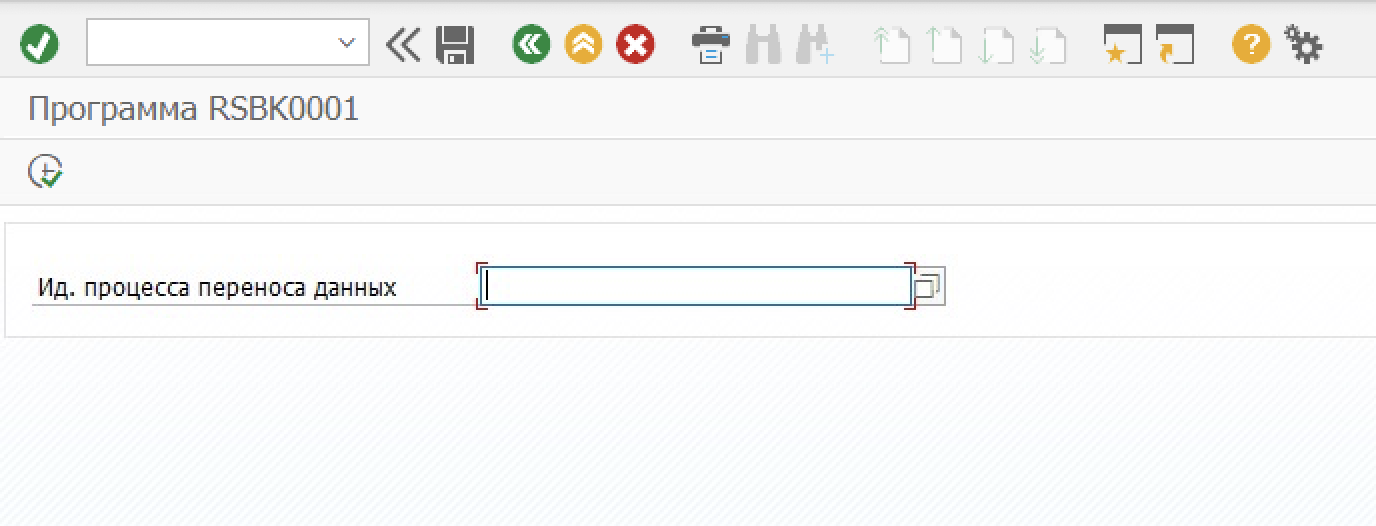
***Как посмотреть сгенерированную программу трансформации в BW4HANA?***

Используйте транзакцию **RSTRANGUI**, что бы открыть трансформацию.



Через этот интерфейс можно открыть сгенерированную программу через «Дополнительная информация» \ «Посмотреть сгенерированную программу». Там ставим точку останова.

Через транзакцию **RSDTP** открываем ППД и запускаем его в режиме «Моделирование» и галочку «Экспертный режим» и после этого программа остановится на коде.



Результат загрузки в ADSO тоже можно посмотреть через транзакцию RSMNG (программа RSPM\_MANAGE).

# **Простые трансформации (ST)**

[Содержание](#Содержание)

Простые трансформации ST (Simple Transformation) позволяют преобразовывать данные из ABAP в XML (Сериализация) и из XML в ABAP (Десериализация). Простые трансформации представляют из себя специальный вид программ, которые при желании можно отлаживать как обычные ABAP программы.

Запустить подобную программу можно с помощью ABAP оператора CALL TRANSFORMATION.

Основные характеристики ST программ, в сравнении с XSLT

* Повышенная производительность, за счёт последовательного доступа к XML элементам.
* Благодаря специальному синтаксису, используемому в данных программах, одна программа может быть использована как для сериализации, так и для десериализации.
* Трансформация из одного набора данных ABAP в другой набор ABAP данных или из одного XML документа в другой не поддерживается, но возможна через два вызова трансформации.
* Преобразование ссылочных переменных и объектов в настоящее время не поддерживается. Так же не поддерживается трансформация структур и таблиц, компонентами которых являются ссылочные переменные. Несмотря на это, допускается работа со ссылочными переменными (например, для вызова метода класса).

ST программы создаются либо через SE80, либо через транзакцию STRANS. ST программа должна представлять собой XML документ, со следующей структурой

<?sap.transform simple?>

<tt:transform [template="tmpl"]

              xmlns:tt="http://www.sap.com/transformation-templates">

    [<tt:type name="..." [...]>

     ...

    </tt:type>

     ...]

    [<tt:root name="root1" [...] />

     <tt:root name="root2" [...] />

   ...]

  [<tt:parameter name="para1" [...] />

   <tt:parameter name="para2" [...] />

   ...]

  [<tt:variable name="vari1" [...] />

   <tt:variable name="vari2" [...] />

   ...]

  <tt:template [name=="tmpl"]>

    ...

  </tt:template>

  [<tt:template [name=="..."]>

     ...

   </tt:template>

   ...]

</tt:transform>

Первая строка <? … ?> используется для идентификации типа программы. Нет необходимости вводить её вручную, т.к. при проверке синтаксиса она будет введена автоматически.

Главный элемент ST программы tt:transform, в котором описаны пространства имен и главный шаблон. Элементы пространства имен «http://www.sap.com/transformation-templates» являются командами ST программы. Префикс пространства имен tt: используется для отображения команд из этого пространства. ST команды могут выступать в роли узлов XML документа или атрибутов в тестовых XML элементах (например при определении значения для текстового XML элемента).

В ST программах допускается использование обыкновенного текста, который при сериализации подставляется в созданный XML документ, данный текст обычно представляет набор тегов XML.

При десериализации происходит следующее: XML документ представляет собой входной поток, из которого по очереди выбираются элементы и в случае их совпадения с элементами в ST программе, их значения подставляются в соответствующие, привязанные к ним значения в ST программе (ABAP переменные). В зависимости от логики, заложенной в ST программе, элементы могут пропускаться или объединяться.

Кроме того можно вставлять комментарии, т.к. ST программа это XML документ, комментарии определяются теми же правилами которые действуют для XML документа

<!-- Comment Text -->

Имена в текстовых элементах зависят от регистра, т.о. <ELEMENT> и <element> два разных элемента.

Основная логика преобразования лежит внутри так называемых шаблонов, которые включают в себя набор ST команд. Когда вы определяете главный элемент **tt:transform** вы можете указать главный шаблон через атрибут **template**. Это первый шаблон, который выполнит ST программа при своем запуске. Если главный шаблон явно не определен, в ST программе должен присутствовать шаблон без имени, который будет запущен в роли главного. Шаблоны определяются командой tt:template.

Кроме определения главного шаблона и главного элемента, ST программы еще могут содержать следующие элементы

* Корневые узлы данных (Data Roots) определяющие связь между ABAP данными и ST программой (определяются командой tt:root);
* Параметры, которые могут быть переданы через оператор CALL TRANSFORMATION (определяются командой tt:parameter);
* Переменные (определяются командой tt:variable);
* Типы данных, используемые в трансформации (определяются через оператор tt:type), заданные таким образом типы, можно использовать как в узлах данных, так и в параметрах и переменных;
* Вложенные шаблоны, используемые в качестве разбиения логики ST программы на части.

## Определение корневых узлов параметров и переменных

### 

### ***Определение корневых узлов***

Для доступа к данным из ABAP программы, ST программа должна иметь хотя бы одно определение корневого узла. Корневые узлы представляют собой интерфейс по обмену данными из ST в ABAP программу и обратно, переменные из ABAP программы подставляются в параметрах source (сериализация) и result (десериализация) оператора CALL TRANSFORMATION.

Корневые узлы могут быть объявлены на уровне всей ST программы (tt:transformation) и на уровне вложенных шаблонов. Во вложенных шаблонах они не связаны с ABAP данными и используются для связи с корневыми узлами на верхнем уровне ST программы, при передаче через команду tt:with-root и объявляются в элементе tt:context.

Синтаксис при объявлении корневых узлов следующий

<tt:root name="..." [[line-]type="..."[length="..."][decimals="..."]][...] />

В качестве атрибута name указывается ABAP переменная, с которой необходимо связать узел, имя узла должно быть уникальным как среди других корневых узлов, так и среди объявленных в трансформации переменных (tt:variable) и параметров (tt:parameter). Имя не чувствительно к регистру.

Тип данных указывается через атрибут line-type или type. Line-type используется в случае указания на внутреннюю таблицу.

Если явно не указать тип данных, проверки сработают только во время выполнения трансформации, что в случае неверных данных может приводить к ошибкам

* К доступу несуществующего элемента структуры
* Циклу не по табличной переменной
* Обработки структуры как обычной переменной

В качестве типа могут быть указаны как элементарные ABAP типы, так и типы из словаря, и типы, объявленные через команду tt:type. При сериализации значения корневых узлов не могут быть изменены (например, через tt:assign) и доступны только для чтения, при десериализации только для записи (например, через tt:value).

Length – указывает на длину узла, Decimals на количество знаков после запятой, эти параметры указываются в случае необходимости для соответствующих им типов данных.

В приведенном ниже примере узел R1 объявлен с элементарным типом D, R2 – как таблица, состоящая из элементарного типа I, R3 — как структура, из ABAP словаря, R4 – Структура из пула типов, R5 – внутренняя таблица из структуры в классе cl\_abc, R6 – структура с типом из трансформации. Обратите внимание на использование пространств имен.

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates"

              xmlns:ddic="http://www.sap.com/abapxml/types/dictionary"

              xmlns:tp="http://www.sap.com/abapxml/types/type-pool/ABCD"

              xmlns:cl="http://www.sap.com/abapxml/types/class-pool/CL\_ABC"

              xmlns:def="http://www.sap.com/abapxml/types/defined" >

  <tt:type name="STRUCT">

    <tt:node name="C1" type="I"/>

    <tt:node name="C2" type="I"/>

  </tt:type>

  <tt:root name="R1" type="D"/>

  <tt:root name="R2" line-type="I"/>

  <tt:root name="R3" type="ddic:DSTRUCT"/>

  <tt:root name="R4" type="tp:ABCDSTRUCT"/>

  <tt:root name="R5" line-type="cl:ABCSTRUCT"/>

  <tt:root name="R6" type="def:STRUCT"/>

Доступ к компонентам узла осуществляется через «.». Ниже приведен небольшой пример (в приведенных ниже примерах проверка исключений во время трансформаций опущена специально, в продуктивных системах подобного стоит избегать, используя оператор TRY..Catch)

DATA:

  gt\_test\_struct TYPE addr,

  lv\_xml         TYPE string.

gt\_test\_struct-name1 = 'Mike'.

gt\_test\_struct-line1 = 'Leningradkaya St'.

CALL TRANSFORMATION ztest\_st

      SOURCE struct = gt\_test\_struct

      RESULT XML lv\_xml.

WRITE lv\_xml.

SKIP. CLEAR gt\_test\_struct.

CALL TRANSFORMATION ztest\_st

     SOURCE XML lv\_xml

     RESULT struct = gt\_test\_struct.

WRITE: / gt\_test\_struct-name1,

         gt\_test\_struct-line1.

Трансформация

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="struct"/>

  <tt:template>

    <lines>

      <name>

        <tt:value ref="struct.name1"/>

      </name>

      <line>

        <tt:value ref="struct.line1"/>

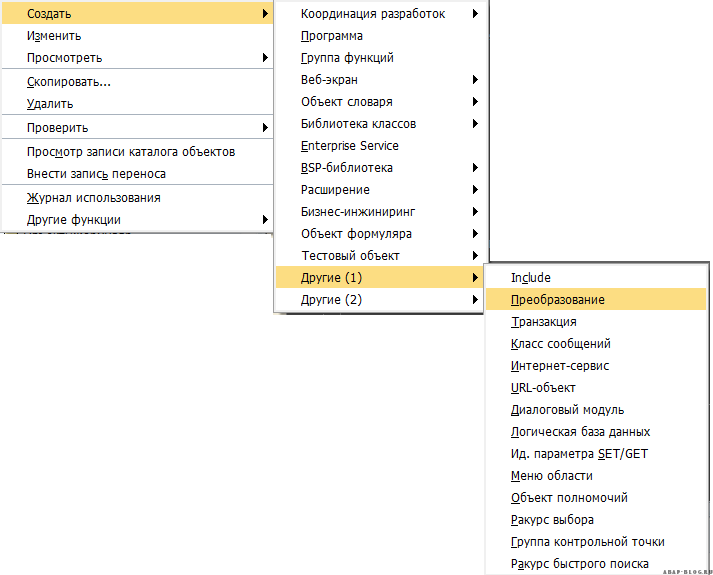
      </line>

    </lines>

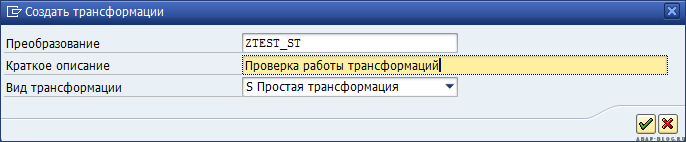
  </tt:template>

</tt:transform>

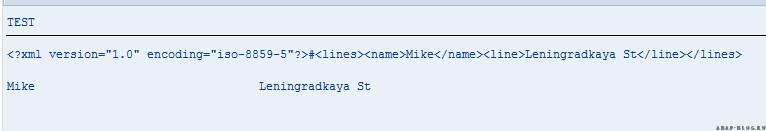
Создать трансформацию можно нажав правой кнопкой на пакете разработки, выбрав соответствующий пункт меню

[](http://abap-blog.ru/wp-content/uploads/2013/04/st_1.png)

В появившемся диалоге выбрать тип трансформации (S – простая трансформация)

[](http://abap-blog.ru/wp-content/uploads/2013/04/st_2.png)

Результат работы программы

[](http://abap-blog.ru/wp-content/uploads/2013/04/st_3.png)

### **Определение параметров**

ST программы могут содержать параметры которые передаются в оператор CALL TRANSFORMATION и могут использоваться для определения логики поведения. Синтаксис их определения следующий

<tt:parameter name="..." [ref-type="..."]

                         [kind="..."]

                         [[s-val="..."][d-val="..."]]|[val="..."] />

Через атрибут name определяется имя параметра, которое не должно превышать 30 символов и должно быть уникально среди имен в корневых узлах, параметрах, переменных. Имя не чувствительно к регистру. Параметры определяются как на уровне главного шаблона, так и остальных используемых шаблонов, при этом параметры, определенные в одном шаблоне не могут быть доступны в другом.

Параметры могут быть использованы для вызова вложенных шаблонов (tt:apply) или других ST программ (tt:call), при указании атрибута with-parameter.

Атрибут kind определяет направление параметра, может принимать значения «in» — входящий, «out» — исходящий, «in/out» — входящий и исходящий. Если явно не задать направление, по умолчанию такой параметр будет входящим «in».

Параметры s-val, d-val, val – присваивают значение для параметра по умолчанию, могут быть выполнены следующие присвоения (приставки s- и d- тут и в других атрибутах указывают на процесс сериализации и десериализации)

|  |  |
| --- | --- |
| **Тип данных** | **Представление** |
| Floating point numbers (ABAP type f) | F(‘value’) |
| Byte strings (ABAP types x and xstring) | X(‘value’) |
| Date fields (ABAP type d) | D(‘value’) |
| Decimal floating point numbers (ABAP types decfloat16, decfloat34) | DECFLOAT16(‘value’)DECFLOAT34(‘value’) |
| Packed numbers (ABAP type p) | P(value) |
| Integers (ABAP type i) | I(value) | value |
| Numeric text fields (ABAP type n) | N(‘value’) |
| Character strings (ABAP types c and string) | C(‘value’) | ‘value’ |
| Time fields (ABAP type t) | T(‘value’) |
| Initial value (all ABAP types) | initial |

С помощью атрибута ref-type определяется атрибут, имеющий ссылочный тип, т.е. ссылка на глобальный ABAP класс или интерфейс, а так же на ссылочную переменную (REF TO DATA). При этом имена классов и интерфейсов указываются напрямую без указания пространства имен, а ссылочная переменная должна быть заранее определена через тип данных в команде tt:type, где нужно указать пространство имен. Параметры классов (интерфейсов) используются обычно для создания объектов или вызова методов, а ссылочные переменные как атрибуты в передаваемые методы. Получить значение параметра, можно используя команду tt:assign.

Примеры объявления параметров

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:parameter kind="in" name="PARA1" val="4"/>

  <tt:parameter kind="out" name="PARA2"/>

  <tt:parameter kind="in/out" name="PARA3"/>

  <tt:template>

    <tt:assign to-var="PARA2" var="PARA3"/>

    <tt:assign to-var="PARA3" var="PARA1"/>

    <tt:assign to-var="PARA1" val="555"/>

  </tt:template>

</tt:transform>

Как параметры при вызове других трансформаций

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:variable name="VARI1" val="1"/>

  <tt:variable name="VARI2" val="2"/>

  <tt:variable name="VARI3" val="3"/>

  <tt:template>

    <tt:call transformation="...">

      <tt:with-parameter name="PARA1" var="VARI1"/>

      <tt:with-parameter name="PARA2" var="VARI2"/>

      <tt:with-parameter name="PARA3" var="VARI3"/>

    </tt:call>

    <X1>

      <tt:write var="VARI1"/>

    </X1>

    <X2>

      <tt:write var="VARI2"/>

    </X2>

    <X3>

      <tt:write var="VARI3"/>

    </X3>

  </tt:template>

</tt:transform>

### **Определение переменных**

Переменные в ST программах объявляются командой tt:variable, которая имеет следующий синтаксис

<tt:variable name="..." [ref-type="..."]

                        [[s-val="..."][d-val="..."]]|[val="..."] />

Атрибуты аналогичны описанным выше атрибутам для параметров. К переменным можно получать прямой доступ в шаблонах, но переменные, объявленные в одном шаблоне, не могут быть получены в другом. В случае необходимости их значения могут быть переданы при вызове шаблона.  В отличие от корневых узлов, переменные не связаны с ABAP данными. Для получения значения переменных используется команда tt:assign.

### **Определение типов**

Корневые узлы, используемые в ST программах, могут, но не обязаны иметь присвоение типа. Обычно для этого используются типы из словаря или глобальных классов. Кроме того ST программы позволяют определять свои собственные типы данных, для этого служит команда: tt:type. Объявление типа происходит вне шаблонов ST программы. Команда tt:type имеет следующий синтаксис

<tt:type name="..." [[line-]type="..." [length="..."] [decimals="..."]]

  [extensible=extensible\_flag]>

  [<tt:front>

     <tt:node name="..." [[line-]type="..." [...]]

       [extensible=extensible\_flag]>

       [...]

     </tt:node>

     ...

   </tt:front>]

  [<tt:node name="..." [[line-]type="..." [...]]

     [extensible=extensible\_flag]>

     [...]

   </tt:node>

  ...]

</tt:type>

Атрибут name определяет имя типа. Вы можете определять как элементарные, структурные, так и табличные типы данных, которые могут быть частично или полностью родовыми. Кроме того возможна ссылка на ранее определенный локально в ST программе или глобальный тип данных через атрибут type.

При определении ссылки через атрибут [line-]type можно использовать следующие пространства имен

* http://www.sap.com/abapxml/types/dictionary если ссылаетесь на словарь.
* http://www.sap.com/abapxml/types/type-pool/*name* если ссылаетесь на пул типов.
* http://www.sap.com/abapxml/types/class-pool/*class* если ссылаетесь на глобальный атрибут класса.
* http://www.sap.com/abapxml/types/defined/*name* если ссылаетесь на ранее определенный в текущей ST программе тип tt:type.

Имя типа необходимо указывать в верхнем регистре. Пример p:NAME, где p – определенное заранее, пространство имен.

**Определение элементарных типов**

При определении элементарного типа данных, атрибуту type должно быть присвоено одно из следующих значений: C, D, F, I, N, P, STRING, T, X или, XSTRING (в верхнем регистре).

Для типов C, N, P, X может быть указан атрибут length определяющий длину, а для типа P атрибут decimals указывающий на кол-во знаков после запятой. Если атрибут decimals указан без длинны, то он игнорируется. Если оба атрибута пропущены, их значения берутся как значения по умолчанию для родового типа.

Если для типа C не указать длину, это означает, что тип может иметь любую длину, как для string или csequence. Тоже самое справедливо и для типа X.

### **Определение структурных типов**

Для определения структурных типов используется команда tt:node. Вложенная команда tt:node определяет элемент структуры с именем определенным в атрибуте name и типом type. Команда tt:node вложенная в другую команду tt:node определяет подструктуру. Элемент tt:type или tt:node,  который имеет вложенный элемент tt:node не должен иметь ссылку на тип в атрибуте type, вместо этого должно стоять значение «?». Порядок, в котором компоненты расположены в структуре, не имеет значения при связывании с ABAP структурой, кроме того некоторые поля могут отсутствовать в указании типа (см. атрибут extensible). Для определения порядка, в котором будут расположены компоненты структуры можно воспользоваться элементом tt:front, данный элемент может быть задан только в единственном числе для элементов tt:node и tt:type.

Для определения ограничений на тип структуры используется атрибут extensible. Может иметь следующие значения

* «on» — указывает на то, что текущий элемент может иметь вложенные элементы, явно не определенные в типе.
* «deep» — указывает на то, что как текущий, так и его дочерние элементы могут иметь вложенные элементы, явно не определенные. Данное значение установлено по умолчанию.
* «off» — текущий элемент не должен иметь других элементов, явно не описанных в типе.
* «deep-off» — как текущий, так и его дочерние элементы не должны иметь элементы, явно не описанные в типе.

### **Определение табличных типов**

Если для элемента tt:node или tt:type задан атрибут line-type, значит, данный элемент представляет собой внутреннюю таблицу с типом указанным в line-type. Если вы хотите объявить внутреннюю таблицу с указанным через команду tt:node структурным типом, для элемента верхнего уровня в качестве типа необходимо указать «?»

<tt:node line-type=”?”>

  <tt:node type=”C”/>

</tt:node>

Примеры объявления типов

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates"

xmlns:ddic="http://www.sap.com/abapxml/types/dictionary"

xmlns:def="http://www.sap.com/abapxml/types/defined" >

  <tt:type name="T1">

    <tt:node name="C" line-type="?">

      <tt:node name="C1" type="ddic:DDT1"/>

      <tt:node name="C2" line-type="def:T2"/>

    </tt:node>

  </tt:type>

  <tt:type name="T2">

    <tt:node name="D">

      <tt:node name="D1" line-type="?"/>

    </tt:node>

  </tt:type>

### **Определение ссылочных типов**

Напрямую нельзя задать тип ссылки, но можно задать ссылку на ссылочную переменную из словаря. Например, на какой-нибудь класс, затем при объявлении параметров (переменных) указать на созданный тип со ссылкой на класс, после чего можно будет создать его и вызывать методы (см. ниже)

## 

## Адресация ABAP данных в ST программах

В ST программах все объекты данных представляют собой древовидную структуру.  Доступ к элементам (если они есть) в узлах осуществляется через точку. Каждый корневой узел данных (tt:root) имеет свою древовидную структуру. К данным можно обращаться либо указывая имя узла в ссылке, либо через текущий узел. Текущим узлом можно установить любой корневой узел, но текущим, в один момент времени, может быть только один узел.

Рассмотрим пример

TYPES:

  BEGIN OF ty\_root\_one,

    X TYPE i,

  END OF ty\_root\_one,

  BEGIN OF ty\_root\_two,

    Y TYPE i,

  END OF ty\_root\_two.

DATA:

  ls\_one TYPE ty\_root\_one,

  ls\_two TYPE ty\_root\_two,

  lv\_xml TYPE string.

ls\_one-x = 1.

ls\_two-y = 2.

CALL TRANSFORMATION ZTEST\_ST

  SOURCE root1 = ls\_one

         root2 = ls\_two

  RESULT XML lv\_xml.

WRITE lv\_xml.

Трансформация

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:root name="ROOT2"/>

  <tt:template>

    <roots>

      <root1>

        <x><tt:value ref="ROOT1.X"/></x>

      </root1>

      <root2>

        <x><tt:value ref="ROOT2.Y"/></x>

      </root2>

    </roots>

  </tt:template>

</tt:transform>

Для того чтобы не указывать имя узла каждый раз, мы можем определить текущий узел. Для установки текущего узла можно воспользоваться командой tt:ref

<tt:ref name="node">..</ref>

где name – имя узла. Приведенная выше трансформация может быть преобразована в следующий вид

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:root name="ROOT2"/>

  <tt:template>

    <roots>

      <root1>

        <tt:ref name="ROOT1">

          <x><tt:value ref="X"/></x>

        </tt:ref>

      </root1>

      <root2>

        <tt:ref name="ROOT2">

          <x><tt:value ref="Y"/></x>

        </tt:ref>

      </root2>

    </roots>

  </tt:template>

</tt:transform>

Для многих ST команд текущий узел может быть задан через атрибут ref.

Пример

<tt:instruction ref="node">

...

</tt:instruction>

Где instruction – имя ST команды. Кроме того текущий узел можно установить в ST атрибуте для текстового XML элемента, тогда наша трансформация будет иметь следующий вид

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:root name="ROOT2"/>

  <tt:template>

    <roots>

      <root1 tt:ref="ROOT1">

        <x><tt:value ref="X"/></x>

      </root1>

      <root2 tt:ref="ROOT2">

        <x><tt:value ref="Y"/></x>

      </root2>

    </roots>

  </tt:template>

</tt:transform>

Для текстовых элементов допускается указание значения через атрибут tt:value-ref, тогда трансформация примет следующий вид

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:root name="ROOT2"/>

  <tt:template>

    <roots>

      <root1>

        <x tt:value-ref="ROOT1.X"/>

      </root1>

      <root2>

        <x tt:value-ref="ROOT2.Y"/>

      </root2>

    </roots>

  </tt:template>

</tt:transform>

Если определен текущий узел, и доступ получаете через указание имени корневого элемента, необходимо указывать точку

<root2 tt:ref="ROOT2">

<x><tt:value ref=".ROOT2.Y"/></x>

</root2>

Кроме того во всех ST командах, которые позволяют явно задать текущий узел, к заданному узлу можно обратиться через константу $ref, при этом если текущий узел не задан, обращаться через $ref нельзя

<root2 tt:ref="ROOT2">

<x><tt:value ref="$ref.Y"/></x>

</root2>

Существует следующее ограничение - если первый символ узла не является буквой или подчеркиванием или в имени встречаются символы кроме букв, чисел, дефисов и подчеркиваний, доступ к такому узлу необходимо получать с помощью оператора ref(‘ИмяУзла’).

## Контроль выполнения сериализации и десериализации

Выполнение ST программы начинается с обработки главного шаблона программы. Внутри данного шаблона простой текст обрабатывается вместе с ST командами, после чего происходит либо сериализация, либо десериализация.

В процессе сериализации, текст введенный в ST программу переносится в создающийся XML документ без каких-либо изменений, выполняются ST команды, встречающиеся в этом тексте.

В процессе десериализации исходный XML документ рассматривается как входящий поток и сравнивается элемент за элементом с введенным шаблоном в ST программе. Если имена элементов (атрибутов) совпадают и располагаются в том же месте ST программы, они обрабатываются. Существует возможность пропуска сравнения определенных элементов и другие операции, которые позволяют  повлиять на логику десериализации.

### **Определение текстового шаблона**

В качестве текста шаблона могут быть представлены либо XML элементы с атрибутами, либо просто текст.

XML элементы и атрибуты определяются по обычным правилам построения XML документа, но не должны лежать в пространстве имен ST команд (без префикса tt:).  Используется следующий синтаксис

<element [attr] [tt:ref=″node″>]

                [tt:lax=lax\_flag]

                [tt:extensible=extensible\_flag]>

  ...

</element>

Имя XML элемента может быть любым в рамках правил XML. Тут же могут быть статически прописаны атрибуты и их значения [attr].

Атрибут tt:ref устанавливает ссылку на текущий узел (см. выше).

Атрибут tt:lax определяет правило, по которому реагирует интерпретатор на несоответствие XML документа при десериализации относительно имён XML элементов и наличия атрибутов. Принимает следующие значения (по умолчанию deep-off)

* «off» — XML атрибут во входящем потоке должен быть с тем же именем и наличием тех же атрибутов с такими же значениями что в программе.
* «on» — XML элемент во входящем потоке может иметь любое имя, но при этом атрибуты и их содержимое должно совпадать.
* «deep-on», «deep-off» — устанавливают те же правила, включая вложенные элементы.

Для всех вложенных элементов, значение атрибута может быть перезаписано.

Атрибут tt:extensible определяет правило, по которому реагирует интерпретатор на несоответствие вложенности и наличия других XML элементов, не описанных статически в ST программе. Может принимать следующие значения (По умолчанию «on»)

* «on» — элемент XML может иметь вложенные элементы, явно не определенные в ST программе.
* «deep-static» и «deep-dynamic» — текущий элемент и его вложенные элементы могут  иметь неопределенные вложенные элементы во входящем XML потоке.
* «off» — запрещает расширение для текущего элемента, но не для его вложенных элементов.
* «deep-off» — запрещает расширение для текущего элемента и его вложенных элементов.

Для всех вложенных элементов, значение атрибута может быть перезаписано. Разница между deep-static и deep-dynamic в том, что они имеют разные области действия. Deep-static используется только в текущем шаблоне, deep-dynamic выполняется для всех шаблонов и вызываемых ST программ.

Если необходимо задать атрибут для XML элемента динамически, делается это с помощью команды tt:attribute

<tt:attribute name=″attr″ [ref=″node″]>

  ...

</tt:attribute>

Где name – имя атрибута, ref – ссылка на текущий узел.  Значение передается командой tt:value, но можно так же использовать и короткую форму используя ST атрибут value-ref

<root2>

<tt:attribute name="attribute" value-ref="ROOT2.Y" />

</root2>

### **Вставка текста**

Текстом называется часть шаблона, которая не является элементом XML документа. Текст в шаблон может быть вставлен двумя способами, либо просто как текст

…>*text*<…

Либо через ST команду

…><tt:text>*text*</tt:text><…

В первом случае текст будет вставлен в документ XML, но проигнорирован в случае наличия в нем только пробелов. Во втором случае такой текст не игнорируется.

**Сериализация -** если текст не проигнорирован, все символы текста вставляются в создающийся XML документ, включая пробелы. Если текст был проигнорирован, ничего не попадает в XML.

**Десериализация-** Текст из XML потока посимвольно сравнивается с текстом из шаблона (включая пробелы и разрывы строк), если сравнение будет неудачным, система вызовет исключение CX\_ST\_MATCH\_ELEMENT**.** Тексты могут быть пропущены командой tt:skip.

### **Трансформация ABAP данных**

Для обработки ABAP данных служат следующие команды

* tt:value – для элементарных типов данных и компонентов структур.
* tt:loop – для внутренних таблиц.
* tt:copy – для работы с объектами данных целиком вместе с компонентами.

#### **Обработка элементарных типов и компонентов структур**

Для сериализации и десериализации элементарных типов данных используется команда tt:value

<tt:value [ref="node"] [map="mapping\_list"]

                       [length|minLength|maxLength="length"]

                        [validation] />

Атрибут ref – определяет текущий узел. Узел должен быть элементарного типа, внутренние таблицы и структуры обрабатываются через команду tt:copy.   Элементарные типы отображаются в XML и обратно согласно заранее определенному asXML формату. Формат определяет соответствие элементарных типов ABAP и типов данных языка описания XML ([XML schema](http://www.w3.org/TR/xmlschema-2/#built-in-datatypes)). Следующая таблица показывает пример соответствия для этих преобразований (остальное можно найти в документации по asXML)

|  |  |  |  |
| --- | --- | --- | --- |
| **ABAP тип** | **ABAP представление** | **XML Scheme тип** | **XML представление** |
| b | 123 | xsd:unsignedByte | 123 |
| s | -123 | xsd:short | -123 |
| i | -123 | xsd:int | -123 |
| p | -1.23 | xsd:decimal | -1.23 |
| decfloat16 | 123E+1 | precisionDecimal, totalDigits = 16 | 1.23E+3 |
| decfloat34 | -3.140…0E+02 | precisionDecimal, totalDigits = 34 | -314.0…0 |
| f | -3.140…0E+02 | xsd:double | -3.14E2 |

Кроме того для отдельных доменов из ABAP словаря существуют свои правила преобразования

|  |  |  |  |
| --- | --- | --- | --- |
| **Домент** | **ABAP представление** | **XML Schema тип** | **XML представление** |
| XSDBOOLEAN | «X», » « | boolean | «true», «false» |
| XSDDATE\_D | «20071001» | date | «2007-10-01» |
| XSDDATETIME\_Z, XSDDATETIME\_LONG\_Z, XSDDATETIME\_OFFSET, XSDDATETIME\_LOCAL | «20060727170334» | dateTime | «2006-07-27T17:03:34Z» |
| XSDLANGUAGE | «E», «D» | language | «EN», «DE» |
| XSDQNAME | «{URI}name» | QName | prefix:name |
| XSDTIME\_T | «115500» | time | «11:55:00» |
| XSDUUID\_RAW, XSDUUID\_CHAR | «123456781234ABCDEF12123456789012» | UUID | «12345678-1234-abcd-ef12-123456789012 |

С помощью атрибута map можно задать правила для замены одного или нескольких значений другими. Правила задаются через запятую и с соблюдением следующего синтаксиса

* val(a1…an) > xml(x) – срабатывает при сериализации, указывает на то, что значения a1..an необходимо преобразовать в X.
* xml(x1…xn) > val(a) – срабатывает при десериализации и указывает на то, что значения x1..xn необходимо преобразовать в A.
* xml(x) = val(a) или val(a) = xml(x) – срабатывает как при сериализации так и при десериализации.

Пример, программа при сериализации заменяет все значения Woman и Man на Person

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:template>

      <X1>

        <tt:value  ref="ROOT1"

                   map="val(C('Woman'), C('Man')) > xml('Person')" />

      </X1>

  </tt:template>

</tt:transform>

С помощью атрибутов length, minLength, maxLength задается ограничение на длину значения при сериализации и десериализации.

В блоке validation для команд tt:value, tt:read и tt:write устанавливаются ограничения связанные с XML Schema типами и их сериализацией и десериализацией.

Указываются следующие атрибуты

… xsd-type=»type»

[xsd-maxInclusive=»max»]

[xsd-maxExclusive =»max»]

[xsd-minInclusive=»min»]

[xsd-minExclusive =»min»]

[xsd-totalDigits=»dgts»]

[xsd-fractionDigits=»dgts»] …

Атрибут xsd-type указывает на тип данных XML Schema. Значение для сериализации и десериализации должно входить в диапазон указанного типа, в противном случае система вызовет исключение CX\_ST\_SERIALIZATION\_ERROR при сериализации и CX\_ST\_DESERIALIZATION\_ERROR при десериализации.

Кроме указания типа задаются следующие атрибуты

* xsd-maxInclusive – максимально допустимое значение, включая указанное, xsd-maxExclusive – максимально допустимое число не включая указанное.
* xsd-minInclusive – минимально допустимое число, включая указанное, xsd-minExclusive – минимально допустимое число не включая указанное.
* xsd-totalDigits, fractionDigits – максимальное кол-во число цифр до и после запятой.

Пример трансформации

<tt:transform

  xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="NUM"/>

  <tt:template>

    <Number>

      <tt:value ref="NUM"

          xsd-type="short" xsd-minInclusive="30000"/>

    </Number>

  </tt:template>

</tt:transform>

Программа вызова

PARAMETERS int TYPE i.

DATA xml\_string TYPE string.

TRY.

    CALL TRANSFORMATION ...

         SOURCE num = int

         RESULT XML xml\_string.

    WRITE / 'OK'.

  CATCH cx\_st\_error.

    WRITE / 'Not OK'.

ENDTRY.

#### **Ограничение на длину значений**

Вы можете использовать атрибуты minLength, maxLength и length для определения длинны в таких командах как: tt:value, tt:read, tt:write. Параметры определяют ограничения на значения, передаваемые при сериализации и десериализации.

В качестве значения может быть передано положительное целое число. Данные атрибуты можно задать для узлов данных или переменных с типами: с, x, string и xstring. Для всех остальных типов данные ограничения игнорируются.

**Сериализация -** атрибуты length или minLength определяют количество символов или байт (x), которые будут переданы в итоговый XML документ. Если будет передано значение с меньшей реальной длинной, недостающие до length символы будут заполнены справа либо пробелами, либо нулевым байтом (в зависимости от типа). Атрибуты length или maxLength определяют максимально возможное число символов или байт, которые могут быть переданы в XML документ. Если будет передано больше символов, возникнет исключение CX\_SY\_CONVERSION\_DATA\_LOSS (если только оставшиеся символы не состоят из пробелов или нулевых байтов).

**Десериализация -** атрибут minLength игнорируется, атрибуты length и maxLength задают максимальное число символов или байтов, которое ожидается для получения из XML документа. Если оно содержит большее количество символов и эти символы не являются пробелами и  нулевыми байтами, система вызовет исключение CX\_ST\_CONSTRAINT\_ERROR. Причем данное исключение не может быть обработано при вызове CALL TRANSFORMATION.

#### **Обработка таблиц**

Сериализация и десериализация внутренних таблиц  происходит в цикле, где на каждой итерации текущим узлом задается строка таблицы. Для обработки таблиц служит команда tt:loop

<tt:loop [ref=“node“] [name=“alias“]>

  ...

</tt:loop>

Атрибут ref – как и в других командах, указывает на узел данных, который содержит внутреннюю таблицу, name – задает псевдоним для доступа к текущей строке внутренней таблицы.

Пример трансформации с обработкой внутренней таблицы

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT"/>

  <tt:template>

    <tab1>

      <tt:loop ref=".ROOT" name="line">

        <key>

          <tt:value ref="$line.key" />

        </key>

        <tab2>

          <tt:loop ref="$line.values">

            <value>

              <tt:value />

            </value>

          </tt:loop>

        </tab2>

      </tt:loop>

    </tab1>

  </tt:template>

</tt:transform>

Программа

DATA xml\_string TYPE string.

DATA: BEGIN OF line,

        key TYPE i,

        values TYPE TABLE OF i,

      END OF line.

DATA num TYPE i.

DATA itab LIKE TABLE OF line.

DATA result LIKE itab.

DO 3 TIMES.

  CLEAR line.

  line-key = sy-index + 1.

  num = line-key \*\* 2.

  APPEND num TO line-values.

  num = line-key \*\* 3.

  APPEND num TO line-values.

  num = line-key \*\* 4.

  APPEND num TO line-values.

  APPEND line TO itab.

ENDDO.

CALL TRANSFORMATION ...

  SOURCE root = itab

  RESULT XML xml\_string.

cl\_abap\_browser=>show\_xml( EXPORTING xml\_string = xml\_string ).

CALL TRANSFORMATION ...

  SOURCE XML xml\_string

  RESULT root = result.

Возвращаемый XML

<tab1>

  <key>2</key>

  <tab2>

    <value>4</value>

    <value>8</value>

    <value>16</value>

  </tab2>

  <key>3</key>

  <tab2>

    <value>9</value>

    <value>27</value>

    <value>81</value>

  </tab2>

  <key>4</key>

  <tab2>

    <value>16</value>

    <value>64</value>

    <value>256</value>

  </tab2>

</tab1>

#### **Комплексная обработка ABAP данных**

С помощью команды tt:copy можно провести сериализацию и десериализацию для всех типов данных, кроме ссылочных переменных. Данная команда запускает XSL трансформацию и преобразует данные в формат asXML (аналогично стандартной трансформации ID). Так, например, мы можем вывести целиком всю структуру без указания всех её компонентов, либо внутреннюю таблицу, если нам не нужно дополнительной обработки этих компонентов. Пример трансформации к предыдущей программе

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT"/>

  <tt:template>

    <node>

      <tt:copy  ref="ROOT" />

    </node>

  </tt:template>

</tt:transform>

Результатом обработки будет следующий XML

<node>

  <item>

    <KEY>2</KEY>

    <VALUES>

      <item>4</item>

      <item>8</item>

      <item>16</item>

    </VALUES>

  </item>

  <item>

    <KEY>3</KEY>

    <VALUES>

      <item>9</item>

      <item>27</item>

      <item>81</item>

    </VALUES>

  </item>

  <item>

    <KEY>4</KEY>

    <VALUES>

      <item>16</item>

      <item>64</item>

      <item>256</item>

    </VALUES>

  </item>

</node>

### **Присвоение значений**

Вы можете использовать команду tt:assign для присвоения значений узлам данных, параметрам и переменным. Команда имеет следующий синтаксис

<tt:assign [to-ref="node"|to-var="variable"]

           [ref="node"|val="value"|var="variable"] />

Атрибуты to-ref и to-var указывают куда передать значение, ref, val и var – какое значение передавать. В качестве полей источников и полей назначения могут выступать

* Узлы данных (ref), следует помнить, что их содержимое не может быть изменено при сериализации и считано во время десериализации. Нельзя записать из одного узла в другой.
* Переменные и параметры (var).
* Так же можно передавать значения напрямую (константы) через атрибут val.

В случае необходимости можно использовать команду по преобразованию типов tt:cast. Для работы с внутренними таблицами можно использовать следующий синтаксис

<tt:assign [to-ref="itab"]>

  <tt:assign [to-ref="comp"]

             [val="value"|var="variable"] />

  ...

</tt:assign>

В данном случае в таблицу itab добавляется новая строка и компоненту comp присваивается значение.

Для очистки значений переменных, узлов данных и параметров служит команда tt:clear

<tt:clear [ref="node"|var="variable"] />

Где можно указать либо имя узла через атрибут ref, либо переменную или параметр через атрибут var.

**Сериализация -** во время сериализации установка значений может быть выполнена только для параметров и переменных, если в качестве назначения указан узел данных или неявно текущий узел, данное присвоение игнорируется.

**Десериализация -** во время десериализации в качестве источника данных могут выступать только переменные (параметры) и значения, если указать в качестве источника ссылку на узел данных, данное присвоение будет проигнорировано.

### **Вывод значений переменных (параметров)**

Во время сериализации можно вывести значения переменных и параметров в XML документ, делается это с помощью команды tt:write

<tt:write var="variable" [map="mapping\_list"]

                         [length|minLength=″length″]

                         [validation] />

Где var – имя переменной или параметра, map – список с правилами замены (см. выше), Length, minLength – ограничения вывода по длине.

Пример использования команды

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:variable name="VARI" val="333" />

  <tt:template>

    <X>

      <tt:write var="VARI"/>

    </X>

  </tt:template>

</tt:transform>

### **Чтение значений XML элементов**

Во время выполнения десериализации с помощью команды tt:read можно получить значение XML атрибута, синтаксис команды следующий

<tt:read var="variable" type="type"

                         [length="len"]

                         [decimals="dec"]

                         [map="mapping list"]

                         [maxLength="length"]

        [validation] />

Атрибут variable – указывает на имя переменной или параметра, куда необходимо записать считанное значение. Length, maxLength – ограничение по длине, в данном случае обозначают одно и тоже. Decimals – кол-во знаков после запятой, map – список правил присвоения. Type – тип данных. Команда поддерживает считывание только родовых типов.

Пример использования команды

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT"/>

  <tt:variable name="VARI"/>

  <tt:template>

    <X>

      <tt:read type="I" var="VARI"/>

    </X>

    <tt:assign to-ref="ROOT" var="VARI"/>

  </tt:template>

</tt:transform>

## Управление ходом выполнения

### ***Определение направления***

С помощью команд <tt:serialize> и <tt:deserialize> можно определить, какие команды будут выполнены при сериализации, а какие при десериализации. Синтаксис следующий

<tt:serialize>

  ...

</tt:serialize>

<tt:deserialize>

  ...

</tt:deserialize>

Пример трансформации

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:root name="ROOT2"/>

  <tt:template>

    <X>

      <tt:serialize>

        <Y>

          <tt:value ref=".ROOT1.COL1" />

        </Y>

        <Y>

          <tt:value ref=".ROOT1.COL2" />

        </Y>

        <Y>

          <tt:value ref=".ROOT1.COL3" />

        </Y>

      </tt:serialize>

      <tt:deserialize>

        <tt:loop ref=".ROOT2">

          <Y>

            <tt:value />

          </Y>

        </tt:loop>

      </tt:deserialize>

    </X>

  </tt:template>

</tt:transform>

#### **Пропуск элементов при десериализации**

В том случае, когда необходимо пропустить тот или иной элемент XML документа можно воспользоваться командой tt:skip

<tt:skip [name="name"] [count="cnt"]/>

При десериализации данная команда имеет следующий эффект

* Если не заданы атрибуты, пропускается текущий элемент в XML потоке.
* Если атрибуты установлены, пропускается элемент с именем name, пропуск будет cnt раз, при этом, если значение будет равным «\*», элемент будет полностью проигнорирован при десериализации.
* Если имя не установлено, будет пропущено cnt следующих элементов.

Пример трансформации

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:template>

    <roots>

      <root1>

        <x tt:value-ref="ROOT1.X" />

        <tt:skip name="Y" count="\*"/>

        <z tt:value-ref="ROOT1.Z" />

      </root1>

    </roots>

  </tt:template>

</tt:transform>

Программа

TYPES:

  BEGIN OF ty\_root\_one,

    x TYPE i,

    y TYPE i,

    z TYPE i,

  END OF ty\_root\_one.

DATA:

  ls\_one TYPE ty\_root\_one,

  lv\_xml TYPE string.

ls\_one-x = 1.

ls\_one-y = 2.

ls\_one-z = 3.

CALL TRANSFORMATION ZTEST\_ST

  SOURCE root1 = ls\_one

  RESULT XML lv\_xml.

WRITE lv\_xml.

CALL TRANSFORMATION ZTEST\_ST

   SOURCE XML lv\_xml

   RESULT root1 = ls\_one.

#### **Условные трансформации**

Позволяют отделить набор ST команд и выполнить их только при определенных условиях. Используются как отдельно, так и внутри таких команд как tt:switch и tt:group. Синтаксис выглядит следующим образом

<tt:[s-|d-]cond[-var] [using="precond"] [data="assertion"] [[s-|d-]check="cond"]>

  ...

</tt:[s-|d-]cond>

Префикс s- и d- используются тогда, когда условия должны обрабатываться только для сериализации или только для десериализации. Префикс –var используется если условная трансформация применяется относительно переменных или параметров.

**Определение предусловий**

Атрибут using определяет предусловия, могут быть следующими

|  |  |
| --- | --- |
| **Предусловие** | **Обозначение** |
| exist() | Срабатывает когда указанный узел существует |
| type-C() | Срабатывает если узел типа C |
| type-D() | Срабатывает если узел типа D |
| type-F() | Срабатывает если узел типа F |
| type-I() | Срабатывает если узел типа I |
| type-N() | Срабатывает если узел типа N |
| type-P() | Срабатывает если узел типа P |
| type-T() | Срабатывает если узел типа T |
| type-X() | Срабатывает если узел типа X |

Пример

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT"/>

  <tt:template>

    <tt:s-cond using="type-C(ref('ROOT'))">

      <X>

        ...

      </X>

    </tt:s-cond>

  </tt:template>

</tt:transform>

Указанное условие можно записать следующим образом: «type-C(ROOT)».  Через запятую можно указывать несколько условий

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:root name="ROOT2"/>

  <tt:template>

    <tt:s-cond using="type-I(ROOT1), type-F(ROOT2)">

      <X>

        ...

      </X>

    </tt:s-cond>

  </tt:template>

</tt:transform>

**Определение утверждений**

Оператор data определяет утверждения, если утверждение истинно выполняется блок ST команд. В противном случае вызывается исключение CX\_ST\_COND\_CHECK\_FAIL. В утверждениях могут быть использованы как узлы данных, так и переменные и просто значения

* Узлы данных, как и во всех других условиях, записываются в форме ref(‘ИмяУзла’), либо просто ИмяУзла.
* Переменные и параметры var(ИмяПеременной)
* Утверждения бывают двух видов:
* Сравнение со значением по умолчанию — initial(..)
* Сравнение со значением: ref(‘ИмяУзла’) = 22

В следующем примере, если значение ROOT1 не будет равно ABC, система вызовет исключение

<tt:template>

    <ROOTS>

      <ROOT>

        <tt:cond data="ROOT1='ABC'">

          <tt:value ref="ROOT1"/>

        </tt:cond>

      </ROOT>

    </ROOTS>

  </tt:template>

**Определение проверок**

Атрибут **check** позволяет задавать условия относительно значений переменных, параметров и узлов данных. Приставка –var (var-check) в данном атрибуте используется для условий относительно переменных и параметров.

Через условия можно определить состояние значений в переменных, параметрах, узлах

* exist(‘ИмяУзла’) – выполняется, если указанный узел существует.
* Initial(ref(‘ИмяУзла’) или ИмяПеременной) – выполняется если узел или переменная (параметр) имеют пустое значение
* Not-initial(..) – противоположность initial.

Пример

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT"/>

  <tt:template>

    <tt:s-cond check="not-initial(ROOT)">

      <X>

        <tt:value ref="ROOT" />

      </X>

    </tt:s-cond>

  </tt:template>

</tt:transform>

В атрибуте check так же можно задавать условия сравнения синтаксис следующий

* Ref(‘ИмяУзла’)|var(ИмяПеременной)|Значение **Оператор**
* Ref(‘ИмяУзла’)|var(ИмяПеременной)|Значение

Существуют следующие операторы

|  |  |
| --- | --- |
| Оператор | Значение |
| = | Условие выполняется, если оба операнда равны |
| != | Условие выполняется, если значения операндов не равны |
| >, &gt; | Значение выполняется если значение левого операнда больше правого |
| >=, &gt;= | Значение выполняется если значение левого операнда больше или равно правого |
| &lt; | Значение выполняется если правый операнд больше |
| &lt;= | Значение выполняется если правый операнд меньше или равен левому |

Пример использования

<tt:cond check="ref('ROOT1.X') >= 10">

  <X tt:value-ref="ROOT1.X"/>

</tt:cond>

<tt:cond check="ref('ROOT1.X') &lt;= 10">

  <Y tt:value-ref="ROOT1.X"/>

</tt:cond>

Условия в атрибуте check можно объединять через оператор and или or, так же доступны скобки и отрицание, через оператор not. Пример

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:root name="ROOT2"/>

  <tt:root name="ROOT3"/>

  <tt:template>

    <tt:ref name="ROOT1">

      <tt:s-cond check="($ref > ref('.ROOT2')) and ($ref < ref('.ROOT3'))">

        <X>

          <tt:value/>

        </X>

      </tt:s-cond>

    </tt:ref>

  </tt:template>

</tt:transform>

Если содержимое tt:[s-|d-]cond не является шаблоном, должен быть указан хотя бы один атрибут: check, data, using.

**Определение шаблона**

Когда условная трансформация десериализуется, содержимое шаблона, находящееся между <tt:cond> .. </tt:cond> используется как критерий присвоения.  Шаблон содержит одну или несколько конструкций используемых как маркеры. Наиболее распространенная форма маркера – текстовый XML элемент, если содержимое шаблона состоит из одного XML элемента, с тем же именем что и обрабатываемый элемент из XML потока, то условие считается положительным и XML элемент обрабатывается. Если в шаблоне несколько маркеров, то условие обрабатывается, если все маркеры были обработаны.

Допустим, имеем следующий XML документ

<roots>

  <root1>

    <Z>3</Z>

    <X>8</X>

    <Y>2</Y>

  </root1>

</roots>

Трансформация, использующаяся при десериализации

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:template>

    <roots>

      <root1>

        <tt:cond>

          <Z tt:value-ref="ROOT1.Z"/>

          <X tt:value-ref="ROOT1.X"/>

        </tt:cond>

        <tt:skip count="2"/>

        <Y tt:value-ref="ROOT1.Y"/>

      </root1>

    </roots>

  </tt:template>

</tt:transform>

Шаблон обработается успешно, если во входящем потоке будут последовательно найдены элемент Z и элемент X, если их порядок в шаблоне будет другим, шаблон не выполнится, и будут пропущены эти два элемента, обработается только Y.

В следующей трансформации порядок элементов будет не важен, т.к. есть два шаблона и один из них будет выполнен

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT1"/>

  <tt:template>

    <roots>

      <root1>

        <tt:cond>

          <X tt:value-ref="ROOT1.X"/>

          <Z tt:value-ref="ROOT1.Z"/>

        </tt:cond>

        <tt:cond>

          <Z tt:value-ref="ROOT1.Z"/>

          <X tt:value-ref="ROOT1.X"/>

        </tt:cond>

        <Y tt:value-ref="ROOT1.Y"/>

      </root1>

    </roots>

  </tt:template>

</tt:transform>

Шаблоном можно назвать условия, содержимое которых содержит

* Обычный текст в виде XML элементов и статических атрибутов
* Динамические атрибуты, объявленные в команде tt:attribute
* Текст, если он не пустой
* Явно определенный пустой шаблон, через команду tt:empty, может быть использован, если не известно есть ли внутри узла компоненты.

***Порядок обработки условных трансформаций при сериализации***

* Предварительные условия
* Утверждения
* Условия сравнения

Блок условий обрабатывается, когда все три проверки возвращают истинное значение.

***Порядок обработки условных трансформаций при десериализации***

1. Является ли содержимое условной трансформации шаблоном

1.1. Если содержимое является шаблоном, сравниваются последовательно элементы в XML потоке. Если шаблон не выполняется, условная трансформация пропускается, при этом элементы из XML потока не пропускается из обработки. Если шаблон корректен, переходим к шагу 2.

1.2. Если не шаблон переходим к шагу 2.

2. Проверка предусловий

2.1. Если содержимое является шаблоном и предусловие не выполняется, команда tt:cond пропускается, при этом элементы во входящем XML потоке пропускаются (не десериализуются). Если предусловие выполняется, переходим к шагу 3.

2.2. Если содержимое не является шаблоном и предусловие не выполняется, при десериализации срабатывает исключение CX\_ST\_REF\_ACCESS. Если предусловие выполняется, переходим к шагу 3.

3. Выполнение инструкций внутри условной трансформации. На данном этапе происходит десериализация узлов данных, их существование можно было проверить на шаге 2. Результат десериализации можно будет проверить на шаге 5.

4. Установка утверждений. Если в момент десериализации значение в утверждении не совпадет со значением в XML потоке, система вызовет исключение CX\_ST\_COND\_CHECK\_FAIL.

5. Проверка условий. Если условия не выполняются, десериализация будет прервана с исключением CX\_ST\_COND\_CHECK\_FAIL.

#### **Использование команды выбора**

Аналогично оператору CASE в ABAP, в ST программах может использоваться команда выбора, позволяющая от определенных условий выполнять те или иные ST команды или обработать шаблон, синтаксис данной команды

<tt:switch[-var]>

   <tt:[s-|d-]cond ..>

   </tt:[s-|d-]cond ..>

…

<tt:[s-|d-]cond ..>

   </tt:[s-|d-]cond ..>

</tt:switch>

Префикс var применяется, когда команда выбора применяется относительно переменной или параметра.

В отличие от условных трансформаций, используемых вне оператора tt:case, tt:group, нет необходимости в объявлении хотя-бы одного из атрибутов using,check,data, если содержимое условных трансформаций не шаблон.

При использовании команд выбора необходимо помнить следующее

* Вы можете определить только одно условие стандартное для сериализации (s-), т.е. то где не заполнены атрибуты check,using, data.
* Вы можете определить только одно стандартное условие десериализации (d-), т.е. то, что не содержит шаблон.

Шаги, выполняемые при сериализации

1. Первое релевантное для сериализации условие выбора tt:[s-]cond, выбранное относительно условий в using, data, check обрабатывается, далее происходит выход из команды tt:switch.
2. Если подобного условия не было найдено, обрабатывается условие выбора без атрибутов с проверками (без data, check, using – стандартное условие сериализации, если такое условие выбора есть), далее выход из команды tt:switch.
3. Если ни одно из условий не было обработано система вызовет исключение CX\_ST\_SWITCH\_NO\_CASE.

Шаги, выполняемые при десериализации

1. Сначала ищется первое подходящее условие выбора tt:[d-]cond с подходящим шаблоном и условиями, заданными через атрибуты. Если найдено подходящее, произойдет выход из команды tt:switch.
2. Если система не нашла подходящего условия выбора, обрабатывается стандартное условие для десериализации, если такое существует. Если при этом условия определенные через атрибуты для стандартного условия выбора не выполняются, система выдает исключение CX\_ST\_REF\_ACCESS.
3. Если система не смогла найти подходящего условия выбора, она выдаст исключение  CX\_ST\_SWITCH\_NO\_CASE.

Следующий пример демонстрирует работу с командой switch, при сериализации числовые значения заменяются на текстовые, при десериализации текстовые заменяются числовыми

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="SIZE"/>

  <tt:template>

    <Paragraph>

      <tt:attribute name="size">

        <tt:switch>

          <tt:s-cond check="SIZE<10">

            <tt:text>Small</tt:text>

          </tt:s-cond>

          <tt:s-cond check="SIZE>20">

            <tt:text>Big</tt:text>

          </tt:s-cond>

          <tt:s-cond>

            <tt:text>Medium</tt:text>

          </tt:s-cond>

          <tt:d-cond using="exist(SIZE)" data="SIZE=8">

            <tt:text>Small</tt:text>

          </tt:d-cond>

          <tt:d-cond using="exist(SIZE)" data="SIZE=16">

            <tt:text>Medium</tt:text>

          </tt:d-cond>

          <tt:d-cond using="exist(SIZE)" data="SIZE=28">

            <tt:text>Big</tt:text>

          </tt:d-cond>

          <tt:d-cond using="exist(SIZE)" data="SIZE=12">

            <tt:skip/>

          </tt:d-cond>

        </tt:switch>

      </tt:attribute>

      <tt:text>Text</tt:text>

    </Paragraph>

  </tt:template>

</tt:transform>

Тот же пример без использования префиксов, но с указанием направления для команды выбора

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="SIZE"/>

  <tt:template>

    <Paragraph>

      <tt:attribute name="size">

        <tt:serialize>

          <tt:switch>

            <tt:cond check="SIZE<10">

              <tt:text>Small</tt:text>

            </tt:cond>

            <tt:cond check="SIZE>20">

              <tt:text>Big</tt:text>

            </tt:cond>

            <tt:cond>

              <tt:text>Medium</tt:text>

            </tt:cond>

          </tt:switch>

        </tt:serialize>

        <tt:deserialize>

          <tt:cond using="exist(SIZE)">

            <tt:switch>

              <tt:cond data="SIZE=8">

                <tt:text>Small</tt:text>

              </tt:cond>

              <tt:cond data="SIZE=16">

                <tt:text>Medium</tt:text>

              </tt:cond>

              <tt:cond data="SIZE=28">

                <tt:text>Big</tt:text>

              </tt:cond>

              <tt:cond data="SIZE=12">

                <tt:skip/>

              </tt:cond>

            </tt:switch>

          </tt:cond>

        </tt:deserialize>

      </tt:attribute>

      <tt:text>Blahblah</tt:text>

    </Paragraph>

  </tt:template>

</tt:transform>

#### **Использование группировок**

В том случае, когда не известно сколько раз, тот или иной элемент в XML документе будет повторяться и в какой последовательности, можно воспользоваться группировками.

Следующий пример демонстрирует такой случай

DATA:

  lv\_xml TYPE string,

  x1     TYPE i,

  x2     TYPE i,

  x3     TYPE i.

lv\_xml = '<roots>' &&

         '  <X1>1</X1>' &&

         '  <X2>2</X2>' &&

         '  <X1>3</X1>' &&

         '  <X3>1</X3>' &&

         '  <X2>1</X2>' &&

         '</roots>'.

CALL TRANSFORMATION ZTEST\_ST2

  SOURCE XML lv\_xml

  RESULT x1 = x1

         x2 = x2

         x3 = x3.

WRITE: / 'X1=', X1, 'X2=', X2, 'X3=', X3.

Для удобства обработки такой последовательности, используются группировки

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="X1"/>

  <tt:root name="X2"/>

  <tt:root name="X3"/>

  <tt:template>

    <roots>

      <tt:group>

        <tt:cond frq="\*">

          <X1 tt:value-ref="X1"/>

        </tt:cond>

        <tt:cond frq="\*">

          <X2 tt:value-ref="X2"/>

        </tt:cond>

        <tt:cond frq="\*">

          <X3 tt:value-ref="X3"/>

        </tt:cond>

      </tt:group>

    </roots>

  </tt:template>

</tt:transform

Как и для команды tt:switch условные трансформации можно создавать для конкретного направления ([s-|d-]cond). Но нельзя внутри них использовать вложенные условные трансформации с другим типом направления.

В отличие от условных трансформаций, используемых вне оператора tt:case, tt:group, нет необходимости в объявлении хотя-бы одного из атрибутов using,check,data, если содержимое условных трансформаций не шаблон.

Для контроля того, как часто относительно десериализации условная трансформация может быть выполнена, заполняется параметр frq (frequency- частота)

* 1 – должна быть обработана 1 раз.
* ? – может быть обработана, а может и нет.
* \* — может быть обработана ноль и более раз.

**Сериализация** - во время сериализации элемент tt:group выполняется один раз: все зависимые относительно сериализации условные трансформации с предустановленными условиями или без них выполняются в указанном порядке.

**Десериализация** - при десериализации элемент tt:group инициирует цикл. Система пытается обработать все условные трансформации с обязательным присутствием элемента (частота -1). Как только это происходит, цикл завершается. Пока не обработаны все обязательные условные трансформации, могут быть так же обработаны опциональные (? – встречается хотя бы раз) и множественные условные трансформации (\*). На каждой итерации цикла, одна из подходящих для сериализации условных трансформаций, чья частота не достигла указанного максимума, может быть выбрана следующим образом

1. Выбирается первая условная трансформация с подходящим шаблоном
2. Если такая не обнаружена, система пытается выполнить стандартную для десериализации условную трансформацию, не содержащую шаблона. Если для неё не будут выполнены ограничения: using, check, data система выдаст исключение CX\_ST\_REF\_ACCESS.
3. Если система не может выполнить обязательных условных трансформаций, вызывается исключение CX\_ST\_GROUP\_MISSING\_CASE.

## Модульное построение ST программ

В ST программах возможно разбиение логики на отдельные части, путем определения и вызова вложенных шаблонов. Все шаблоны в ST программе, за исключением главного являются вложенными.

### ***Определение вложенных шаблонов***

Вложенные шаблоны определяются следующим способом

<tt:template name=“tmpl“>

  [<tt:context>

    [<tt:root name="root1" [[line-]type=...] />

     <tt:root name="root2" [[line-]type=...] />

     ...]

    [<tt:parameter name="para1" [[s-|d-]val="def1"] [kind="knd1"] />

     <tt:parameter name="para2" [[s-|d-]val="def2"] [kind="knd2"] />

     ...]

    [<tt:variable name="vari1" [[s-|d-]val="val1"] />

     <tt:variable name="vari2" [[s-|d-]val="val2"] />

     ...]

   </tt:context>]

  ...

</tt:template>

Атрибут name определяет уникальное имя для вложенного шаблона, имя не должно быть пустым и не должно совпадать с главным шаблоном или другими вложенными шаблонами.

Во вложенных шаблонах через команду tt:context можно объявлять свои узлы данных, параметры и переменные. Узлы данных во вложенных шаблонах не имеют прямой связи с ABAP данными, они используются (как и параметры) в роли интерфейса между шаблонами, т.е. мы можем, запуская вложенный шаблон, передать в значения его узлов данные из узлов запускающего шаблона через команду tt:with-root.  В значения параметров так же передаются значения из вызывающих шаблонов. Переменные (tt:variable) используются для внутренних целей шаблона и не изменяются через вызывающий шаблон, доступ к ним может иметь только шаблон, внутри которого они определены.

### ***Вызов вложенных шаблонов***

Для вызова вложенных шаблонов используется команда tt:apply

<tt:apply name="tmpl" [ref="node"]>

  [<tt:with-root name="root1" [ref="node1"] />

   <tt:with-root name="root2" [ref="node2"] />

   ...]

  [<tt:with-parameter name="para1" [ref="node1"|val="val1"|var="var1"] />

   <tt:with-parameter name="para2" [ref="node2"|val="val2"|var="var2"] />

   ...]

</tt:apply>

Атрибут name задает имя вложенного шаблона, ref- текущий узел, для передачи данных во внутренние узлы данных используется команда tt:with-root, для передачи в параметры команда tt:with-parameter.

Пример трансформации с использованием вложенного шаблона

tt:transform template="TEMP\_MAIN"

    xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:root name="ROOT"/>

  <tt:template name="TEMP\_MAIN">

    <week>

      <day1 tt:ref="ROOT.DAY1">

        <tt:apply name="TEMP\_SUB" />

      </day1>

      <day2 tt:ref="ROOT.DAY2">

        <tt:apply name="TEMP\_SUB" />

      </day2>

      <day3 tt:ref="ROOT.DAY3">

        <tt:apply name="TEMP\_SUB" />

      </day3>

      <day4 tt:ref="ROOT.DAY4">

        <tt:apply name="TEMP\_SUB" />

      </day4>

      <day5 tt:ref="ROOT.DAY5">

        <tt:apply name="TEMP\_SUB" />

      </day5>

      <day6 tt:ref="ROOT.DAY6">

        <tt:apply name="TEMP\_SUB" />

      </day6>

      <day7 tt:ref="ROOT.DAY7">

        <tt:apply name="TEMP\_SUB" />

      </day7>

    </week>

  </tt:template>

  <tt:template name="TEMP\_SUB">

    <name>

      <tt:value ref="$ref.name" />

    </name>

    <work>

      <tt:value ref="$ref.work" />

    </work>

  </tt:template>

</tt:transform>

Программа для её вызова

TYPES: BEGIN OF day,

         name TYPE string,

         work(1) TYPE c,

       END OF day.

DATA: BEGIN OF week,

        day1 TYPE day,

        day2 TYPE day,

        day3 TYPE day,

        day4 TYPE day,

        day5 TYPE day,

        day6 TYPE day,

        day7 TYPE day,

      END OF week.

DATA xml\_string TYPE string.

DATA result LIKE week.

week-day1-name = 'Monday'.    week-day1-work = 'X'.

week-day2-name = 'Tuesday'.   week-day2-work = 'X'.

week-day3-name = 'Wednesday'. week-day3-work = 'X'.

week-day4-name = 'Thursday'.  week-day4-work = 'X'.

week-day5-name = 'Friday'.    week-day5-work = 'X'.

week-day6-name = 'Saturday'.  week-day6-work = ' '.

week-day7-name = 'Sunday'.    week-day7-work = ' '.

CALL TRANSFORMATION ...

  SOURCE root = week

  RESULT XML xml\_string.

CALL TRANSFORMATION ...

  SOURCE XML xml\_string

  RESULT root = result.

Результат

<week>

  <day1>

    <name>Monday</name>

    <work>X</work>

  </day1>

  <day2>

    <name>Tuesday</name>

    <work>X</work>

  </day2>

  <day3>

    <name>Wednesday</name>

    <work>X</work>

  </day3>

  <day4>

    <name>Thursday</name>

    <work>X</work>

  </day4>

  <day5>

    <name>Friday</name>

    <work>X</work>

  </day5>

  <day6>

    <name>Saturday</name>

    <work/>

  </day6>

  <day7>

    <name>Sunday</name>

    <work/>

  </day7>

</week>

## Использование других ST программ

В ST программах есть возможность вызова других ST программ и включение содержимого одной ST программы в другую.

### ***Вызов ST программ***

Для вызова программ необходимо воспользоваться командой tt:call

<tt:call transformation=“trafo“>

  [<tt:with-root name="root1" [ref=”node1”] / >

   <tt:with-root name=" root2" [ref=”node2”] / >

   ...]

  [<tt:with-parameter name="para1" [ref=”node1”|val=”val1”|var=”var1”] />

   <tt:with-parameter name="para2" [ref=”node2”|val=”val2”|var=”var2”] />

   ...]

</tt:call>

Атрибут transformation задает имя ST программы, как и при вызове вложенных шаблонов ST программу можно вызвать, передав в неё узлы данных и параметры (описание см. выше).

### ***Включение содержимого программ***

Для включения содержимого одной программы в другую необходимо воспользоваться командой tt:include, со следующим синтаксисом

<tt:include name=" trafo " [templates=""tmpl1 tmpl2 ...] />

Данная команда не может быть вызвана внутри шаблона, по умолчанию она копирует все именованные вложенные шаблоны (пропуская шаблоны без имени). Необходимо убедиться в том, что не будет дублирования имен шаблонов. Через атрибут templates можно напрямую задать, какие вложенные шаблоны необходимо скопировать.

## Доступ к ABAP объектам из ST программ

В ST программах можно использовать объекты и классы из ABAP’а следующим образом

* Вызывать статические методы объекта
* Вызывать методы инстанции объекта
* Создавать объекты
* В методах так же можно получать доступ к XML потоку

### ***Вызов методов***

Для вызова метода необходимо воспользоваться командой tt:call-method

<tt:call-method [var=”oref”] [class="class"] [s-|d-]name="meth"

                              [writer = "writer\_para"]

                              [reader = "reader\_para"]

  [<tt:with-parameter [s-|d-]name="para1"

                      [ref="node1"|val="val1"|var="var1"] />

   <tt:with-parameter [s-|d-]name="para2"

                      [ref="node2"|val="val2"|var="var2"] />

   ...]

</tt:call-method>

Если вызывается статический метод, в атрибуте class указывается имя класса, в котором этот метод находится, если метод инстанции, то в атрибут var передается имя переменной или параметра, ссылочного типа, которая указывает на объект, атрибут ref-type переменной или параметра должен быть явно указан. Атрибут name указывает на имя метода, а префиксы в случае необходимости можно использовать для разграничения направления трансформации.

С помощью элементов tt:with-paramteter задаются параметры, передающиеся в метод класса. Атрибут name указывает на имя параметра, префиксы s- и d- указывают на направление трансформации. В качестве значения может быть передана ссылка на узел данных (ref), параметр или переменная (var) или значение (val).

Не стоит забывать, что при сериализации узлы будут доступны только для чтения, а при десериализации только для записи.

Тип передаваемого в метод параметра должен совпадать с типом в интерфейсе метода. Типом узла данных, в котором явно не указана ссылка на тип, будет тип ABAP переменной связанной с этим узлом. Параметры и переменные, не имеющие явного указания на тип через атрибут ref-type, определяются как ссылочные переменные.

Когда выполняется ABAP метод, управление из ST обработчика переходит к ABAP обработчику. Нет никаких ограничений на конструкции, используемые в методе, если выполняются операторы ENDMETHOD и RETURN управление возвращается назад к ST обработчику, однако возврат может быть не осуществлен если

* Если будут запущены операторы LEAVE PROGRAM, SUBMIT без дополнений AND RETURN, LEAVE TO TRANSACTION.
* Метод вызвал исключение объектное исключение, тогда ST программа вызовет свое исключение CX\_ST\_CALL\_METHOD\_ERROR, в котором в качестве атрибута EXCEPTION\_NAME будет передано имя оригинального класса исключения.
* Если будет вызвано не объектное исключение, ST обработчик вызовет не отлавливаемое исключение.

Если к узлам данных привязаны данные, которые изменяются вызовом метода, эти изменения немедленно отображаются на узлах данных. Но существуют следующие ограничения: в ST команде tt:loop нельзя изменить данные таблицы, которые обрабатываются в цикле.

В атрибуты reader и writer передается имя атрибута, в вызываемом методе, который является ссылочным типом, со ссылкой на интерфейс IF\_SXML\_WRITER или IF\_SXML\_READER. В момент вызова метода, в него передается ссылочная переменная на объект, реализующий данный интерфейс, через который можно получить доступ к XML потоку. Объект создается каждый раз при вызове метода. В зависимости от объявления имени метода и его направления можно указать только writer, если метод запущен для сериализации и только reader, если метод запущен для десериализации. Если метод запускается для обоих направлений, можно указать как reader, так и writer.

Пример вызова статического метода в сериализации

<?sap.transform simple?>

<tt:transform xmlns:tt="http://www.sap.com/transformation-templates">

  <tt:parameter name="STR"/>

  <tt:root name="X1"/>

  <tt:template>

    <tt:call-method class="CL\_ABAP\_DEMO\_SERVICES" name="IS\_PRODUCTION\_SYSTEM">

      <tt:with-parameter name="FLAG" var="STR"/>

    </tt:call-method>

    <X tt:value-ref="X1"></X>

  </tt:template>

</tt:transform>

Программа, вызывающая трансформацию

DATA:

  lv\_xml    TYPE string,

  lv\_is\_production TYPE abap\_bool,

  lv\_x  TYPE i.

lv\_xml = '<X>1</X>'.

CALL TRANSFORMATION ZTEST\_ST2

  SOURCE XML lv\_xml

  PARAMETERS str = lv\_is\_production

  RESULT x1 = lv\_x.

WRITE: / lv\_is\_production, lv\_x.

**Создание объектов**

При вызове методов инстанции ABAP объектов, необходимо чтобы эти объекты существовали. Если в качестве ссылочной переменной, при вызове метода, указана ST переменная, необходимо чтобы объект был создан (Параметр с объектом может быть создан вне ST программы). Для создания объектов необходимо воспользоваться командой tt:create-object

<tt:create-object var="oref" [class="class"]>

  [<tt:with-parameter name="para1"

                      [ref="node1"|val="val1"|var="var1"] />

   <tt:with-parameter name="para2"

                      [ref="node2"|val="val2"|var="var2"] />

   ...]

</tt:call-method>

Атрибут var – содержит имя переменной или параметра со ссылочной переменной, указывающей на ABAP класс или интерфейс. Атрибут class определяет имя создаваемого класса

* Если var – переменная со ссылкой на класс, атрибут class должен содержать имя этого класса или имя класса его наследника.
* Если var – переменная со ссылкой на интерфейс, атрибут class должен содержать имя класса, его реализующего.

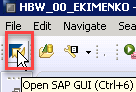
Через элементы tt:with-parameter передаются параметры в конструктор класса. Исключения, создаваемые внутри класса и правила передачи параметров те же, что и для вызова методов.

**Цепочки загрузки данных**

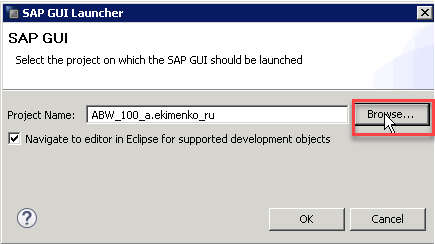
[Содержание](#Содержание)

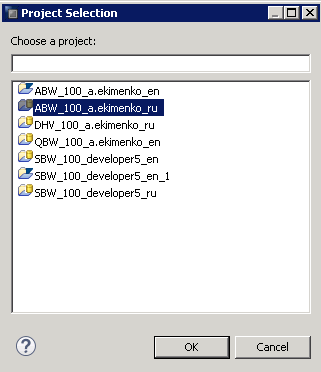
**Программа ведения цепочек**

Нажать на сине-желтую пиктограмму *Open SAP GUI*

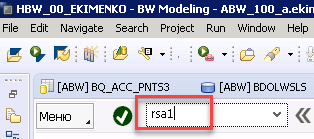


Нажать кнопку *Browse*, чтобы выбрать проект *ABW* /если он не выбран/



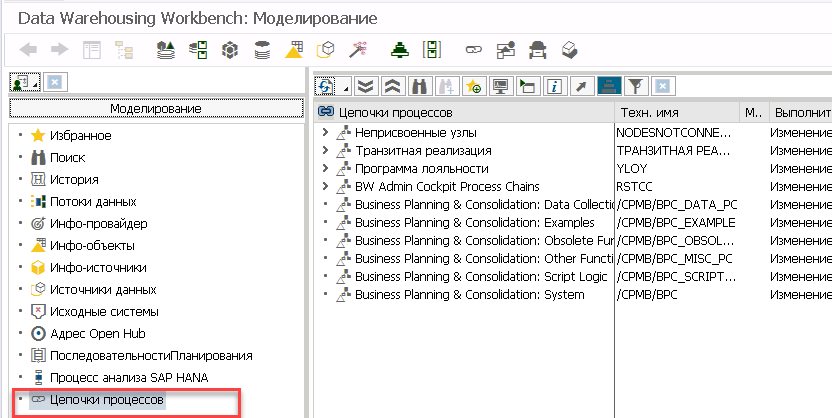


Откроется новая вкладка - на ней в верхнем левом углу ввести имя транзакции /программы/ ***rsa1***

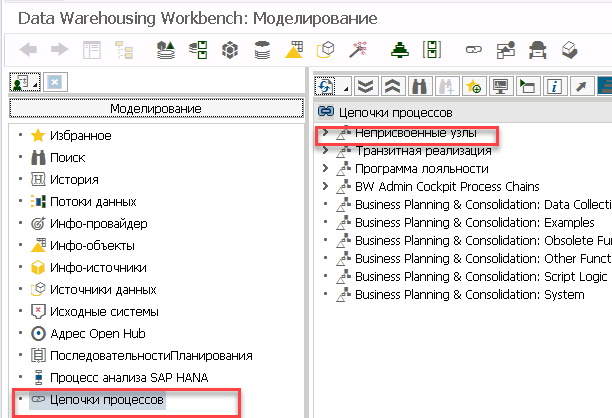
 и нажать *зеленую кнопку с галкой* слева

C:\Users\ABD50~1.EKI\AppData\Local\Temp\SNAGHTML7662b3.PNG.

В интерфесе программы rsa1 – выбрать слева пункт *Цепочки процессов*

.

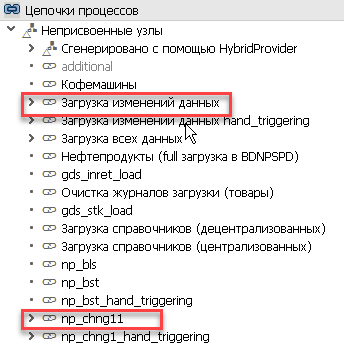
В правом окне – выбрать ветку *Неприсвоенные узлы*.

.

Раскрыть ее – появится список разных цепочек загрузки.

Нас интересуют только две из них – это такие мега цепочки, которые содержат в себе все остальные.

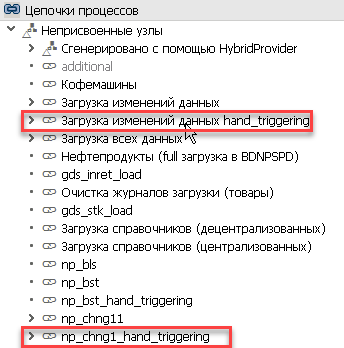
Это

.

Где *Загрузка изменений данных* – это цепочка загрузки данных по товарам, а *np\_chng11* – это цепочка загрузки данных по топливу.

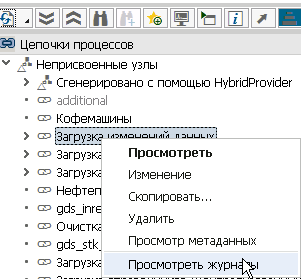
**Примечание**

У каждой из этих цепочек – есть версия для ручного запуска

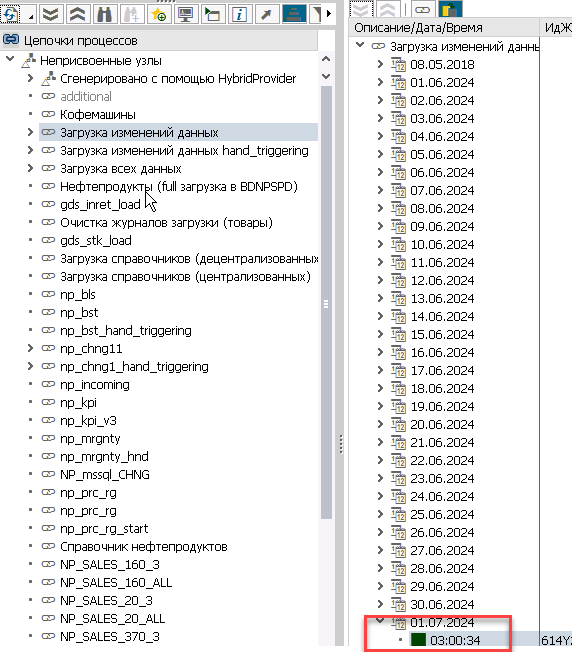
.

**Просмотр журнала выполнения цепочки**

[Содержание](#Содержание)

Находясь на цепочке – нажать пр. кнопу мыши и выбрать .

Если в журнале – зеленый квадрат – то все хорошо /цепочка корректно отработала/

.

**Цепочки с ручным запуском**

[Содержание](#Содержание)

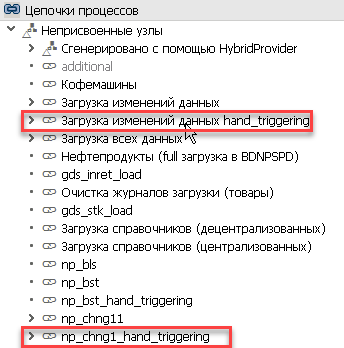
Иногда с источниками данных, откуда данные грузятся в SAP BW, могут быть проблемы. Такое случается очень редко, но тем не менее.

Основной источник – это сервер mssql *bl-sql-dw*. Там две базы данных – *dwbeloil* /данные с чеками от АЗС/ и KISjoinMDM /данные КИС НПО – справочники и данные по реализации топлива и товаров/.

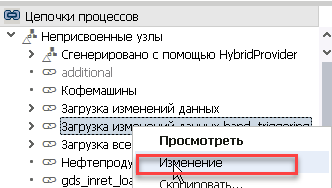
Вспомогательные источники – базы данных Oracle

* *Курсы валют* TUNNEL.IT.BELOIL.BY:21521/GML1, схема EPC\_SAP\_EXCHANGE.
* *Адреса комплексов АЗС* программы *Паспортизация* tunnel.it.beloil.by:31521/ORAWEB, схема PASSPORT/.
* *Служебные топливные карты* - TUNNEL.it.beloil.by:41521/OPER.it.beloil.by, схема doc\_validator.

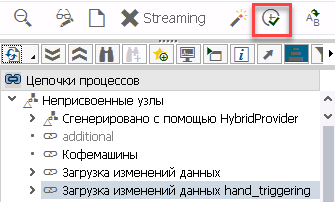
Если, например, произошел сбой на сервере mssql и данные загрузились /цепочка отработала/, но есть подозрение, что на момет загрузки в mssql пришлина не все данные от АЗС /продажи резко отличаются от предыдущего дня/, то тогда можно запустить цепочку с ручным запуском

.

Для этого, находясь на цепочке, - нажать пр. кнопку мыши и выбрать *Изменение*

.

Потом вверху нажать на пиктограмму с часами - *Запланировать*

.

Появится диалоговое окно, где система предложит выбрать приоритет выполнения цепочки – выбрать высший *A*.

**Внимание**

Нельзя запускать цепочки -  и , поскольку они могут блокировать друг-друга /обращаются к одним и тем же источникам данных/.

Поэтому сначала дождаться, когда отработает одна – потом когда другая.

**Для справки**

Цепочка по товарам  автоматически стартует в 3:00 и длится 50-60 мин.

Цепочка по топливу  - в 1:40 и длится 30-40 мин.

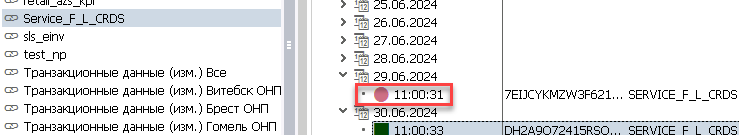
Аналоги с ручным запуском отрабатывают намного /в разы/ дольше, потому что происходит чтение из таблиц, куда в это же время пишутся данные.

Обработчик чеков от АЗС отключается в 24:00 с тем, чтобы дать загрузиться данным и включается в 9:00. Т.е. цепочки с ручным запуском работают параллельно с обработчиком чеков.

**Что делать, если в журнале цепочки – красный кружок /ошибка/**

[Содержание](#Содержание)

Например, 29.06.24 был недоступен сервер, от которого забираются служебные топливные карты

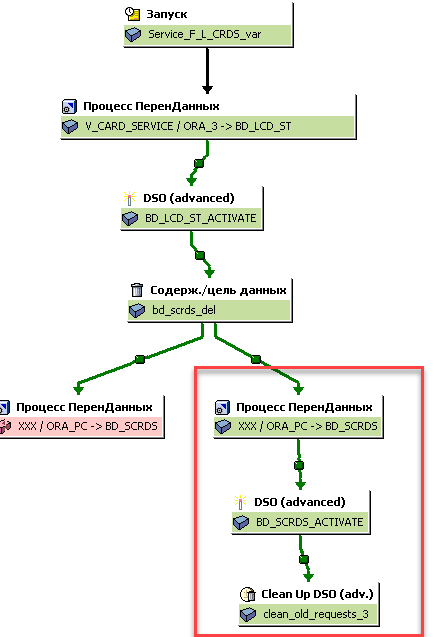
.

Что в таком случае делать.

Стать на тот элемент цепочки, который выделен красным – нажать пр. кнопку мыши и выбрать *Пропустить*. Есть еще вариант – выбрать не *Пропустить*, а *Исправить*, но это в том случае, когда есть уверенность, что с источником данных уже все в порядке.

Чтобы не забивать голову – лучше просто всегда выбирать *Пропустить* .

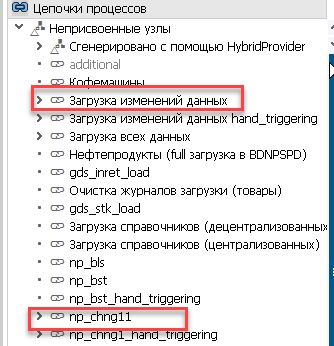
У меня картинки нет, когда выбираю *Пропустить* или *Исправить* но есть картинка после того как нажал *Исправить* – справа выделен блок шагов, которые система выполнила повторно.



**Основные цепочки**

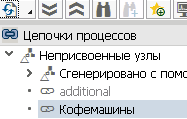
[Содержание](#Содержание)

Данные по ***товарам*** /*Загрузка изменений данных* - старт в 3:00, длительность ~ 50 мин/ и ***топливу*** /*np\_chng11* - старт в 1:40 длительность ~ 40 мин/



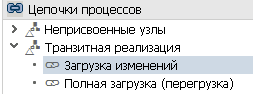
***Служебные топливные карты и карты лояльности*** *Service\_F\_LCRDS* - - старт в 11:00, длительность ~ 1 мин.

***Опт. продажи топлива*** *oil\_wholesales* - - старт в 7:45, длительность ~ 1 мин.

***Реализация через кофемашины***  - старт в 9:00, длительность ~ 4 мин.

***Маржинальность нефтепродуктов*** /расчет надбавок на топливо/ *np\_mrgnty* и ее версия для ручного запуска - старт в 9:30, длительность ~ 2 мин.

***Транзитная реализация*** /данные Процессингового центра топливных карт/

 - старт в 10:00, длительность ~ 4 мин.

Эта цепочка /в отличие от остальных/ находися не в узле *Неприсвоенные узлы,* а в узле *Транзитная реализация*.