**XML & http\_post & http\_get**

[Содержание](#Содержание)

[Simple Transformations](#Simple_Transformations)

[http\_post](#http_post)

### [Курсы валют](#Курсы_валют)

http\_get

## [Simple XML Processing in ABAP Part I. – Overview](#Simple_XML_Processing_inABAP_Part_I)

## [Simple XML Processing in ABAP Part II. – Create ABAP Dictionary Objects](#Simple_XML_Processing_in_ABAP_Part_II)

## [Simple XML Processing in ABAP Part III. – Create Simple Transformation to Parse XML Conten](#Simple_XML_Processing_in_ABAP_Part_III)t

## [Simple XML Processing in ABAP Part IV. – Create Framework Program to Process XML Files](#Simple_XML_Processing_in_ABAP_Part_IV)

## [Simple XML Processing in ABAP Part V. – Processing XML Attributes](#Simple_XML_Processing_in_ABAP_Part_V)

# XSLT преобразование внутренней таблицы в ABAP имеющей поле типа generic reference

# <https://www.pvsm.ru/programmirovanie/59207>

**Simple Transformations**

[Содержание](#Содержание)

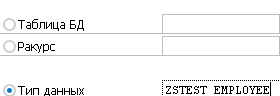
### **Простая трансформация – пример**

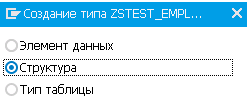
<https://abap-guide.blogspot.com/2019/05/abap-simple-transformation.html>

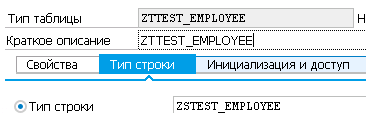
Допустим, имеем простой XML-файл.

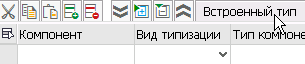
|  |  |
| --- | --- |
| <?xml version="1.0" encoding="utf-8"?>  <Company>  <Name>SuperCompany</Name>  <Address>Russia, Moscow</Address>  <Code>AA-001</Code>  <Employee *Type="FT"*>  <Name>James</Name>  <Number>007</Number>  </Employee>  <Employee Type="FT">  <Name>Steve</Name>  <Number>010</Number>  </Employee>  </Company> |  |

**Шаг 1** *Создаем структуру ZSTEST\_EMPLOYEE /se11/*









|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Длина | Описание |
| TYPE | CHAR | 2 |  |
| EMP\_NAME | CHAR | 50 |  |
| EMP\_NUMBER | NUMC | 10 |  |

@EndUserText.label : 'ZSTEST\_EMPLOYEE'

@AbapCatalog.enhancementCategory : #NOT\_CLASSIFIED

define type **zstest\_employee** {

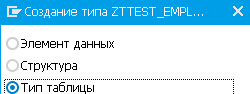
type : abap.char(2);

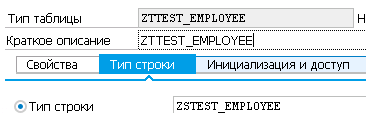
emp\_name : abap.char(50);

emp\_number : abap.numc(10);

}

**Шаг 2** *Создаем тип таблицы ZTTEST\_EMPLOYEE*





**Шаг 3** *Создаем структуру ZSTEST\_COMPANY*

**Структура ZSTEST\_COMPANY**

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Длина | Описание |
| COMP\_NAME | CHAR | 50 |  |
| COMP\_ADDRESS | CHAR | 100 |  |
| EMPLOYEE | *ZTTEST\_EMPLOYEE* |  |  |

@EndUserText.label : 'ZSTEST\_COMPANY'

@AbapCatalog.enhancementCategory : #NOT\_CLASSIFIED

define type **zstest\_company** {

comp\_name : abap.char(50);

comp\_address : abap.char(100);

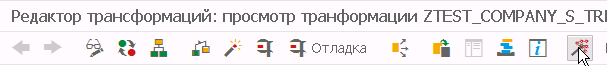
employee : zttest\_employee;

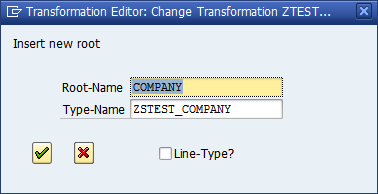
}

*Rem*

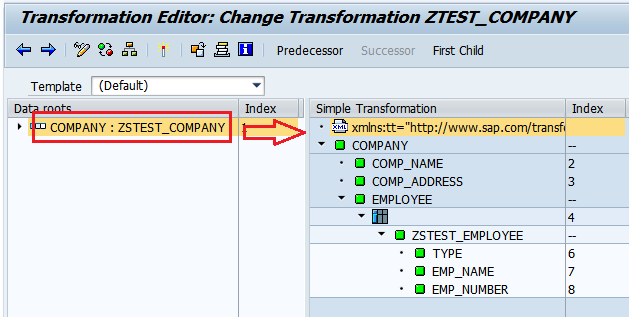
ZTTEST\_EMPLOYEE - Тип таблицы со структурой ZSTEST\_EMPLOYEE.

#### **Создание трансформации**

В транзакции XSLT\_TOOL /или ***strans***/ ввести имя трансформации /*ztest\_company\_*s\_*trfn*/, нажать *Создать*. Далее ввести краткое описание и выбрать тип **S** /*простая трансформация*/. Нажать кнопку *Графическая обработка простой трансформации*  /Ctrl+Shift+F11/. Нажать пр. кнопкой в левой части, выбрать *Insert new root*.

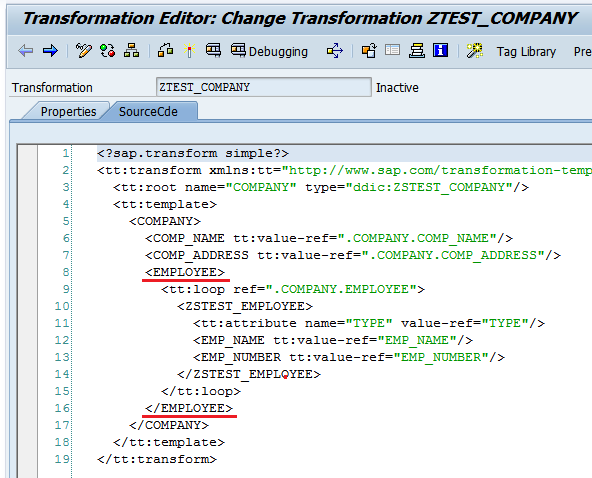
[](https://1.bp.blogspot.com/-O9Jkyjg_0gE/XP43LbV18OI/AAAAAAAAAEE/26rvTe_zt3AUVJF0kcn0-unc5mxROcTtgCLcBGAs/s1600/pic01-input-root-name.png)

Нажать зелёную галку. Перетащить элемент COMPANY в правую часть.

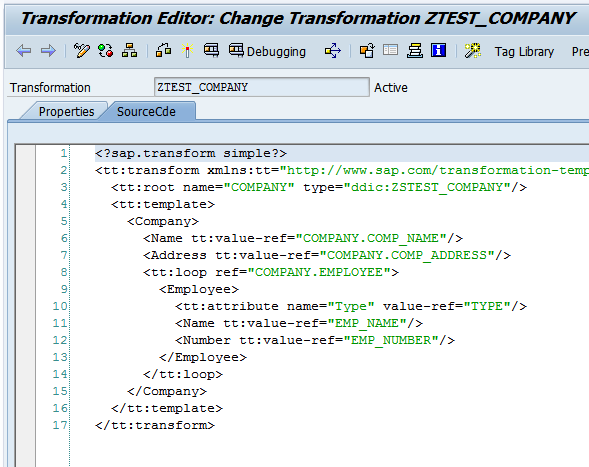
[](https://1.bp.blogspot.com/-1TuX_aiMX88/XP43bs2BkJI/AAAAAAAAAEM/RZ7wi6znAGspRtCTElKu2ku8pNVYmCTMQCLcBGAs/s1600/pic02-create-root-element.png)

Элемент ROOT в левой части можно удалить. Теперь в правой части - иерархию узлов нужно привести в соответствие с иерархией в XML-файле. В нашем случае надо *сделать элемент TYPE атрибутом /*<Employee *Type="FT"*>*/* - для этого нажать на нём пр. кнопкой и выбрать***Change to attribute****.* Обратите внимание, что символом таблицы отмечены элементы, которые могут повторяться в XML-файле. Сохраняем и нажимаем ***Назад***.

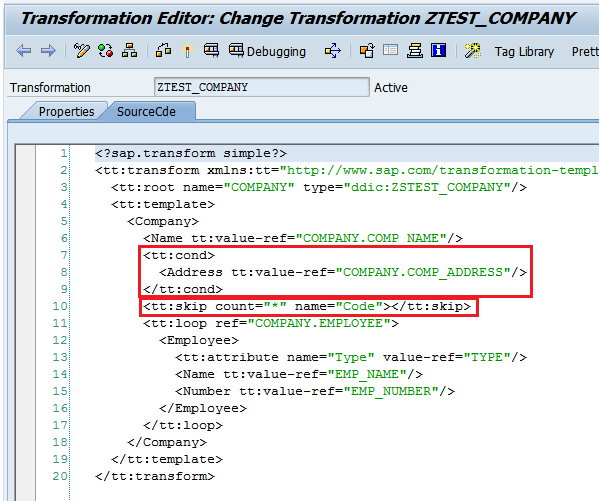
В редакторе подкорректируем имена узлов в соответствии с файлом и удалим лишние тэги. Например, в нашем случае *тэги <EMPLOYEE> и </EMPLOYEE> - лишние*.

[](https://1.bp.blogspot.com/-O8PewzkDzK4/XP43pmbMoTI/AAAAAAAAAEQ/xmqjXUjWwCE0DXduUDXLNcT5kBoiNP51wCLcBGAs/s1600/pic03-modify-xml-1.png)

Обратите внимание, что элементы между <tt:loop> и </tt:loop> могут повторяться в XML-файле. Должно получиться примерно так.

[](https://1.bp.blogspot.com/-ivHWzH75YXI/XP43zyuXqGI/AAAAAAAAAEY/00JPDALKf54ZKB2o12xzwsDMtdLmBYragCLcBGAs/s1600/pic04-modify-xml-2.png)

***Узлы, которые могут отсутствовать*** в XML-файле, нужно заключить в *<tt:cond></tt:cond>*. А ***узлы, которые нужно пропустить*** – в *<tt:skip></tt:skip>*. В примере ниже узел Address может отсутствовать, а узел Code будет пропущен.

[](https://1.bp.blogspot.com/-l1JJeev8_MQ/XP44AJMMLeI/AAAAAAAAAEg/saF7Cr1ZCGAj9JgQ_cDoT9up6lzHFSjcgCLcBGAs/s1600/pic05-modify-xml-3.png)

#### **ABAP-программа**

ABAP-код парсинга XML в структуру

DATA:

lt\_xml TYPE solix\_tab,

ls\_company TYPE *zstest\_company*,

lr\_excep1 TYPE REF TO cx\_st\_match\_element,

lr\_excep2 TYPE REF TO cx\_st\_match\_attribute,

lv\_msg TYPE string.

TRY.

CALL TRANSFORMATION *ztest\_company*

SOURCE XML lt\_xml

RESULT company = ls\_company.

CATCH cx\_st\_match\_element INTO lr\_excep1.

lv\_msg = lr\_excep1->get\_text( ).

CATCH cx\_st\_match\_attribute INTO lr\_excep2.

lv\_msg = lr\_excep2->get\_text( ).

ENDTRY.

И обратное преобразование из структуры в XML-файл

CLEAR: ls\_company, ls\_employee.

ls\_company-comp\_name = 'Company 1'.

ls\_employee-type = 'AA'.

ls\_employee-emp\_name = 'Nick'.

ls\_employee-emp\_number = '099'.

TRY .

CALL TRANSFORMATION *ztest\_company*

SOURCE company = ls\_company

RESULT XML lt\_xml.

CATCH cx\_root.

ENDTRY.

*Код трансформации*

<?sap.transform simple?>

<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:root name="COMPANY" type="ddic:ZSTEST\_COMPANY"/>

<tt:template>

<Company>

<Name tt:value-ref="COMPANY.COMP\_NAME"/>

<tt:cond>

<Address tt:value-ref="COMPANY.COMP\_ADDRESS"/>

</tt:cond>

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

<tt:loop ref="COMPANY.EMPLOYEE">

<Employee>

<tt:attribute name="Type" value-ref="TYPE"/>

<Name tt:value-ref="EMP\_NAME"/>

<Number tt:value-ref="EMP\_NUMBER"/>

</Employee>

</tt:loop>

</Company>

</tt:template>

</tt:transform>

# Простые трансформации

# <https://abap-blog.ru/osnovy-abap/prostye-transformacii/>

# ABAP -> XML – Сериализация

# XML -> ABAP – *Десериализация*

Основные характеристики ST программ, в сравнении с XSLT

* Повышенная производительность, за счёт последовательного доступа к XML элементам.
* 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.

***Не поддерживается***

* 1. преобразование *ссылочных переменных и объектов;*
  2. трансформация *структур и таблиц*, компонентами которых явл. ссылочные переменные.
* Допускается работа со ссылочными переменными - например, для вызова метода класса.

# Параметрымогут быть переданы через оператор CALL TRANSFORMATION /определяются командой *tt:parameter*/.

# Переменные определяются командой *tt:variable* - переменные из ABAP программы подставляются в параметрах source /сериализация/ и result /десериализация/ оператора CALL TRANSFORMATION.

# Десериализация XML -> ABAP

# XML документ представляет собой вх. 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.

# *Имена в текстовых элементах* зависят от регистра - <ELEMENT> и <element> два разных элемента.

**Элементы ST программы**

***Элементы ST программы***

* ***Корневые узлы данных*** /Data Roots/ - *tt:root* - определят связь между ABAP данными и ST программой;
* ***Параметры*** /*tt:parameter*/ - передаются через оператор CALL TRANSFORMATION/;
* ***Переменные*** /*tt:variable/* из ABAP программы - подставляются в параметрах *source* /сериализация/ и *result* /десериализация/ оператора CALL TRANSFORMATION.
* ***Типы данных*** /*tt:type/*;
* ***Вложенные шаблоны*** – используются для разбиения логики ST программы на части.

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 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.

Пример – *типы данных в ST*

<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"*/> *“ D - дата*

  <tt:root name="R2" *line-type="I"*/> *“таблица из integer*

  <tt:root name="R3" type="ddic:DSTRUCT"/> *“структура ABAP словаря*

  <tt:root name="R4" type="*tp*:ABCDSTRUCT"/> *“структура из пула типов*

*<!-- внутренняя таблица из структуры в классе cl\_abc -->*

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

  <tt:root name="R6" type="*def*:STRUCT"/> *“структура с типом из трансформации*

*Rem*

В приведенных ниже примерах проверка исключений во время трансформаций опущена, в продуктивных системах необходимо использовать оператор TRY..Catch.

Пример

*Трансформация ztest\_st*

<?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>

*И ее вызов из abap*

DATA:  gt\_test\_struct TYPE addr,

  lv\_xml          TYPE string.

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

gt\_test\_struct-line1 = 'Leningradskaya 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.

### **Определение параметров**

<*tt:parameter* name="..." [ref-type="..."] [kind="..."] [[s-val="..."][d-val="..."]]|[val="..."] />

где

* ***name***

1. max 30 символов;
2. must be *уникально* среди имен в корневых узлах | параметрах| переменных;
3. *не чувствительно к регистру*.

* **kind** – *in* – входящий /default/; *out* – исходящий; *in/out*.
* **s-val**, **d-val**, **val** – присваивают значение для параметра по умолчанию /s- сериализация и *d- десериализация*/.

|  |  |
| --- | --- |
| **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 |

* **ref-type** –

1. ссылка на глобальный ABAP класс | интерфейс - имя класса | интерфейса указывается без указания namespace;
2. ссылочная переменная - *REF TO DATA -* ссылочная переменная must be заранее определена через тип данных в команде *tt:type*, где нужно указать namespace.

***Получить значение параметра***, можно используя команду *tt:assign*.

***Параметры***

* *определенные в одном шаблоне не могут быть доступны в другом*;
* can be использованы для вызова
* вложенных шаблонов - ***tt:apply***
* др. ST программ - ***tt:call*** при указании атрибута ***with-parameter***

*tt:call* transformation="..."><*tt:with-parameter* name="PARA1" var="VARI1"/>

Пример 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>

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**

<tt:variable name="..." [ref-type="..."] [[s-val="..."][d-val="..."]]|[val="..."] />

***Переменные***

* объявленные в одном шаблоне, не могут быть *получены* в другом /при необходимости их значения *могут быть переданы при вызове шаблона*/;
* *не связаны с ABAP данными*.

Для получения значения переменных используется команда *tt:assign*.

### **Определение типов**

# Объявление типа происходит вне шаблонов ST программы

<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>

*Rem*

1. *Возможна ссылка на ранее определенный* /локально в ST программе или глобальный/ *тип данных* через атрибут t*ype*.
2. Имя типа указывается **в верхнем регистре** - p:NAME, где p – namespace.

***При определении ссылки через атрибут [line-]type*** ***можно использовать следующие namespaces***

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

**Определение элементарных типов**

C, D, F, I, N, P, STRING, T, X | XSTRING.

Для типов **C**, **N**, **P**, **X** – can be указан атрибут *length.*

для **P** - атрибут *decimals*.

Если для **C** не указать длину - тип может иметь любую длину, как для string или csequence. Тоже самое справедливо и для X.

### **Определение структурных типов**

Команда **tt:node** - определение структурного типа.

tt:node*, вложенная в tt:node* - определяет *подструктуру*.

Элемент tt:type | tt:node с вложенным tt:node must not have ссылку на тип в атрибуте *type*, вместо этого должно стоять значение **?**.

Пример - объявление вн. таблицы

<tt:node line-type=”?”>

  <*tt:node type=”C”/>*

</tt:node>

Пример 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"/> *“структура ABAP словаря*

      <tt:node name="C2" line-type="def:T2"/> *“структура с типом из трансформации*

    </tt:node>

  </tt:type>

  <tt:type name="T2">

    <tt:node name="D">

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

    </tt:node>

  </tt:type>

Порядок, в котором компоненты расположены в структуре, не имеет значения при связывании с ABAP структурой.

Для определения порядка*, в котором будут расположены компоненты структуры* можно воспользоваться элементом **tt:front** /can be задан в единственном числе для элементов tt:node и tt:type/.

<tt:type …

  <tt:front>

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

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

</tt:front>]

</tt:type>

Некоторые поля могут отсутствовать в указании типа - для определения ограничений на тип структуры используется ***атрибут extensible***

* ***on*** – текущий элемент may have вложенные элементы, явно не определенные в типе.
* ***deep*** – текущий и дочерние элементы may have вложенные элементы, явно не определенные /default/.
* ***off*** – текущий элемент must not have др. элементов, явно не описанных в типе.
* ***deep-off*** – текущий и его дочерние элементы не должны иметь элементы, явно не описанные в типе.

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

  extensible=extensible\_flag>

…

</tt:type>

## **Адресация ABAP данных в ST программах**

# В ST программах все объекты данных представляют древовидную структуру.

# Доступ к элементам в узлах осуществляется через точку - к данным можно обращаться либо указывая *имя узла* в ссылке, либо через *текущий 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 ZTEST\_ST

<?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 - можно определить *текущий узел* /команда *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*

<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>

*Rem*

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**

### **Определение текстового шаблона**

 <element [attr] [tt:ref=″node″>]

                [tt:lax=lax\_flag]

                [tt:extensible=extensible\_flag]>

  ...

</element>

*Rem*

XML элементы и атрибуты не должны лежать в пространстве имен ST команд /без префикса tt:/.

**tt:ref** - ссылка на текущий узел.

**tt:lax** - правило, по которому реагирует интерпретатор на несоответствие XML документа /относительно имён XML элементов и наличия атрибутов / при десериализации /XML -> ABAP/

* *off* – XML атрибут во вх. потоке must have same имя и атрибуты, что и в ST программе.
* *on* – XML элемент во вх. потоке может иметь любое имя, но при этом атрибуты и их содержимое должно совпадать.
* *deep-on* | *deep-off* – те же правила, включая вложенные элементы.

**tt:extensible** - правило, по которому реагирует интерпретатор на несоответствие вложенности и наличия др. XML элементов, не описанных статически в ST программе

* *on* – элемент XML can have вложенные элементы, явно не определенные в ST программе.
* *deep-static* | *deep-dynamic* – текущий элемент и его вложенные элементы can have неопределенные вложенные элементы во вх. 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**

Текст не является элементом XML документа и в шаблон can be вставлен двумя способами

1. как текст - …>*text*<… - текст будет вставлен в документ XML, но проигнорирован в случае наличия в нем только пробелов;
2. через ST команду - …><tt:text>text</tt:text><… - текст не игнорируется.

**Сериализация** /ABAP -> XML/ - *если текст не был проигнорирован*, все его символы /включая пробелы/ вставляются в создающийся XML документ.

**Десериализация** - /XML -> ABAP/ - 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 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/десериализации *элементарных типов* используется команда tt:value

*<tt:value* [ref="node"] [map="mapping\_list"] [length|minLength|maxLength="length"] [validation] />

где

*ref* – определяет текущий узел /must be элементарного типа/.

Элементарные типы отображаются в XML и обратно согласно **asXML** формату /определяет соответствие элементарных типов ABAP и типов данных XML

|  |  |
| --- | --- |
| **ABAP тип** | **XML Scheme тип** |
| b | xsd:unsignedByte |
| s | xsd:short |
| i | xsd:int |
| p | xsd:decimal |
| decfloat16 | precisionDecimal, totalDigits = 16 |
| decfloat34 | precisionDecimal, totalDigits = 34 |
| f | xsd:double |

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

|  |  |
| --- | --- |
| **Домент** | **XML Schema тип** |
| XSDBOOLEAN | boolean |
| XSDDATE\_D | date |
| XSDDATETIME\_Z, XSDDATETIME\_LONG\_Z, XSDDATETIME\_OFFSET, XSDDATETIME\_LOCAL | dateTime |
| XSDLANGUAGE | language |
| XSDQNAME | QName |
| XSDTIME\_T | time |
| XSDUUID\_RAW, XSDUUID\_CHAR | UUID |

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)* – при сериализации;
* *xml(x1…xn) > val(a)* – при десериализации;
* *xml(x) = val(a)* или *val(a) = xml(x)* –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* | *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”] …

где

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

*xsd-maxInclusive* – max допустимое значение, включая указанное,

*xsd-maxExclusive* – max допустимое число не включая указанное.

*xsd-totalDigits*, *fractionDigits* – max 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.

В данном примере допустимо только преобразования чисел в диапазоне от 30000 до 32767 /предел типа Short/.

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

Атрибуты *minLength* | *maxLength* | *length* можно задать для узлов данных | переменных с типами с, x, string и xstring /для всех остальных типов данные ограничения игнорируются/.

**minLength**

* *Сериализация* /ABAP -> XML/ - если будет передано значение с меньшей длинной, недостающие до length символы будут заполнены справа либо пробелами | нулевым байтом /в зависимости от типа/.
* *Десериализаци*я /XML -> ABAP/ - атрибут minLength игнорируется.

#### **Обработка таблиц**

Для обработки таблиц служит команда tt:loop

<tt:loop [ref=“node“] [name=“alias“]>

  ...

</tt:loop>

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/десериализацию для всех типов данных, *кроме ссылочных переменных*. Данная команда запускает XSL трансформацию и преобразует данные в формат asXML. Так, например, можно 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

<tt:assign [to-ref="node"|to-var="variable"] [ref="node"|val="value"|var="variable"] />

где

*to-ref* и *to-var* - куда передать значение,

*ref*, *val* и *var* – какое значение передавать.

В качестве полей источников/назначения могут выступать

* Узлы данных – *ref* /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.

### **Вывод значений переменных /параметров/**

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

<tt:write var="variable" [map="mapping\_list"] [length|minLength=″length″] [validation] />

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>

Результат - <X>333</X>. При десериализации /XML -> ABAP/ данная команда игнорируется.

### **Чтение значений XML элементов**

Во время выполнения десериализации с помощью команды **tt:read** можно получить значение XML атрибута

<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. 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>

Если входной XML будет таким - <X>333</X>, узлу ROOT будет назначено значение 333. Данная команда игнорируется при сериализации.

## **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

<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**

|  |  |
| --- | --- |
| **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.

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

* *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

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

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

|  |  |
| --- | --- |
| Оператор |  |
| = |  |
| != |  |
| >, &gt; |  |
| >=, &gt;= |  |
| &lt; | > |
| &lt;= | ≥ |

Как видно из таблицы, оператор < в XML всегда должен быть заменен &lt; Оператор > может быть заменен на &gt; но можно и не менять.

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.

Если в предыдущую программу передать следующий XML

<roots>

  <root1>

  </root1>

</roots>

Её обработка вызовет ошибку, но если добавить в неё пустой шаблон, ошибки не произойдет

<tt:cond>

<tt:empty/>

</tt:cond>

# Simple Transformations в SAP

# <https://signatov.com/primer-ispolzovania-simple-transformations-v-sap/>

Simple Transformation - ST - это [проприетарный язык программирования](https://en.wikipedia.org/wiki/Programming_language) разработанный вендором для описания внутренностей SAP посредством XML.

**Сериализация** - преобразование ABAP-данных в XML.

**Десереализация** - обратный процесс .



<http://sapblog.rmtiwari.com/2009/02/discovering-hidden-gem-generate-simple.html>

### 

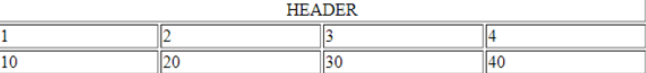
### <http://sapblog.rmtiwari.com/2009/04/generate-simple-transformation-for-xml.html>

# Simple Transformation парсинг HTML таблицы

<https://sapcode.ru/simple-transformations-%D0%BF%D0%B0%D1%80%D1%81%D0%B8%D0%BD%D0%B3-html-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8B/>

[Содержание](#Содержание)

***Файл test.html***



<table border="1" width="600">

<tr>

   <td colspan="4" align="center">HEADER</td>

</tr>

<tr>

   <td>1</td>

   <td>2</td>

   <td>3</td>

   <td>4</td>

</tr>

<tr>

   <td>10</td>

   <td>20</td>

   <td>30</td>

   <td>40</td>

</tr>

</table>

**Структура ZHTML\_S** /тр. **se11**/

@EndUserText.label : 'ZHTML\_SCTR'

@AbapCatalog.enhancementCategory : #NOT\_CLASSIFIED

define type **zhtml\_sctr** {

col\_1 : abap.string(0);

col\_2 : abap.string(0);

col\_3 : abap.string(0);

col\_4 : abap.string(0);

}

**Table type ZHTML\_T**

Line type = ZHTML\_S

**Трансформация ZHTML\_TRFN** /тр. **strans**/

<?sap.transform simple?>

<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:root name="RESULT" type="ddic:ZHTML\_T"/>

<tt:template>

*<RESULT>*

<tt:loop ref=".RESULT">

*<ZHTML\_S>*

<COL\_1 tt:value-ref="COL\_1"/>

<COL\_2 tt:value-ref="COL\_2"/>

<COL\_3 tt:value-ref="COL\_3"/>

<COL\_4 tt:value-ref="COL\_4"/>

*</ZHTML\_S>*

</tt:loop>

*</RESULT>*

</tt:template>

</tt:transform>

Необходимо привести теги в соответствии с таблицей HTML. Важно помнить, что они регистрозависимые.

Также необходимо пропустить первый тег <TR> т.к. в нем находится строка  заголовка - <tt:skip count=”1″ name=”tr”/>

<?sap.transform simple?>

<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:root name="RESULT" type="ddic:ZHTML\_T"/>

<tt:template>

*<table>*

<tt:skip count=”1″ name=”tr”/>

<tt:loop ref=".RESULT">

*<tr>*

<td tt:value-ref="COL\_1"/>

<td tt:value-ref="COL\_2"/>

<td tt:value-ref="COL\_3"/>

<td tt:value-ref="COL\_4"/>

*</tr>*

</tt:loop>

*</table>*

</tt:template>

</tt:transform>

REPORT **ZHTML\_READ**.

DATA lt\_res TYPE ZHTML\_TT.

DATA lv\_html\_data TYPE string. *" сюда подаем HTML таблицу*

field-symbols: <fs\_dfies> like line of lt\_res.

SELECTION-SCREEN BEGIN OF BLOCK 002.

PARAMETERS : p\_aps LIKE rlgrap-filename MODIF ID pth .

SELECTION-SCREEN END OF BLOCK 002.

DATA : path\_name(150) TYPE C.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR p\_aps.

SUBMIT rs\_get\_f4\_dir\_from\_applserv AND RETURN.

IMPORT path\_name FROM MEMORY ID 'PATH\_NAME\_SDL'.

p\_aps = path\_name.

TRY.

lv\_html\_data = p\_aps.

CALL TRANSFORMATION ZHTML\_PRS

SOURCE XML lv\_html\_data

RESULT result = lt\_res[].

loop at lt\_res assigning <fs\_dfies>.

WRITE: / <fs\_dfies>-col\_1.

endloop.

CATCH cx\_root.

ENDTRY.

# Генерация XML при помощи Simple Transformation

<https://sapcode.ru/%d0%b3%d0%b5%d0%bd%d0%b5%d1%80%d0%b0%d1%86%d0%b8%d1%8f-xml-%d0%bf%d1%80%d0%b8-%d0%bf%d0%be%d0%bc%d0%be%d1%89%d0%b8-simple-transformation/>

[Содержание](#Содержание)

**Структура zzz\_s\_xml**

@EndUserText.label : 'zzz\_s\_xml'

@AbapCatalog.enhancementCategory : #NOT\_CLASSIFIED

define type **zzz\_s\_xml** {

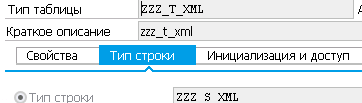
name : abap.char(50);

age : abap.int1;

position : abap.char(50);

}

**Table type** **zzz\_t\_xml**



**Трансформация zzz\_test\_xml\_generate**

<?sap.transform simple?>

<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:root name="ROOT" type="?"/>

<tt:root name="PEOPLE" type="ddic:ZZZ\_T\_XML"/>

<tt:template>

<PEOPLE>

<tt:loop ref=".PEOPLE">

*<ZZZ\_S\_XML>*

<NAME tt:value-ref="NAME"/>

<AGE tt:value-ref="AGE"/>

<POSITION tt:value-ref="POSITION"/>

*</ZZZ\_S\_XML>*

</tt:loop>

</PEOPLE>

</tt:template>

</tt:transform>

REPORT **ZZZ\_P\_XML**.

DATA lt\_file TYPE sdokcntbins.

DATA lv\_len TYPE i.

DATA(lt\_people) = VALUE zzz\_t\_xml(

( name = 'Иванов И.И.' age = 20 position = 'Менеджер' )

( name = 'Петров П.П.' age = 25 position = 'Старший Менеджер' )

( name = 'Сидоров С.С.' age = 25 position = 'Руководитель' )

).

CALL TRANSFORMATION zzz\_test\_xml\_generate

SOURCE people = lt\_people

RESULT XML DATA(lv\_xml).

*“ lv\_xml -> lt\_file*

CALL FUNCTION 'SCMS\_XSTRING\_TO\_BINARY'

EXPORTING

buffer = lv\_xml

IMPORTING

output\_length = lv\_len

TABLES

binary\_tab = lt\_file.

CONSTANTS c\_file TYPE string VALUE 'C:\Users\a.ekimenko\example.xml'.

write: / 'lv\_len = ', lv\_len.

*“lt\_file -> c\_file*

cl\_gui\_frontend\_services=>gui\_download(

EXPORTING

bin\_filesize = lv\_len " File length for binary files

filename = c\_file " Name of file - CHAR(128)

filetype = 'BIN' " File type (ASCII, binary ...)

CHANGING

data\_tab = lt\_file " Transfer table

EXCEPTIONS

OTHERS = 1

).

IF sy-subrc <> 0.

MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.

RETURN.

ENDIF.

*Rem*

**SDOKCNTBIN** - is an SAP Structure that can be used to process ***SDOK: Line of binary document content for WWW server*** Information within sap ABAP programs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Field* | *Component Type* | *Data Type* | *length (Decimals)* | *Description* |
| LINE | [SDOK\_SDATX](https://www.se80.co.uk/dataelements/s/sdok/sdok_sdatx.htm) | RAW | 1022 | Line for binary document contents, length for upload |

[**GUI\_DOWNLOAD**](https://www.sapdatasheet.org/abap/func/gui_download.html)- Download an Internal Table to the PC.

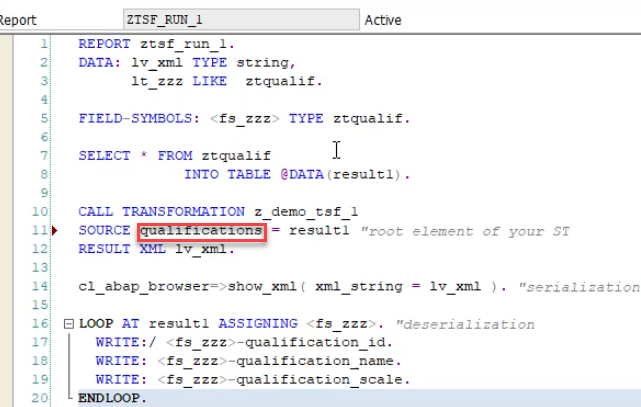
### **Чтение XML средствами ABAP**

<http://zabapstory.blogspot.com/2013/10/xml-abap.html>

Можно написать трансформацию которая будет парсить XML и складывать данные в таблицу но мы пойдем обходным путем - будем разбирать XML функционалом ABAP.

# [Parse an xml file and insert values in ABAP](https://wiki.scn.sap.com/wiki/display/ABAP/Parse+an+xml+file+and+insert+values+in+ABAP)

<https://wiki.scn.sap.com/wiki/display/ABAP/Parse+an+xml+file+and+insert+values+in+ABAP>



**http\_post**

[Содержание](#Содержание)

# Usage of the ABAP REST Library [SAP\_BASIS 7.40]

<https://blogs.sap.com/2013/05/16/usage-of-the-abap-rest-library-sapbasis-740/>

# General Notes on Using the ABAP REST Library

<https://help.sap.com/viewer/753088fc00704d0a80e7fbd6803c8adb/7.4.23/en-US/c4772c15d6454ade967e52935f0ea44c.html>

*Internet Connection Framework* - **ICF**

**HTTP protocol** - the URI schema is http or https.

**URI**

**URI query parameters** *can be part of the identification of a resource, but are not preferred*.

***Типы uri***

1. The ***base URI*** - <http://ldcigmd:50055/sap/bc/rest_cars/>.
2. The ***resource URI*** - [http://ldcigmd:50055/sap/bc/rest\_cars/Cars/17*/*](http://ldcigmd:50055/sap/bc/rest_cars/Cars/17/).

*Rem*

It is preferred to have single entities identified by positional URI path elements*, and not defined as* query parameters /as *name-value-pairs templates*/ - in this way, clients are more decoupled from servers because no knowledge about URI templates of the server is required.

**POST** **or** **GET?**

*Execution* of functions offered by a service, *should make use of the* ***POST*** *method -* you may use the **GET** method *to retrieve inf. of a function prior its call* - for example, reading a form document for the function input parameters/.

**SREST package**

If you want to use the REST library, *you have to declare an access usage for* **SREST** *package* interface.

**MIME types** - XML, JSON, etc.

**Типы клиентов**

1. CL\_REST\_**HTTP**\_CLIENT- to *communicate* with the REST application using *the HTTP protocol*.
2. CL\_REST\_**LOCAL**\_CLIENT - is the local version of the HTTP class, which enables a REST client application *to access REST application locally on the same ABAP system* without using any HTTP communication.

**Способы создания клиента**

1. Using destination configuration - CL\_HTTP\_CLIENT=>*CREATE\_BY\_DESTINATION* /Logon data, etc. is stored by transaction SM59 already in a secure manner and the REST client implementation can rely on this/.
2. Via configuration provided by source code - CL\_HTTP\_CLIENT=>*CREATE\_BY\_URL* /If required, authentication inf. can be set on the returned object - for example, using the AUTHENTICATE method/.

**Способы использования http-объекта**

1. *Create a new HTTP object* for each REST request in a sequence of REST client requests.
2. *Re-use* a stateless *HTTP object* in subsequent REST client calls.

***The URL of a REST client can be changed*** by setting the request *header ~request\_uri* to the new URI.

Use the method *IF\_REST\_CLIENT~SET\_REQUEST\_HEADER* to set this header to the URI to be used. We recommend to use the constant *IF\_HTTP\_HEADER\_FIELDS\_SAP=>REQUEST\_URI* as header name.

# REST Processing

The REST client processing is done *using a class implementing* **IF\_REST\_CLIENT** - on construction of this object, you pass a previously constructed HTTP client object.

For methods requiring an input – POST | PUT, *you use an object with interface* **IF\_REST\_ENTITY**, which has to be constructed prior to the call

* the REST library offers the class **CL\_REST\_ENTITY**, implementing IF\_REST\_ENTITY;
* use REST client method **IF\_REST\_CLIENT~CREATE\_REQUEST\_ENTITY** to create a new instance of this type for each call.

# CSRF Protection

On change requests - PUT, POST, and DELETE of REST clients to an ABAP server, the client has to provide a **CSRF** - *Cross-Site Request Forgery* **token**.

Such a token can be retrieved via a previous service call to the ABAP server - for this,

1. first on a none-changing call - GET, HEAD, OPTIONS, the client has to get this token by setting the *HTTP header X-CSRF-Token* to the value *Fetch*.

lo\_rest\_client->set\_request\_field(

name = if\_rest\_request=>*gc\_header\_csrf\_token*

value = '*Fetch*'

).

1. A CSRF token is returned by the ABAP server in the same header and can be used for subsequent, server state changing calls using header X-CSRF-Token.

If this header is not present on a server state changing REST call, the server will respond with a HTTP 403 -*Forbidden* return code, the HTTP header is set to *Required* and an error text - for example, *CSRF token validation failed i*s returned.

On a successful response of a call with the fetch value, the returned CSRF token can be kept for later state changing REST requests

data: *lv\_csrf\_token* type string.

lv\_csrf\_token = lo\_rest\_client->get\_header\_field( name = if\_rest\_request=>gc\_header\_csrf\_token ).

***The validity of the CSRF token*** depends on the release of the ABAP component SAP\_BASIS and on the activation of the security session management /which is controlled via the transaction SICF\_SESSIONS on the granularity of SAP clients/

Release >= 7.03/7.31, the validity is bound to the security session, which depends on the system parameter *http/security\_session\_timeout value* /see transaction RZ11 for details on this parameter/. By default, the security session management is active in these releases.

# REST Programming Tutorial

*\*The resources of type Car*

types:

begin of car\_s,

ID type i,

MODEL\_NAME type string,

MANU\_ID type i,

PRICE type REST\_PRICE, *" type P, decimals 2*

CURRENCY type ISOCD,

MODEL\_YEAR type REST\_YEAR, *" type N, length 4*

IMAGE\_TYPE type string,

IMAGE\_PATH type string,

UPDATE\_USER type UNAME, *" type C, length 12*

UPDATE\_TIMESTAMP type TZNTSTMPL, *" type DEC, length 21, decimals* 7

end of car\_s.

# Attaching ABAP Processor Classes

All resource URIs served by a REST service must be assigned via URI templates to ABAP processing classes.

The simplest way to create an ABAP REST application is to implement an ABAP class derived from the library class **CL\_REST\_HTTP\_HANDLER**. This class implements the interfaces IF\_REST\_APPLICATION and IF\_HTTP\_EXTENSION.

This is the coding of the method *IF\_REST\_APPLICATION~GET\_ROOT\_HANDLER* for attaching processor classes to URI templates

method ***if\_rest\_application~get\_root\_handler***.

*\* Creates an instance of the REST library router class CL\_REST\_ROUTER.*

data lo\_handler type *ref to* cl\_rest\_router.

create object lo\_handler.

*\* Several resource-URI templates* /CL\_REST\_SAMPLE\_CAR | CL\_REST\_SAMPLE\_CARS/ *get attached*

*\* to ABAP processing classes*

lo\_handler->*attach*(

*\* The resource template /Car/{ID:[1-9][0-9]\*} identifies a resource entity of type Car by an integer number*

*\* greater than 1*

iv\_template = '/Car/*{ID:[1-9][0-9]\*}*'

iv\_handler\_class = 'CL\_REST\_SAMPLE\_CAR' ).

lo\_handler->*attach*(

iv\_template = '/Cars'

iv\_handler\_class = 'CL\_REST\_SAMPLE\_CARS' ).

ro\_root\_handler = lo\_handler.

endmethod.

# Enabling the Service via the ICF

<https://help.sap.com/viewer/753088fc00704d0a80e7fbd6803c8adb/7.4.23/en-US/042eef6e68964e6682e48d7279386189.html>

# The Class CL\_REST\_SAMPLE\_CAR

The processor class CL\_REST\_SAMPLE\_CAR is intended for the processing of single entities of type Car.

In the constructor of the processor class, the member *MV\_CONDITIONAL* is set to value ABAP\_TRUE indicating, that conditional handling is required for the resources served by this class.

method ***constructor***.

super->constructor( ).

mv\_conditional *= abap\_true*.

endmethod.

− Как с помощью АБАП выдернуть с интернет-ресурса текущий курс доллара?

− **CALL FUNCTION 'HTTP\_GET'** позволяет выдернуть страницу с сайта во внутреннюю таблицу SAP.

Дальше пишите парсинг этой красоты и выуживаете из него курсы. Как показывает практика такие страницы формируются автоматически, так что и ваша обработка будет корректная.

### **Курсы валют**

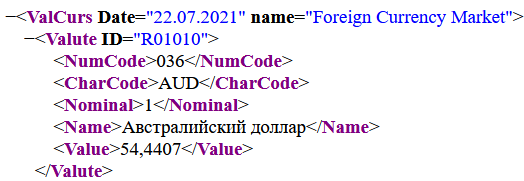
### **Курсы валют. Часть 1.**

<http://zabapstory.blogspot.com/2013/10/1.html>

[Содержание](#Содержание)

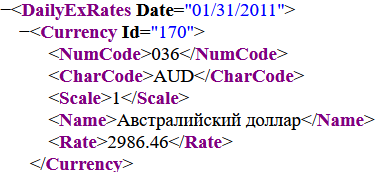
В первой части приведу пример отчета, который считывает курсы валют на дату с сервера ЦБ РФ, конвертирует трансформацией во внутреннюю таблицу и средствами BAPI актуализирует курсы валют в системе.

http://www.cbr.ru/scripts/XML\_daily.asp?date\_req=22/07/2021



**Код трансформации**







Xlst transformation Multiple markers at this line type standart-t is not atomic changed lines, added



simpleTransformation System expected end of attribute

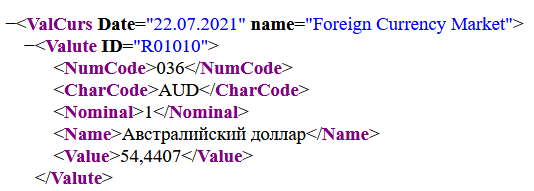
report **ztest\_forcurr**.

types: begin of t\_fline,

line(255) type c,

end of t\_fline.

types: t\_fline\_tab type standard table of t\_fline.



~~types: begin of t\_valute,~~

~~numcode(20),~~

~~charcode(20),~~

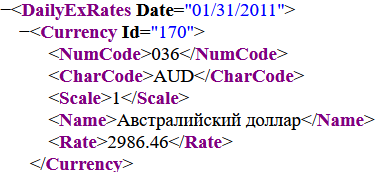
~~nominal type vtb\_dfans-cffact,~~

~~name(150),~~

~~value type vtb\_dfans-value,~~

~~end of t\_valute.~~

https://www.nbrb.by/services/xmlexrates.aspx?ondate=01/31/2011



types: begin of t\_valute,

numcode(20),

charcode(20),

scale(20),

name(150),

rate(20),

end of t\_valute.

types: t\_data type sorted table of t\_valute with unique key numcode charcode.

types: t\_date\_str(10) type c .

start-of-selection.

perform load\_cbrf\_rates\_from\_site using sy-datum.

\*&---------------------------------------------------------------------\*

\*& Form LOAD\_CBRF\_RATES\_FROM\_SITE

\*&---------------------------------------------------------------------\*

form load\_cbrf\_rates\_from\_site using iv\_date type bldat.

data: lr\_client type ref to if\_http\_client,

lr\_response type ref to if\_http\_response,

l\_response type string,

l\_url type string,

lv\_datetext(10) type c,

lv\_url\_add(100) type c.

data: lv\_xml\_string type string,

lt\_data type t\_data,

lv\_date type t\_date\_str.

data: lv\_err\_string TYPE string,

lv\_ret\_code TYPE sy-subrc.

*\*Подставление нужной даты /не обязательно/*

clear lv\_datetext.

if not iv\_date is initial.

write iv\_date to lv\_datetext dd/mm/yyyy.

lv\_datetext+2(1) = lv\_datetext+5(1) = '/'.

concatenate '?date\_req=' lv\_datetext into lv\_url\_add.

endif.

*“ http://www.cbr.ru/scripts/XML\_daily.asp?date\_req=22/07/2021*

concatenate 'http://www.cbr.ru/scripts/XML\_daily.asp' lv\_url\_add into l\_url.

call method cl\_http\_client=>create\_by\_url

exporting url = l\_url *" 'http://www.cbr.ru/scripts/XML\_daily.asp'*

importing client = lr\_client

exceptions

internal\_error = 1

argument\_not\_found = 2

plugin\_not\_active = 3

others = 4.

if sy-subrc ne 0.

*" Ошибка обращения к серверу*

*" message e002 raising create\_url\_err.*

endif.

\* Отправка запроса

call method lr\_client->send

\* exporting timeout = timeout

exceptions

http\_communication\_failure = 1

http\_invalid\_state = 2

http\_processing\_failed = 3

others = 4.

if sy-subrc ne 0.

*" Ошибка запроса к серверу*

" message e003 raising send\_error.

endif.

\* Получение ответа сервера

call method lr\_client->receive

exceptions

http\_communication\_failure = 1

http\_invalid\_state = 2

http\_processing\_failed = 3

others = 4.

if sy-subrc ne 0.

*" Ошибка получения ответа от сервера*

*" message e004 raising receive\_error.*

endif.

lr\_response = lr\_client->response .

l\_response = lr\_response->get\_cdata( ).

\* Закрытие соединения

call method lr\_client->close

exceptions

http\_invalid\_state = 1

others = 2.

check not l\_response is initial.

lv\_xml\_string = l\_response.

try.

perform run\_transformation

using lv\_xml\_string

changing lt\_data lv\_date.

catch cx\_root.

\* if sy-subrc ne 0.

\* "Ошибка преобразования XML

\* message e005 raising trans\_error.

\* endif.

endtry.

perform check\_rates changing lt\_data.

\* perform create\_exchrate using lt\_data lv\_date.

endform. *"load\_cbrf\_rates\_from\_site*

\*&---------------------------------------------------------------------\*

\*& Form run\_transformation

\*&---------------------------------------------------------------------\*

form run\_transformation using iv\_xml

type string

changing et\_data type t\_data ev\_date type t\_date\_str.

data: oref type ref to cx\_root,

xslt\_message type string.

try.

call transformation zm\_cbrf\_rates

source xml iv\_xml

result valcurs = et\_data[] gdate = ev\_date.

catch cx\_xslt\_exception into oref.

xslt\_message = oref->get\_text( ).

message xslt\_message type 'E' raising trans\_error.

catch cx\_st\_match\_element into oref.

xslt\_message = oref->get\_text( ).

message xslt\_message type 'E' raising trans\_error.

catch cx\_root into oref.

xslt\_message = oref->get\_text( ).

message xslt\_message type 'E' raising trans\_error.

endtry.

endform. *"run\_transformation*

\*&---------------------------------------------------------------------\*

\*& Form check\_rates

\*&---------------------------------------------------------------------\*

form check\_rates changing ct\_data type t\_data.

data: begin of ls\_waers,

waers type waers\_curc,

end of ls\_waers,

lt\_waers like standard table of ls\_waers,

ls\_data like line of ct\_data.

select waers from tcurc into table lt\_waers.

loop at ct\_data into ls\_data.

read table lt\_waers with key waers = ls\_data-charcode transporting no fields.

if sy-subrc is not initial.

delete table ct\_data from ls\_data.

endif.

endloop.

endform. *"check\_rates*

\*&---------------------------------------------------------------------\*

\*& Form create\_exchrate

\*&---------------------------------------------------------------------\*

form create\_exchrate

using it\_data type t\_data iv\_date type t\_date\_str.

data: ls\_data like line of it\_data,

ls\_rec type bapi1093\_0,

ls\_rec2 type bapi1093\_0,

lt\_rec type standard table of bapi1093\_0,

lt\_return type standard table of bapiret2,

ls\_return like line of lt\_return,

lv\_usd2\_kurs type p length 9 decimals 4.

loop at it\_data into ls\_data.

replace all occurrences of ',' in ls\_data-value with '.'.

clear ls\_rec.

ls\_rec-rate\_type = 'M'.

ls\_rec-from\_curr = ls\_data-charcode.

ls\_rec-to\_currncy = 'RUB'.

ls\_rec-valid\_from = iv\_date+6(4).

ls\_rec-valid\_from+4(2) = iv\_date+3(2).

ls\_rec-valid\_from+6(2) = iv\_date(2).

ls\_rec-exch\_rate = ls\_data-value.

ls\_rec-from\_factor = ls\_data-nominal.

ls\_rec-to\_factor = '1'.

if ls\_data-charcode = 'USD'. *"#EC NOTEXT*

move ls\_rec to ls\_rec2.

ls\_rec2-from\_curr = 'USD2'.

ls\_rec2-exch\_rate = ls\_rec-exch\_rate

\* '1.02'.

lv\_usd2\_kurs = ls\_rec2-exch\_rate.

ls\_rec2-exch\_rate = lv\_usd2\_kurs.

append ls\_rec2 to lt\_rec.

endif.

append ls\_rec to lt\_rec.

endloop.

perform add\_new\_conversion tables lt\_rec.

call function 'BAPI\_EXCHRATE\_CREATEMULTIPLE'

exporting upd\_allow = 'X' \* CHG\_FIXED = ' ' \* DEV\_ALLOW = '000'

tables exchrate\_list = lt\_rec return = lt\_return .

read table lt\_return into ls\_return with key type = 'E'.

if sy-subrc = 0.

message id ls\_return-id type ls\_return-type number ls\_return-number with ls\_return-message\_v1 ls\_return-message\_v2 ls\_return-message\_v3 ls\_return-message\_v4.

call function 'BAPI\_TRANSACTION\_ROLLBACK'.

else.

call function 'BAPI\_TRANSACTION\_COMMIT'.

endif.

endform. *"create\_exchrate*

\*&---------------------------------------------------------------------\*

\*& Form add\_new\_conversion

\*&---------------------------------------------------------------------\*

form add\_new\_conversion tables ct\_rec structure bapi1093\_0. data: ls\_sap\_data like line of ct\_rec, lt\_convers type standard table of tcurf, ls\_curf type tcurf. data: lv\_inv\_date type gdatu\_invv. loop at ct\_rec into ls\_sap\_data. " get conversion customizing on date select \* from tcurf up to 1 rows into ls\_curf where kurst = ls\_sap\_data-rate\_type and fcurr = ls\_sap\_data-from\_curr and tcurr = ls\_sap\_data-to\_currncy and gdatu ge lv\_inv\_date order by gdatu descending. exit. endselect. lv\_inv\_date = ls\_sap\_data-valid\_from. translate lv\_inv\_date using '09182736455463728190'. if sy-subrc is not initial or ls\_curf-ffact <> ls\_sap\_data-from\_factor or ls\_curf-tfact <> ls\_sap\_data-to\_factor. clear ls\_curf. ls\_curf-kurst = ls\_sap\_data-rate\_type. ls\_curf-fcurr = ls\_sap\_data-from\_curr. ls\_curf-tcurr = ls\_sap\_data-to\_currncy. ls\_curf-gdatu = lv\_inv\_date. ls\_curf-ffact = ls\_sap\_data-from\_factor. ls\_curf-tfact = ls\_sap\_data-to\_factor. insert tcurf from ls\_curf. if sy-subrc is not initial. modify tcurf from ls\_curf. endif. endif. endloop. endform. "add\_new\_conversion

# Calling an external RESTful service from ABAP – HTTP Method GET

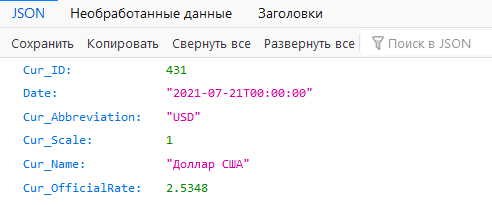
<https://blogs.sap.com/2014/11/09/calling-an-external-restful-service-from-abap-http-method-get/>

[Содержание](#Содержание)

Scenario - GET operation to access

[http://hostname.com:55400/vendavo/rest/agreements/XYZ](http://sapvnddev201.isus.emc.com:55400/vendavo/rest/agreements/370743634-13000)

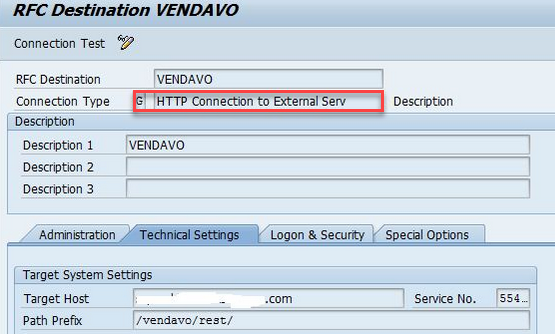
[https://www.nbrb.by*/api/exrates/rates/840*?parammode=1](https://www.nbrb.by/api/exrates/rates/840?parammode=1)



**Create a RFC destination** /**sm59**/

of type /HTTP Connections to External Server/

Menu -> Tools → Administration → Administration → Network → RFC Destinations.



**content-type *application/json***

DATA: lo\_http\_client TYPE REF TO if\_http\_client,

          lo\_rest\_client TYPE REF TO cl\_rest\_http\_client,

         lv\_url TYPE string,

         lv\_body TYPE string,

         token TYPE string,

         agreements TYPE string,

*"Create a structure(or deep) that exactly matches your JSON response*

         abap\_response TYPE zca\_serno\_ln,

         lo\_response TYPE REF TO if\_rest\_entity.

*\* Create HTTP intance using RFC restination created*

*\* You can directly use the REST service URL as well*

   cl\_http\_client=>create\_by\_destination(

    EXPORTING

      destination              = *'VENDAVO'*    " Logical destination (specified in function call)

    IMPORTING

      client = lo\_http\_client    " HTTP Client Abstraction

    EXCEPTIONS

      argument\_not\_found       = 1

      destination\_not\_found    = 2

      destination\_no\_authority = 3

      plugin\_not\_active        = 4

      internal\_error           = 5

      OTHERS                   = 6

  ).

*\* If you are using cl\_http\_client=>create\_by\_url use this code to supress and pass your*

*\* HTTP basic authenication*

\*  lo\_http\_client->propertytype\_logon\_popup = lo\_http\_client->co\_disabled.

\*  DATA l\_username TYPE string.

\*  DATA l\_password TYPE string.

\*  l\_username = 'user'.

\*  l\_password = 'password'.

\*  CALL METHOD lo\_http\_client->authenticate

\*    EXPORTING

\*      username = l\_username

\*      password = l\_password.

\* Create REST client instance

   CREATE OBJECT lo\_rest\_client

     EXPORTING

       io\_http\_client = lo\_http\_client.

\* Set HTTP version

   lo\_http\_client->request->set\_version( if\_http\_request=>co\_protocol\_version\_1\_0 ).

   IF lo\_http\_client IS BOUND AND lo\_rest\_client IS BOUND.

     DATA(id) = 'XYZ'.

     CONCATENATE 'agreements/' id INTO lv\_url.

\* Set the URI if any

     cl\_http\_utility=>set\_request\_uri(

       EXPORTING

         request = lo\_http\_client->request    " HTTP Framework (iHTTP) HTTP Request

         uri     = lv\_url                     " URI String (in the Form of /path?query-string)

     ).

\* Set request header if any

     CALL METHOD lo\_rest\_client->if\_rest\_client~set\_request\_header

       EXPORTING

         iv\_name  = 'auth-token'

         iv\_value = token.

\* HTTP GET

     lo\_rest\_client->if\_rest\_client~get( ).

\* HTTP response

     lo\_response = lo\_rest\_client->if\_rest\_client~get\_response\_entity( ).

\* HTTP return status

     DATA(http\_status)   = lo\_response->get\_header\_field( '~status\_code' ).

\* HTTP JSON return string

     DATA(json\_response) = lo\_response->get\_string\_data( ).

\* Class to convert the JSON to an ABAP sttructure

    DATA lr\_json\_deserializer TYPE REF TO cl\_trex\_json\_deserializer.

    CREATE OBJECT lr\_json\_deserializer.

    lr\_json\_deserializer->deserialize( EXPORTING json = json\_response IMPORTING abap = abap\_response ).

DATA: lo\_http\_client TYPE REF TO if\_http\_client,

          lo\_rest\_client TYPE REF TO cl\_rest\_http\_client,

         lv\_url TYPE string,

         lv\_body TYPE string,

         token TYPE string,

         agreements TYPE string,

*"Create a structure(or deep) that exactly matches your JSON response*

         abap\_response TYPE zca\_serno\_ln,

         lo\_response TYPE REF TO if\_rest\_entity.

*\* If you are using cl\_http\_client=>create\_by\_url use this code to supress and pass your*

*\* HTTP basic authenication*

lo\_http\_client->propertytype\_logon\_popup = lo\_http\_client->co\_disabled.

DATA l\_username TYPE string.

DATA l\_password TYPE string.

l\_username = 'belarusneft\a.ekimenko'.

l\_password = 'shm21987\*'.

CALL METHOD lo\_http\_client->authenticate

  EXPORTING

username = l\_username

password = l\_password.

*\* Create REST client instance*

   CREATE OBJECT lo\_rest\_client

     EXPORTING

io\_http\_client = lo\_http\_client.

*\* Set HTTP version*

   lo\_http\_client->request->set\_version( if\_http\_request=>co\_protocol\_version\_1\_0 ).

   IF lo\_http\_client IS BOUND AND lo\_rest\_client IS BOUND.

     DATA(id) = ' parammode=1'.

     CONCATENATE ' https://www.nbrb.by/api/exrates/rates/840?' id INTO lv\_url.

*\* Set the URI if any*

     cl\_http\_utility=>set\_request\_uri(

       EXPORTING

         request = lo\_http\_client->request    *" HTTP Framework (iHTTP) HTTP Request*

         uri     = lv\_url                     *" URI String (in the Form of /path?query-string)*

     ).

*\* Set request header if any*

     CALL METHOD lo\_rest\_client->if\_rest\_client~set\_request\_header

       EXPORTING

         iv\_name  = 'auth-token'

         iv\_value = token.

\* HTTP GET

     lo\_rest\_client->if\_rest\_client~get( ).

\* HTTP response

     lo\_response = lo\_rest\_client->if\_rest\_client~get\_response\_entity( ).

\* HTTP return status

     DATA(http\_status)   = lo\_response->get\_header\_field( '~status\_code' ).

\* HTTP JSON return string

     DATA(json\_response) = lo\_response->get\_string\_data( ).

\* Class to convert the JSON to an ABAP sttructure

    DATA lr\_json\_deserializer TYPE REF TO cl\_trex\_json\_deserializer.

    CREATE OBJECT lr\_json\_deserializer.

    lr\_json\_deserializer->deserialize( EXPORTING json = json\_response IMPORTING abap = abap\_response ).

### [**Discovering a Hidden Gem :Generate Simple Transformation for XML in ABAP**](http://sapblog.rmtiwari.com/2009/02/discovering-hidden-gem-generate-simple.html)

<http://sapblog.rmtiwari.com/2009/02/discovering-hidden-gem-generate-simple.html>

### [**Generate Simple Transformation for XML in ABAP - Part II**](http://sapblog.rmtiwari.com/2009/04/generate-simple-transformation-for-xml.html)

<http://sapblog.rmtiwari.com/2009/04/generate-simple-transformation-for-xml.html>

## [**Simple XML Processing in ABAP Part I. – Overview**](http://abapmentor.expertise-team.com/post/simple-xml-processing-in-abap-part-i-overview.aspx)

<https://abapmentor.expertise-team.com/post/simple-xml-processing-in-abap-part-i-overview.aspx>

[Содержание](#Содержание)

In ABAP the XML document parsing is not such simple as in C# using **LINQ** expressions.

Here is the sample XML file that we are going to import to an SAP database table

<?xml version="1.0" encoding="utf-8"?>

<CUSTOMERS>

  <CUSTOMER>

    <CUSTOMER\_ID>*C0001*</CUSTOMER\_ID>

    <FIRST\_NAME>Adam</FIRST\_NAME>

    <LAST\_NAME>Rooney</LAST\_NAME>

    <MOBILE\_PHONE>+1-202-555-0114</MOBILE\_PHONE>

    <PERSONAL\_EMAIL>OIncithe1976@superrito.com</PERSONAL\_EMAIL>

    <CURRENT\_ADDRESS>CO84214 South Park, 84 Bigboned Way</CURRENT\_ADDRESS>

    <BIRTH\_DATE>19761203</BIRTH\_DATE>

  </CUSTOMER>

  <CUSTOMER>

    <CUSTOMER\_ID>*C0002*</CUSTOMER\_ID>

    <FIRST\_NAME>Robert</FIRST\_NAME>

    <LAST\_NAME>Sandy</LAST\_NAME>

    <MOBILE\_PHONE>+1-202-555-0150</MOBILE\_PHONE>

    <PERSONAL\_EMAIL>FApere1969@gustr.com</PERSONAL\_EMAIL>

    <CURRENT\_ADDRESS>DC20021 Washington, 821 Zimbabwe Ave</CURRENT\_ADDRESS>

    <BIRTH\_DATE>19690627</BIRTH\_DATE>

  </CUSTOMER>

  <CUSTOMER>

    <CUSTOMER\_ID>*C0003*</CUSTOMER\_ID>

    <FIRST\_NAME>Julie</FIRST\_NAME>

    <LAST\_NAME>Cony</LAST\_NAME>

    <MOBILE\_PHONE>+1-202-555-0127</MOBILE\_PHONE>

    <PERSONAL\_EMAIL>Baid1976@cuvox.de</PERSONAL\_EMAIL>

    <CURRENT\_ADDRESS>WA98052 Redmond, 1 Microsoft Way</CURRENT\_ADDRESS>

    <BIRTH\_DATE>19760213</BIRTH\_DATE>

  </CUSTOMER>

  <CUSTOMER>

    <CUSTOMER\_ID>*C0004*</CUSTOMER\_ID>

    <FIRST\_NAME>Andrew</FIRST\_NAME>

    <LAST\_NAME>O'Connor</LAST\_NAME>

    <MOBILE\_PHONE>+1-202-555-0139</MOBILE\_PHONE>

    <PERSONAL\_EMAIL>Venthic1986@jourrapide.com</PERSONAL\_EMAIL>

    <CURRENT\_ADDRESS>NY10041 In the Sky, 194 Deelux Apartments</CURRENT\_ADDRESS>

    <BIRTH\_DATE>19860912</BIRTH\_DATE>

  </CUSTOMER>

</CUSTOMERS>

First, 1) we are going to *create ABAP Dictionary objects* in ***SE11***, then 2) we are going to *construct a Simple Transformatio*n in the transaction ***STRANS***, and at last 3) we are going to *build a framework program* in ***SE38***. Then later, 4) we will satisfy some other new requirements also, such as *processing XML attributes*.

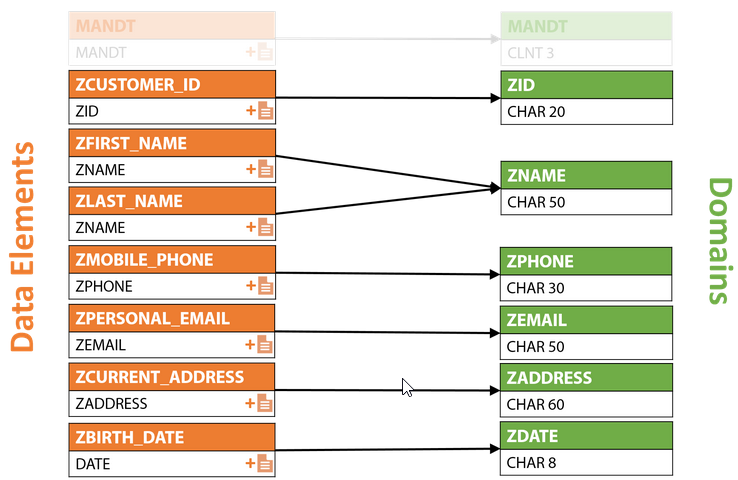


## **[Simple XML Processing in ABAP Part II. – Create ABAP Dictionary Objects](http://abapmentor.expertise-team.com/post/simple-xml-processing-in-abap-part-ii-create-abap-dictionary-objects.aspx)**

http://abapmentor.expertise-team.com/post/simple-xml-processing-in-abap-part-ii-create-abap-dictionary-objects.aspx

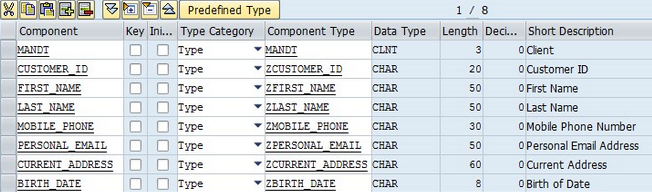
[Содержание](#Содержание)

To create ABAP Dictionary objects, we need some basic building blocks like ***Domains*** and ***Data Elements***.



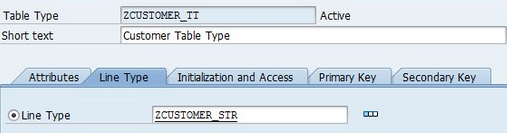
***Create a simple structure in the ABAP Dictionary***

Our next task is to ***create a simple structure*** *ZCUSTOMER\_STR* ***in the ABAP Dictionary***. With this structure, *we define the structure of the XML document*. I highly recommend you to create and use domains and data elements instead of ABAP built-in types to push lower the further maintenance costs.



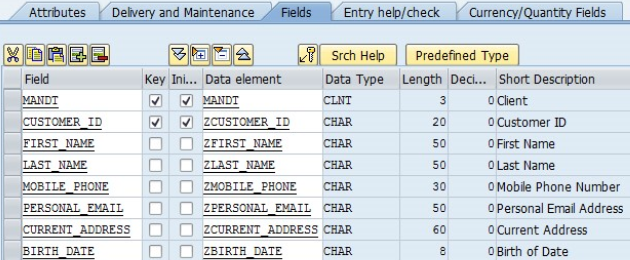
***Wrap the structure into a table type***

Now, we will ***wrap it into a table type***, called *ZCUSTOMER\_TT*. We have to set the line type, which is going to be the previously prepared structure, *ZCUSTOMER\_STR*. As you could see, the XML file contains the same structure in several times - <CUSTOMER></CUSTOMER>, so we have to use this table type, because *we will store the same structure of the data in many times in an internal table*.



***Create a database table***

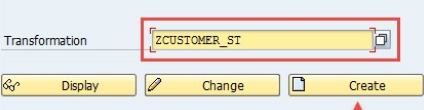
Our next task is to ***create a database table*** *~~SAPABAP1./BI0/~~ZCUSTOMER* ***in the ABAP Dictionary***. If we use domains and data elements instead of ABAP built-in types we can save a lots of time with maintenance tasks. Here *we have to set the key fields*, the *MANDT*and the *CUSTOMER\_ID*.

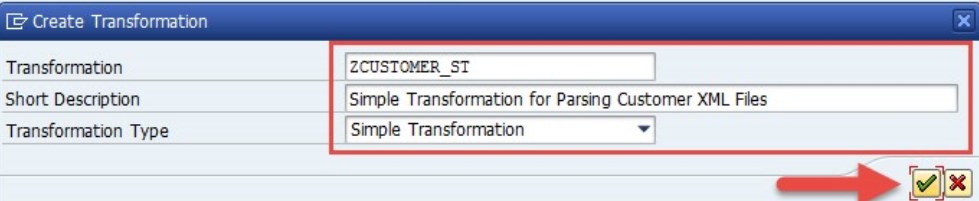


## **[Simple XML Processing in ABAP Part III. – Create Simple Transformation to Parse XML Content](http://abapmentor.expertise-team.com/post/simple-xml-processing-in-abap-part-iii-create-simple-transformation-to-parse-xml-content.aspx)**

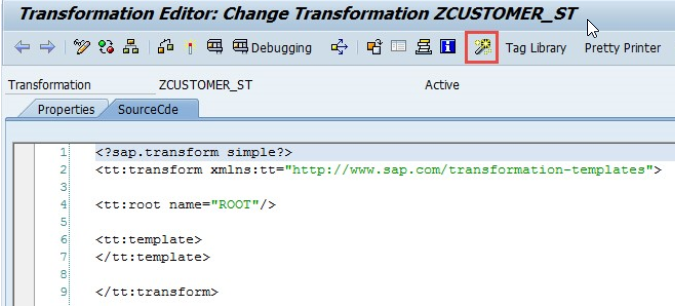
[Содержание](#Содержание)

Open the  ***strans*** transaction to ***create the Simple Transformation*** *ZCUSTOMER\_ST* that will parse the content of the XML document into the internal format of the SAP.





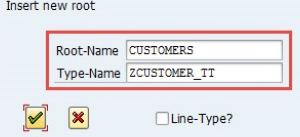
We can implement the transformation logic in two different ways - manually editing the source code, or using the visual editor. Now, I want to show you the easier one, the visual editor that we can reach by pushing the *magic wand* /wɔnd палочка/ button.



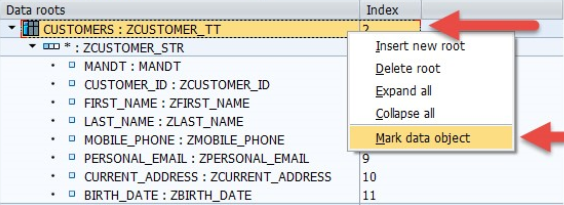
First of all, we are going to delete the default root node by right clicking on the node *ROOT*, then choosing the option ***Delete root***, because *we want to use a different root*.

Now, let’s right click on the empty, highlighted area, and choose the option ***Insert new root***.

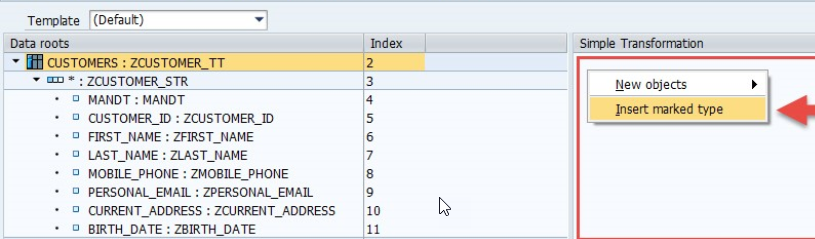
On the pop-up screen, we are going to set the *Root-Name* to the same root as we have in the XML document - *CUSTOMERS*, and then use the previously created *ZCUSTOMER\_TT* table type in the *Type-Name* field. It means that we are going to send back the parsed data to the caller program in an internal table that has the type *ZCUSTOMER\_TT*.



As a result, the system looks up for the structure of the specified table type, and inserts its elements as a group of nodes in a hierarchical view. Here, we want to copy these meta-data to render the same hierarchy on the Simple Transformation side that we can accomplish right clicking on the root node *CUSTOMERS*, and then choosing the option ***Mark data object***.



Next, we are going to use this meta-data and generate the Simple Transformation also by right clicking on the empty, highlighted area, and choosing the option ***Insert marked type***.



As you can see, we got almost the same structure. On the left hand side we defined the result structure, and *on the right hand side we defined the simple transformation that has to correspond to the structure of the XML document*.

Here, I found a problem, namely in the XML document we have a node like *<CUSTOMER> </CUSTOMER>*, but in the Simple Transformation we use *ZCUSTOMER\_STR*. In order to have a properly functioning transformation, we have to ***change the name of this node*** by double clicking on it, and changing its name above to *CUSTOMER*. As a result, we got the following code in the source editor

<?sap.transform simple?>

<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:*root* name=*"CUSTOMERS"* type="ddic:ZCUSTOMER\_TT"></tt:root>

  <tt:template>

*<CUSTOMERS>*

      <tt:loop ref=".CUSTOMERS">

        <CUSTOMER>

          <*MANDT* tt:value-ref="MANDT"></MANDT>

          <CUSTOMER\_ID tt:value-ref="CUSTOMER\_ID"></CUSTOMER\_ID>

          <FIRST\_NAME tt:value-ref="FIRST\_NAME"></FIRST\_NAME>

          <LAST\_NAME tt:value-ref="LAST\_NAME"></LAST\_NAME>

          <MOBILE\_PHONE tt:value-ref="MOBILE\_PHONE"></MOBILE\_PHONE>

          <PERSONAL\_EMAIL tt:value-ref="PERSONAL\_EMAIL"></PERSONAL\_EMAIL>

          <CURRENT\_ADDRESS tt:value-ref="CURRENT\_ADDRESS"></CURRENT\_ADDRESS>

          <BIRTH\_DATE tt:value-ref="BIRTH\_DATE"></BIRTH\_DATE>

        </CUSTOMER>

      </tt:loop>

    </CUSTOMERS>

  </tt:template>

</tt:transform>

As you could see in the source code, we are going to require a ***MANDT*** tag, but there is no *MANDT* tag in the XML document, since it only exist in SAP systems. To solve this problem, we have to *make the MANDT tag optional* in the Simple Transformation source code by using the conditional tag.

<?sap.transform simple?>

<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:root name="CUSTOMERS" type="ddic:ZCUSTOMER\_TT"></tt:root>

  <tt:template>

    <CUSTOMERS>

      <tt:loop ref=".CUSTOMERS">

*<CUSTOMER>*

          <tt:cond>

            <*MANDT* tt:value-ref="MANDT"></MANDT>

          </tt:cond>

          <CUSTOMER\_ID tt:value-ref="CUSTOMER\_ID"></CUSTOMER\_ID>

          <FIRST\_NAME tt:value-ref="FIRST\_NAME"></FIRST\_NAME>

          <LAST\_NAME tt:value-ref="LAST\_NAME"></LAST\_NAME>

          <MOBILE\_PHONE tt:value-ref="MOBILE\_PHONE"></MOBILE\_PHONE>

          <PERSONAL\_EMAIL tt:value-ref="PERSONAL\_EMAIL"></PERSONAL\_EMAIL>

          <CURRENT\_ADDRESS tt:value-ref="CURRENT\_ADDRESS"></CURRENT\_ADDRESS>

          <BIRTH\_DATE tt:value-ref="BIRTH\_DATE"></BIRTH\_DATE>

        </CUSTOMER>

      </tt:loop>

    </CUSTOMERS>

  </tt:template>

</tt:transform>

## **[Simple XML Processing in ABAP Part IV. – Create Framework Program to Process XML Files](http://abapmentor.expertise-team.com/post/simple-xml-processing-in-abap-part-iv-create-framework-program-to-process-xml-files.aspx)**

[Содержание](#Содержание)

Report *zimport\_customers*.

First I am going to introduce the main responsibilities of the framework program –

* 1. first let's *upload the content of the XML file*,

then

* 1. *parse* this *content into an SAP internal format*,

and at last

* 1. *insert the parsed data into the database*.

REPORT ***ZIMPORT\_CUSTOMERS***.

DATA customers TYPE zcustomer\_tt.

DATA customers\_xml TYPE string.

START-OF-SELECTION.

PERFORM xml\_upload CHANGING customers\_xml.

PERFORM xml\_parse USING customers\_xml

CHANGING customers.

PERFORM insert\_into\_db USING customers.

FORM ***xml\_upload*** CHANGING customers\_xml.

DATA xml\_content\_table TYPE TABLE OF string.

DATA xml\_content\_line TYPE string.

CALL FUNCTION 'GUI\_UPLOAD'

EXPORTING

filename = *'/TEST/a.ekimenko/customer.xml'*

TABLES

data\_tab = xml\_content\_table.

LOOP AT xml\_content\_table INTO xml\_content\_line.

CONCATENATE customers\_xml xml\_content\_line INTO customers\_xml.

ENDLOOP.

ENDFORM.

FORM ***xml\_parse*** USING customers\_xml TYPE string

CHANGING customers TYPE zcustomer\_tt.

CALL TRANSFORMATION zcustomer\_st

SOURCE XML customers\_xml

RESULT customers = customers.

ENDFORM.

FORM ***insert\_into\_db*** USING customers TYPE zcustomer\_tt.

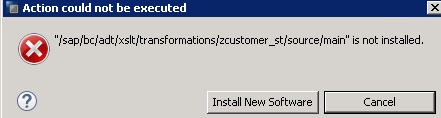
*INSERT zcustomer FROM TABLE customers*.

ENDFORM.

## 

## [**Simple XML Processing in ABAP Part V. – Processing XML Attributes**](http://abapmentor.expertise-team.com/post/simple-xml-processing-in-abap-part-v-processing-xml-attributes.aspx)

[Содержание](#Содержание)



<https://developers.sap.com/tutorials/abap-install-adt.html>

[\\it.beloil.by\public\sis\Кулак Анастасия\Настя\Аттестация\29.06.2020](file:///\\it.beloil.by\public\sis\Кулак%20Анастасия\Настя\Аттестация\29.06.2020)

As you can see, the CUSTOMER\_ID is no longer an element, instead now it's an attribute that is located next to the element, CUSTOMER.

To change the line, CUSTOMER\_ID into an attribute mapping, first we have to move it to the top of the CUSTOMER hierarchy, by replacing it with the line, MANDT, since attributes must not occur after elements, the attributes have to be listed for first.

…

<CUSTOMERS>

<tt:loop ref=".CUSTOMERS">

<CUSTOMER>

<*CUSTOMER\_ID* tt:value-ref="CUSTOMER\_ID"/>

<tt:cond>

<MANDT tt:value-ref="MANDT"></MANDT>

</ tt:cond>

…

Now, we can change the line, CUSTOMER\_ID into an attribute mapping, by defining it with the keywords attribute and name.

<CUSTOMER>  
 <*tt:attribute* name = "CUSTOMER\_ID" value-ref="CUSTOMER\_ID"/></tt:attribute>

<tt:cond>

<MANDT tt:value-ref="MANDT"></MANDT>

</tt:cond>

## [**Display ALV List easily in ABAP using CL\_SALV\_TABLE Part I. - Simple List**](https://abapmentor.expertise-team.com/post/display-alv-list-easily-in-abap-using-cl-salv-table-part-i-simple-list.aspx)

<https://abapmentor.expertise-team.com/post/display-alv-list-easily-in-abap-using-cl-salv-table-part-i-simple-list.aspx>

in SAP Netweaver 2004, SAP introduced a new Object Oriented ALV list family class, called **CL\_SALV**. It consists of different ALVs such as **table**, **hierarchy**, and **tree**.

In this blog post series, we will focus on the **CL\_SALV\_TABLE** class. I will demonstrate you different settings one by one that you can use later to customize your ALV lists.

## [**Display ALV List easily in ABAP using CL\_SALV\_TABLE Part IV. - Individual Column Settings**](http://abapmentor.expertise-team.com/post/display-alv-list-easily-in-abap-using-cl-salv-table-part-iv-individual-column-settings.aspx)

<http://abapmentor.expertise-team.com/post/display-alv-list-easily-in-abap-using-cl-salv-table-part-iv-individual-column-settings.aspx>