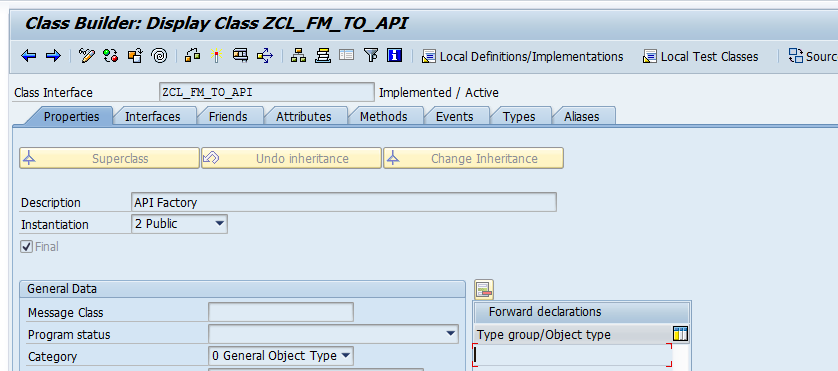
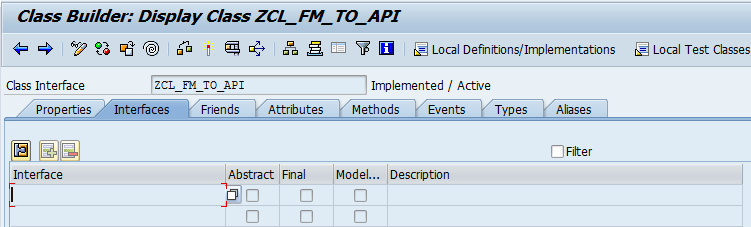
**ZCL\_FM\_TO\_API**

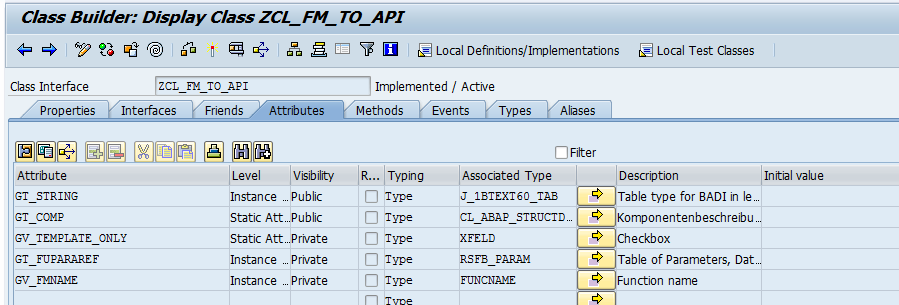
Step-1



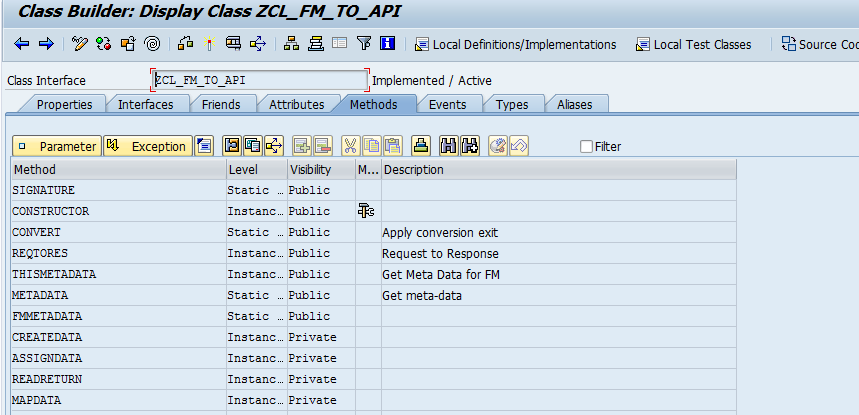
Step-2



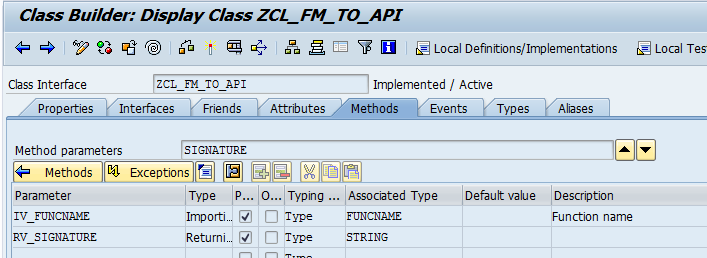
Step-3



Step-4

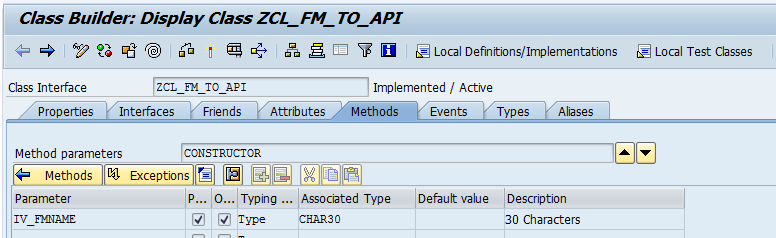


**SIGNATURE method:**



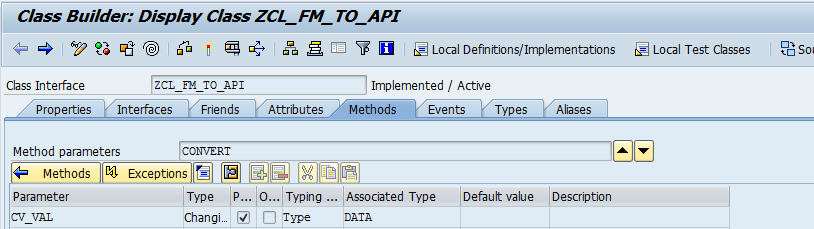
METHOD signature.  
  
  TYPES:  
  BEGIN OF ty\_params,  
  parameter TYPE fupararef-parameter,  
\*  kind      TYPE char20,  
  required  TYPE char5,  
  END OF   ty\_params,  
  
  BEGIN OF ty\_signature,  
  name        TYPE funcname,  
  description TYPE tftit-stext,  
  importing   TYPE STANDARD TABLE OF ty\_params WITH NON-UNIQUE DEFAULT KEY,  
  exporting   TYPE STANDARD TABLE OF ty\_params WITH NON-UNIQUE DEFAULT KEY,  
  changing    TYPE STANDARD TABLE OF ty\_params WITH NON-UNIQUE DEFAULT KEY,  
  tables      TYPE STANDARD TABLE OF ty\_params WITH NON-UNIQUE DEFAULT KEY,  
  END OF   ty\_signature.  
  
  DATA: lw\_signature TYPE ty\_signature,  
        lt\_fupararef TYPE STANDARD TABLE OF fupararef,  
        lw\_fupararef TYPE fupararef,  
        lw\_param     TYPE ty\_params.  
  
  IF iv\_funcname IS NOT INITIAL.  
    SELECT SINGLE  
      funcname stext  
      FROM tftit INTO  
      (lw\_signature-name, lw\_signature-description)  
      WHERE funcname = iv\_funcname  
      AND   spras = sy-langu.  
  
    IF sy-subrc IS NOT INITIAL.  
      rv\_signature = 'FM not found'.  
    ELSE.  
  
      SELECT \*  
        FROM fupararef  
        INTO TABLE lt\_fupararef  
        WHERE funcname = iv\_funcname.  
  
      IF sy-subrc IS INITIAL.  
  
  
        LOOP AT lt\_fupararef INTO lw\_fupararef.  
          lw\_param-parameter = lw\_fupararef-parameter.  
          IF lw\_fupararef-optional IS NOT INITIAL OR lw\_fupararef-paramtype = 'E'.  
            lw\_param-required = 'false'.  
          ELSE.  
            lw\_param-required = 'true'.  
          ENDIF.  
  
\*          CASE lw\_fupararef-type.  
\*            WHEN 'T'.  
\*              lw\_param-kind = 'Table'.  
\*            WHEN ''.  
\*            WHEN OTHERS.  
\*          ENDCASE.  
  
          CASE lw\_fupararef-paramtype.  
            WHEN 'I'.  
              APPEND lw\_param TO lw\_signature-importing.  
            WHEN 'E'.  
              APPEND lw\_param TO lw\_signature-exporting.  
            WHEN 'C'.  
              APPEND lw\_param TO lw\_signature-changing.  
            WHEN 'T'.  
              APPEND lw\_param TO lw\_signature-tables.  
            WHEN OTHERS.  
          ENDCASE.  
          CLEAR lw\_param.  
        ENDLOOP.  
  
        CALL METHOD zui\_cl\_json=>serialize  
          EXPORTING  
            data   = lw\_signature  
          RECEIVING  
            r\_json = rv\_signature.  
  
      ENDIF.  
  
    ENDIF.  
  ENDIF.  
  
  
ENDMETHOD.

**Constructor method:**



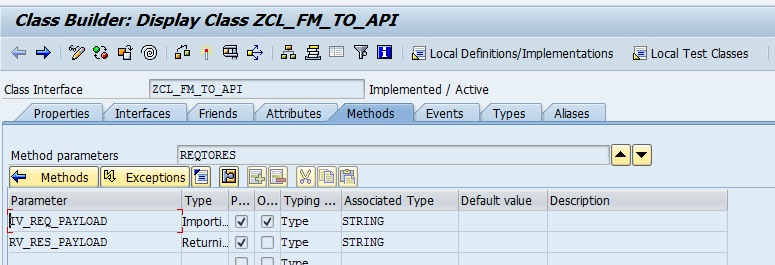
METHOD CONSTRUCTOR.  
  SELECT \*  
    FROM fupararef  
    INTO TABLE gt\_fupararef  
    WHERE funcname = iv\_fmname.  
  IF sy-subrc IS INITIAL.  
    gv\_fmname = iv\_fmname.  
    SORT gt\_fupararef BY paramtype pposition.  
  ENDIF.  
ENDMETHOD.

**Convert Method Implementation:**



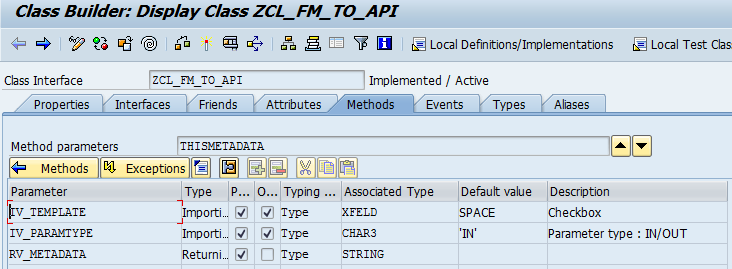
METHOD convert.  
  
  DATA:  ls\_delement   TYPE sydes\_desc,  
         lw\_names      TYPE sydes\_nameinfo,  
         lv\_fmname     TYPE rs38l\_fnam,  
         lv\_len        TYPE i.  
  
  DATA: lo\_anytype  TYPE REF TO cl\_abap\_typedescr,  
        lo\_struct   TYPE REF TO cl\_abap\_structdescr,  
        lo\_new\_tab  TYPE REF TO cl\_abap\_tabledescr,  
        lw\_comp     TYPE LINE OF cl\_abap\_structdescr=>component\_table,  
        lt\_tot\_comp TYPE cl\_abap\_structdescr=>component\_table,  
        lo\_new\_type TYPE REF TO cl\_abap\_structdescr,  
        lo\_data     TYPE REF TO data.  
  
  FIELD-SYMBOLS: <lfs\_val> TYPE any,  
                 <lfs\_tab> TYPE ANY TABLE.  
  TRY.  
      IF cv\_val IS NOT INITIAL.  
        lo\_anytype ?= cl\_abap\_typedescr=>describe\_by\_data( cv\_val ).  
  
        CASE lo\_anytype->kind.  
          WHEN lo\_anytype->kind\_table.  
            ASSIGN cv\_val TO <lfs\_tab>.  
            LOOP AT <lfs\_tab> ASSIGNING <lfs\_val>.  
              convert( CHANGING cv\_val = <lfs\_val> ).  
            ENDLOOP.  
          WHEN lo\_anytype->kind\_struct.  
            DO.  
              ASSIGN COMPONENT sy-index OF STRUCTURE cv\_val TO <lfs\_val>.  
              IF <lfs\_val> IS ASSIGNED.  
                convert( CHANGING cv\_val = <lfs\_val> ).  
                UNASSIGN <lfs\_val>.  
              ELSE.  
                EXIT.  
              ENDIF.  
            ENDDO.  
          WHEN lo\_anytype->kind\_elem.  
  
  
            TRY .  
                FREE: ls\_delement, lw\_names.  
\*get the field properties of the column  
                DESCRIBE FIELD cv\_val INTO ls\_delement.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* Apply conversion exit dynamically  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*3rd row of 'SYDES\_NAMEINFO' stores the conversion routine name,if any.  
                READ TABLE ls\_delement-names INTO lw\_names INDEX 3.  
\*if found, the identification of 'conversion routine' is '==' as the \*first 2 characters  
                IF sy-subrc IS INITIAL AND lw\_names-name+0(2) = '=='.  
\*The FM corresponding conversion routine is always of the format  
\*'CONVERSION\_EXIT\_<NAME>\_INPUT'. Prepare the FM name  
                  lv\_fmname = |CONVERSION\_EXIT\_{ lw\_names-name+2(5) }\_INPUT|.  
                ELSE.  
                  READ TABLE ls\_delement-names INTO lw\_names INDEX lines( ls\_delement-names ).  
                  IF sy-subrc IS INITIAL AND lw\_names-name+0(2) = '=='.  
                    lv\_fmname = |CONVERSION\_EXIT\_{ lw\_names-name+2(5) }\_INPUT|.  
                  ENDIF.  
                ENDIF.  
                IF lv\_fmname IS NOT INITIAL.  
\*call the conversion exit  
                  SELECT SINGLE funcname  
                    FROM tfdir  
                    INTO lv\_fmname  
                    WHERE funcname = lv\_fmname.  
                  IF sy-subrc IS INITIAL.  
  
                    CALL FUNCTION lv\_fmname  
                      EXPORTING  
                        input  = cv\_val  
                      IMPORTING  
                        output = cv\_val.  
                  ENDIF.  
                  CLEAR lv\_fmname.  
                ENDIF.  
              CATCH cx\_root.  
            ENDTRY.  
  
          WHEN OTHERS.  
        ENDCASE.  
      ENDIF.  
    CATCH cx\_root.  
  ENDTRY.  
ENDMETHOD.

**REQTORES 🡪 Method**



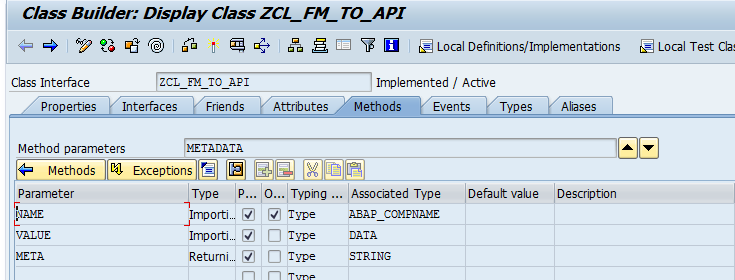
METHOD reqtores.  
  
  FIELD-SYMBOLS:  
  <lfs\_input>     TYPE any,  
  <lfs\_output>    TYPE any.  
  
  DATA: lo\_root       TYPE REF TO cx\_root,  
        lo\_data\_in    TYPE REF TO data,  
        lo\_data\_out   TYPE REF TO data,  
        lt\_fparam     TYPE abap\_func\_parmbind\_tab,  
        lt\_fparam2    TYPE abap\_func\_parmbind\_tab.  
  
  TRY.  
      IF gv\_fmname IS NOT INITIAL.  
  
        lo\_data\_in = createdata( iv\_calltype = 'IN' ).  
        ASSIGN lo\_data\_in->\* TO <lfs\_input>.  
  
  
        IF <lfs\_input> IS ASSIGNED.  
  
          lo\_data\_out = createdata( iv\_calltype = 'OUT' ).  
          ASSIGN lo\_data\_out->\* TO <lfs\_output>.  
          IF <lfs\_output> IS ASSIGNED.  
  
            TRY .  
                zui\_cl\_json=>deserialize(  
                  EXPORTING  
                    json             = iv\_req\_payload  
                  CHANGING  
                    data             =  <lfs\_input>  
                ).  
  
                convert( CHANGING cv\_val = <lfs\_input> ).  
  
\*                lt\_fparam2 = me->mapdata( iv\_input = <lfs\_input> iv\_output = <lfs\_output> ).  
\*                CALL FUNCTION gv\_fmname PARAMETER-TABLE lt\_fparam2.  
  
                lt\_fparam = me->assigndata( iv\_input = <lfs\_input> iv\_output = <lfs\_output> ).  
  
                CALL FUNCTION gv\_fmname PARAMETER-TABLE lt\_fparam.  
                me->readreturn( EXPORTING it\_fparam = lt\_fparam CHANGING rv\_output = <lfs\_output> ).  
  
                IF <lfs\_output> IS ASSIGNED.  
                  CALL METHOD zui\_cl\_json=>serialize  
                    EXPORTING  
                      data   = <lfs\_output>  
                    RECEIVING  
                      r\_json = rv\_res\_payload.  
                ENDIF.  
              CATCH cx\_root INTO lo\_root.  
                rv\_res\_payload = lo\_root->get\_text( ).  
            ENDTRY.  
          ENDIF.  
        ENDIF.  
      ELSE.  
        rv\_res\_payload = 'Mapping not found'.  
      ENDIF.  
    CATCH cx\_root INTO lo\_root.  
      rv\_res\_payload = lo\_root->get\_text( ).  
  ENDTRY.  
  IF rv\_res\_payload IS INITIAL.  
    rv\_res\_payload = 'No result returned'.  
  ENDIF.  
ENDMETHOD.

**THISMETADATA-Method**



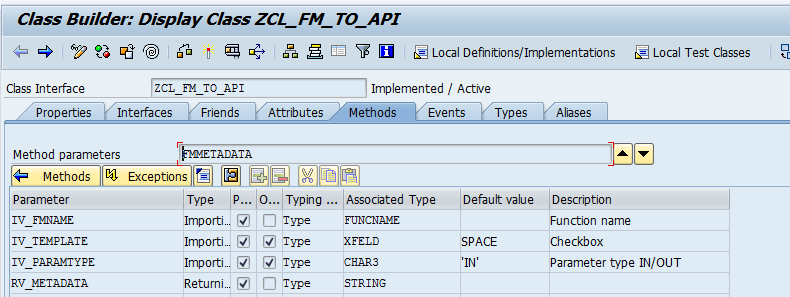
METHOD thismetadata.  
  
  DATA: lo\_root       TYPE REF TO cx\_root,  
        lo\_data    TYPE REF TO data.  
  
  FIELD-SYMBOLS: <lfs\_data> TYPE any.  
  
  IF gv\_fmname IS NOT INITIAL.  
  
    lo\_data = createdata( iv\_calltype = iv\_paramtype ).  
    ASSIGN lo\_data->\* TO <lfs\_data>.  
  
    IF <lfs\_data> IS ASSIGNED.  
      IF iv\_template IS NOT INITIAL.  
        gv\_template\_only = abap\_true.  
      ELSE.  
        CLEAR gv\_template\_only.  
      ENDIF.  
      IF iv\_template IS INITIAL.  
        rv\_metadata = '{' && | "SIGNATURE": { signature( gv\_fmname ) } , "METADATA": { metadata( <lfs\_data> ) } | && '}'.  
\*        rv\_metadata = metadata( <lfs\_data> ).  
      ELSE.  
        rv\_metadata = metadata( <lfs\_data> ).  
      ENDIF.  
    ENDIF.  
  ENDIF.  
  
  
ENDMETHOD.

**METADATA~method**



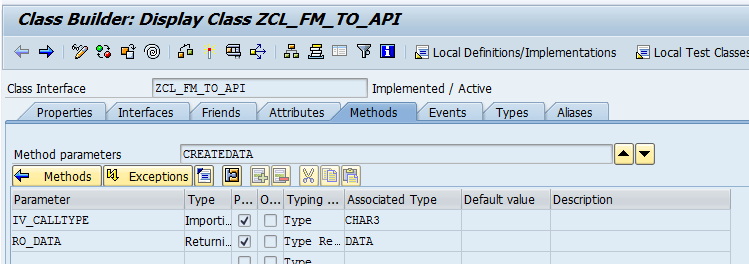
METHOD metadata.  
  
  DATA:  ls\_delement   TYPE sydes\_desc,  
         lw\_names      TYPE sydes\_nameinfo,  
         lv\_fmname     TYPE rs38l\_fnam,  
         lv\_len        TYPE i.  
  
  DATA: lo\_anytype  TYPE REF TO cl\_abap\_typedescr,  
        lo\_struct   TYPE REF TO cl\_abap\_structdescr,  
        lo\_element   TYPE REF TO cl\_abap\_elemdescr,  
        lo\_new\_tab  TYPE REF TO cl\_abap\_tabledescr,  
        lw\_comp     TYPE LINE OF cl\_abap\_structdescr=>component\_table,  
        lt\_tot\_comp TYPE cl\_abap\_structdescr=>component\_table,  
        lo\_new\_type TYPE REF TO cl\_abap\_structdescr,  
        lo\_data     TYPE REF TO data,  
        lw\_components TYPE abap\_compdescr,  
        lv\_prefix     TYPE string,  
        lw\_dfies      TYPE dfies,  
        lv\_meta       TYPE string.  
  
  
  FIELD-SYMBOLS: <lfs\_val> TYPE any,  
                 <lfs\_tab> TYPE STANDARD TABLE,  
                 <lfs\_wa>  TYPE any.  
  TRY.  
      IF name IS NOT INITIAL.  
        lv\_prefix = |"{ name }":|.  
      ENDIF.  
      lo\_anytype ?= cl\_abap\_typedescr=>describe\_by\_data( value ).  
  
      CASE lo\_anytype->kind.  
  
        WHEN lo\_anytype->kind\_table.  
  
          ASSIGN value TO <lfs\_tab>.  
          IF <lfs\_tab> IS ASSIGNED.  
            APPEND INITIAL LINE TO <lfs\_tab> ASSIGNING <lfs\_wa>.  
            IF <lfs\_wa> IS ASSIGNED.  
              meta = |{ lv\_prefix }[ { metadata( name = space value = <lfs\_wa> ) }]|.  
              UNASSIGN <lfs\_wa>.  
            ENDIF.  
            DELETE <lfs\_tab> INDEX lines( <lfs\_tab> ).  
            UNASSIGN <lfs\_tab>.  
          ENDIF.  
\*  
        WHEN lo\_anytype->kind\_struct.  
          lo\_struct ?= lo\_anytype.  
          meta = lv\_prefix && '{'.  
          LOOP AT lo\_struct->components INTO lw\_components.  
            ASSIGN COMPONENT lw\_components-name OF STRUCTURE value TO <lfs\_val>.  
            IF <lfs\_val> IS ASSIGNED.  
  
              meta = |{ meta } { metadata( name = lw\_components-name  value = <lfs\_val> ) },|.  
              UNASSIGN <lfs\_val>.  
            ENDIF.  
          ENDLOOP.  
          SHIFT meta RIGHT DELETING TRAILING ','.  
          meta = meta && '}'.  
        WHEN lo\_anytype->kind\_elem.  
          lo\_element ?= lo\_anytype.  
          IF lo\_anytype->is\_ddic\_type( ) IS NOT INITIAL.  
            lw\_dfies = lo\_element->get\_ddic\_field( ).  
  
            IF gv\_template\_only IS INITIAL.  
  
              lv\_len = lw\_dfies-leng.  
              REPLACE ALL OCCURRENCES OF '"' IN lw\_dfies-fieldtext WITH space.  
              CONDENSE lw\_dfies-fieldtext.  
              lv\_meta = '{' && | "DATATYPE":"{ lw\_dfies-datatype }" ,  "LENGTH":{ lv\_len }, "DESCRIPTION":"{ lw\_dfies-fieldtext }" | && '}'.  
  
            ELSE.  
  
              CASE lw\_dfies-datatype.  
                WHEN 'I' OR 'N'.  
                  lv\_meta = '0'.  
                WHEN 'P'.  
                  lv\_meta = '0.0'.  
                WHEN 'D' or 'DATS'.  
                  lv\_meta = |"{ sy-datum+0(4) }-{ sy-datum+4(2) }-{ sy-datum+6(2) }"|.  
                WHEN 'T' or 'TIMS'.  
                  lv\_meta = |"{ sy-uzeit+0(2) }:{ sy-uzeit+2(2) }:{ sy-uzeit+4(2) }"|.  
                WHEN OTHERS.  
                  lv\_meta = '""'.  
              ENDCASE.  
  
            ENDIF.  
          ELSE.  
            IF gv\_template\_only IS INITIAL.  
              lv\_len = lo\_element->output\_length.  
              lv\_meta = '{' && | "DATATYPE": "GENERIC",  "LENGTH": { lv\_len } | && '}'.  
            ELSE.  
              lv\_meta = '""'.  
            ENDIF.  
          ENDIF.  
\*          lv\_abap\_type = cl\_abap\_typedescr=>g.  
          meta = |{ lv\_prefix } { lv\_meta } |.  
  
        WHEN OTHERS.  
      ENDCASE.  
    CATCH cx\_root.  
  ENDTRY.  
ENDMETHOD.

**FMMETADATA~method**



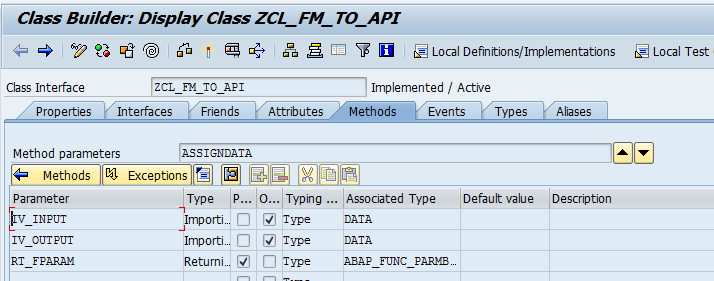
METHOD fmmetadata.  
  
  DATA: lo\_fm2api TYPE REF TO zcl\_fm\_to\_api.  
  
  CREATE OBJECT lo\_fm2api  
    EXPORTING  
      iv\_fmname = iv\_fmname.  
  IF lo\_fm2api IS BOUND.  
    rv\_metadata = lo\_fm2api->thismetadata( iv\_template = iv\_template iv\_paramtype = iv\_paramtype ).  
  ENDIF.  
  
ENDMETHOD.

**CREATEDATA~Method**



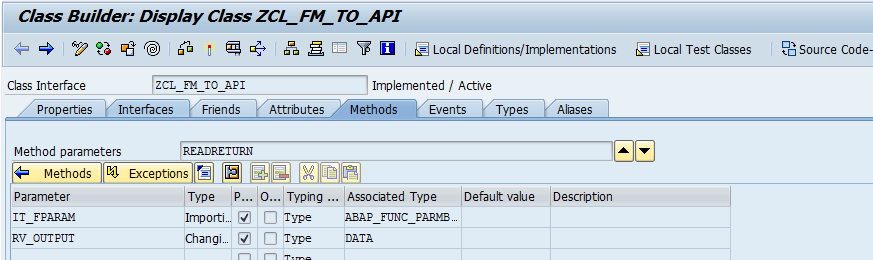
METHOD createdata.  
  
  DATA: lo\_anytype  TYPE REF TO cl\_abap\_typedescr,  
        lo\_struct   TYPE REF TO cl\_abap\_structdescr,  
        lo\_new\_tab  TYPE REF TO cl\_abap\_tabledescr,  
        lw\_comp     TYPE LINE OF cl\_abap\_structdescr=>component\_table,  
        lt\_tot\_comp TYPE cl\_abap\_structdescr=>component\_table,  
        lo\_new\_type TYPE REF TO cl\_abap\_structdescr,  
        lo\_data     TYPE REF TO data.  
  
  DATA: lw\_fupararef TYPE fupararef.  
  
  LOOP AT gt\_fupararef INTO lw\_fupararef.  
    IF iv\_calltype = 'IN' AND lw\_fupararef-paramtype <> 'E'  
    OR iv\_calltype = 'OUT' AND lw\_fupararef-paramtype <> 'I'.  
      FREE lo\_anytype.  
      TRY .  
          IF lw\_fupararef-structure IS INITIAL.  
            lw\_fupararef-structure = 'STRING'.  
          ENDIF.  
          lo\_anytype ?= cl\_abap\_typedescr=>describe\_by\_name( lw\_fupararef-structure ).  
          IF lw\_fupararef-paramtype = 'T' AND lo\_anytype->kind <> lo\_anytype->kind\_table.  
            lo\_struct ?= lo\_anytype.  
            lo\_new\_tab = cl\_abap\_tabledescr=>create(  
                            p\_line\_type  = lo\_struct  
                            p\_table\_kind = cl\_abap\_tabledescr=>tablekind\_std  
                            p\_unique     = abap\_false ).  
            lw\_comp-type = lo\_new\_tab.  
          ELSE.  
            lw\_comp-type ?= lo\_anytype.  
          ENDIF.  
          lw\_comp-name = lw\_fupararef-parameter.  
          APPEND lw\_comp TO lt\_tot\_comp.  
          CLEAR: lw\_comp.  
        CATCH cx\_root.  
      ENDTRY.  
    ENDIF.  
  ENDLOOP.  
  TRY.  
      lo\_new\_type = cl\_abap\_structdescr=>create( lt\_tot\_comp ).  
      CREATE DATA lo\_data TYPE HANDLE lo\_new\_type.  
      ro\_data ?= lo\_data.  
    CATCH cx\_root.  
  ENDTRY.  
  
  
ENDMETHOD.

**ASSIGNDATA~Method**



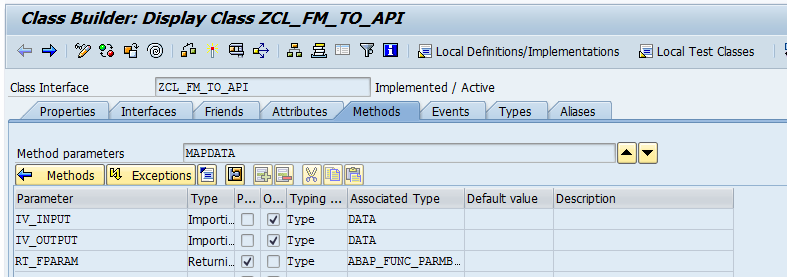
METHOD assigndata.  
  
  DATA: lo\_data       TYPE REF TO data,  
        lo\_data\_wa    TYPE REF TO data,  
        lw\_fparam     TYPE abap\_func\_parmbind,  
        lo\_data\_error TYPE REF TO cx\_sy\_create\_data\_error,  
        lv\_message    TYPE string,  
        lo\_anytype  TYPE REF TO cl\_abap\_typedescr.  
  
  DATA: lw\_fupararef TYPE fupararef.  
  
  FIELD-SYMBOLS: <lfs\_val>  TYPE any,  
                 <lfs\_val2> TYPE any.  
  
  LOOP AT gt\_fupararef INTO lw\_fupararef.  
  
    IF lw\_fupararef-paramtype <> 'E'.  
      ASSIGN COMPONENT lw\_fupararef-parameter OF STRUCTURE iv\_input TO <lfs\_val>.  
    ELSE.  
      ASSIGN COMPONENT lw\_fupararef-parameter OF STRUCTURE iv\_output TO <lfs\_val>.  
    ENDIF.  
    IF <lfs\_val> IS ASSIGNED.  
      TRY.  
          FREE: lo\_data,lo\_data\_wa.  
  
          IF lw\_fupararef-paramtype = 'T'.  
            lo\_anytype ?= cl\_abap\_typedescr=>describe\_by\_name( lw\_fupararef-structure ).  
            IF lo\_anytype->kind <> lo\_anytype->kind\_table.  
  
              CREATE DATA lo\_data    TYPE TABLE OF (lw\_fupararef-structure).  
              CREATE DATA lo\_data\_wa TYPE (lw\_fupararef-structure).  
            ELSE.  
              CREATE DATA lo\_data TYPE (lw\_fupararef-structure).  
            ENDIF.  
          ELSE.  
            CREATE DATA lo\_data TYPE (lw\_fupararef-structure).  
          ENDIF.  
  
          IF lo\_data IS BOUND.  
  
            ASSIGN lo\_data->\* TO <lfs\_val2>.  
  
            IF <lfs\_val2> IS ASSIGNED.  
              <lfs\_val2> = <lfs\_val>.  
              UNASSIGN <lfs\_val2>.  
            ENDIF.  
  
            lw\_fparam-value = lo\_data.  
  
            IF lw\_fupararef-paramtype = 'T'.  
              lw\_fparam-tables\_wa = lo\_data\_wa.  
            ELSE.  
            ENDIF.  
            UNASSIGN <lfs\_val>.  
            lw\_fparam-name = lw\_fupararef-parameter.  
            CASE lw\_fupararef-paramtype.  
              WHEN 'E'.  
                lw\_fparam-kind = abap\_func\_importing.  
              WHEN 'I'.  
                lw\_fparam-kind = abap\_func\_exporting.  
              WHEN 'C'.  
                lw\_fparam-kind = abap\_func\_changing.  
              WHEN 'T'.  
                lw\_fparam-kind = abap\_func\_tables.  
              WHEN OTHERS.  
            ENDCASE.  
            INSERT lw\_fparam INTO TABLE rt\_fparam.  
            FREE lw\_fparam.  
            FREE lo\_data.  
          ENDIF.  
        CATCH cx\_sy\_create\_data\_error INTO lo\_data\_error.  
          lv\_message = lo\_data\_error->get\_text( ).  
      ENDTRY.  
      UNASSIGN <lfs\_val>.  
    ENDIF.  
  
  ENDLOOP.  
ENDMETHOD.

**READRETURN~Method**



METHOD readreturn.  
  
  DATA: lw\_fparam     TYPE abap\_func\_parmbind.  
  
  
  FIELD-SYMBOLS: <lfs\_target>  TYPE any,  
                 <lfs\_source>  TYPE any.  
  
  LOOP AT it\_fparam INTO lw\_fparam.  
    IF lw\_fparam-kind <> abap\_func\_exporting.  
      ASSIGN COMPONENT lw\_fparam-name OF STRUCTURE rv\_output TO <lfs\_target>.  
      IF <lfs\_target> IS ASSIGNED.  
\*        IF lw\_fparam-kind <> abap\_func\_tables.  
          ASSIGN lw\_fparam-value->\* TO <lfs\_source>.  
\*        ELSE.  
\*          ASSIGN lw\_fparam-tables\_wa->\* TO <lfs\_source>.  
\*        ENDIF.  
        IF <lfs\_source> IS ASSIGNED.  
          <lfs\_target> = <lfs\_source>.  
          UNASSIGN <lfs\_source>.  
        ENDIF.  
        UNASSIGN <lfs\_target>.  
      ENDIF.  
    ENDIF.  
  ENDLOOP.  
ENDMETHOD.

**MAPDATA~Method**



METHOD mapdata.  
  
  DATA: lo\_data       TYPE REF TO data,  
        lo\_data\_wa    TYPE REF TO data,  
        lw\_fparam     TYPE abap\_func\_parmbind,  
        lo\_data\_error TYPE REF TO cx\_sy\_create\_data\_error,  
        lv\_message    TYPE string,  
        lo\_anytype    TYPE REF TO cl\_abap\_typedescr.  
  
  DATA: lw\_fupararef TYPE fupararef.  
  
  FIELD-SYMBOLS: <lfs\_val>  TYPE any,  
                 <lfs\_val2> TYPE any.  
  
  LOOP AT gt\_fupararef INTO lw\_fupararef.  
  
    IF lw\_fupararef-paramtype <> 'E'.  
      ASSIGN COMPONENT lw\_fupararef-parameter OF STRUCTURE iv\_input TO <lfs\_val>.  
    ELSE.  
      ASSIGN COMPONENT lw\_fupararef-parameter OF STRUCTURE iv\_output TO <lfs\_val>.  
    ENDIF.  
    IF <lfs\_val> IS ASSIGNED.  
      TRY.  
          FREE: lo\_data,lo\_data\_wa.  
  
          IF lw\_fupararef-paramtype = 'T'.  
            FREE lo\_anytype.  
            lo\_anytype ?= cl\_abap\_typedescr=>describe\_by\_name( lw\_fupararef-structure ).  
            IF lo\_anytype IS BOUND.  
              IF lo\_anytype->kind <> lo\_anytype->kind\_table.  
                CREATE DATA lo\_data\_wa TYPE (lw\_fupararef-structure).  
              ENDIF.  
            ENDIF.  
          ENDIF.  
  
          GET REFERENCE OF <lfs\_val> INTO lo\_data.  
          IF lo\_data IS BOUND.  
            lw\_fparam-value = lo\_data.  
            IF lw\_fupararef-paramtype = 'T'.  
              lw\_fparam-tables\_wa = lo\_data\_wa.  
            ELSE.  
            ENDIF.  
            UNASSIGN <lfs\_val>.  
            lw\_fparam-name = lw\_fupararef-parameter.  
            CASE lw\_fupararef-paramtype.  
              WHEN 'E'.  
                lw\_fparam-kind = abap\_func\_importing.  
              WHEN 'I'.  
                lw\_fparam-kind = abap\_func\_exporting.  
              WHEN 'C'.  
                lw\_fparam-kind = abap\_func\_changing.  
              WHEN 'T'.  
                lw\_fparam-kind = abap\_func\_tables.  
              WHEN OTHERS.  
            ENDCASE.  
            INSERT lw\_fparam INTO TABLE rt\_fparam.  
            FREE lw\_fparam.  
            FREE lo\_data.  
          ENDIF.  
        CATCH cx\_sy\_create\_data\_error INTO lo\_data\_error.  
          lv\_message = lo\_data\_error->get\_text( ).  
      ENDTRY.  
      UNASSIGN <lfs\_val>.  
    ENDIF.  
  
  ENDLOOP.  
ENDMETHOD.