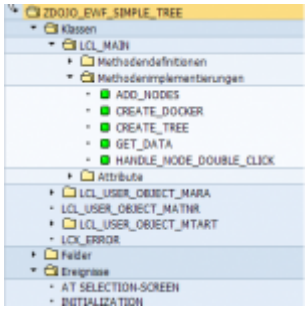


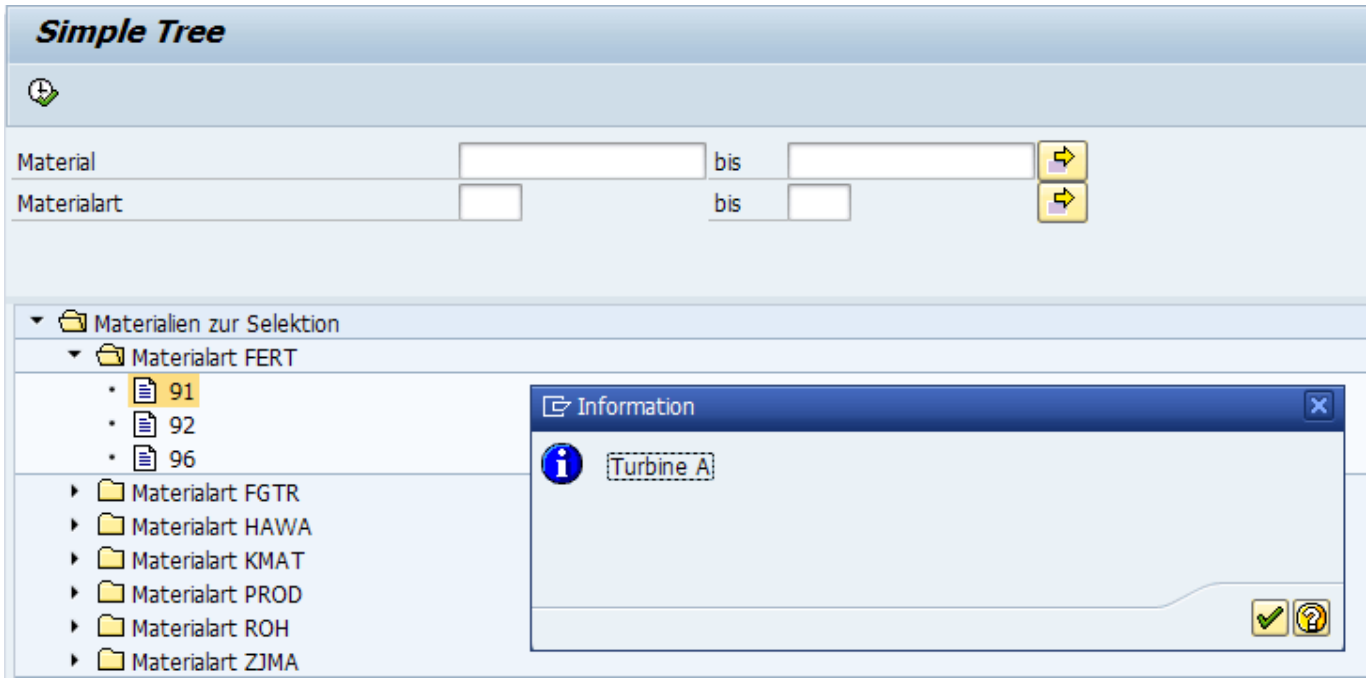
Simple Tree Model mit User-Object

Bäume sind immer interessant, finde ich. Sie sind nicht so eintönig gleichmäßig wie die meisten Listen. Der CL_SIMPLE_TREE_MODEL hat jedoch zudem auch in der Programmierung eine schöne Besonderheit: Zu jedem Knoten kann die Instanz einer beliebigen Klasse übergeben werden. Mit der Methode NODE_GET_USER_OBJECT kann man sich dann die Instanz geben lassen und hiermit weiter arbeiten.

Tree-Ausgabe

Als Beispielaufgabe habe ich mir die folgende gestellt: Zeige alle Materialien zu einer Selektion nach Materialart unterteilt an. Also: Hauptknoten – Materialart – Materialien. Die Selektion der Daten erfolgt im Selektionsbild des Reports bei AT SELECTION-SCREEN. Auf unserem IDES-System haben ich nur ein paar Dutzend Materialien, da geht die Selektion zügig.

Mit Doppelklick auf einen Eintrag soll – je nach Knotenart „Materialart“ oder „Materialnummer“ – die jeweilige Information ausgegeben werden. Es muss also bei dem Knoten „Materialart“ eine andere Klasse verwendet werden, als bei der Knotenart „Materialnummer“.



Zusätzlich zur Demonstration des CL_SIMPLE_TREE_MODEL habe ich versucht, möglichst viele neue Sprachbefehle zu verwenden. Gerade bei der Verwendung des User-Objects macht der Befehl NEW zur Instantiierung einer Klasse die Programmierung wirklich elegant.

User-Object

für das User-Object habe ich eine Hauptklasse LCL_USER_OBJECT_MARA definiert:

```
CLASS lcl_user_object_mara DEFINITION ABSTRACT.
  PUBLIC SECTION.
    DATA mara TYPE mara.
    METHODS constructor IMPORTING i_mara TYPE mara.
    METHODS get_text.
ENDCLASS.

CLASS lcl_user_object_mara IMPLEMENTATION.
  METHOD constructor.
    mara = i_mara.
  ENDMETHOD.
  METHOD get_text.
    SELECT SINGLE maktx FROM makt INTO @DATA(text)
     WHERE matnr = @mara-matnr
       AND spras = @sy-langu.
    IF sy-subrc = 0.
      MESSAGE text TYPE 'I'.
    ENDIF.
  ENDMETHOD.
ENDCLASS.
```

Die Klasse hat nur das Attribut MARA, das bei der Erzeugung mitgegeben werden muss und die Methode GET_TEXT mit der ein Text zum Objekt ermittelt wird.

Da ich die Klasse als ABSTRAKT definiert habe, kann ich diese Klasse nicht instantiieren. Das geht nur bei den von dieser Klasse abgeleiteten Klassen:

```
CLASS lcl_user_object_mtart DEFINITION INHERITING FROM lcl_user_object_mara.
  PUBLIC SECTION.
    METHODS get_text REDEFINITION.
ENDCLASS.
CLASS lcl_user_object_mtart IMPLEMENTATION.
  METHOD get_text.
    MESSAGE |Materialart { mara-mtart }|  TYPE 'I'.
  ENDMETHOD.
ENDCLASS.

CLASS lcl_user_object_matnr DEFINITION INHERITING FROM lcl_user_object_mara.
  PUBLIC SECTION.
ENDCLASS.
```

In der Klasse *MTART* redefiniere ich die Methode „GET_TEXT“ um einen eigenen Text für *Materialart* zu bekommen.

Die Klasse *MATNR* ist nur eine leere Hülle, da sie alles andere von der Hauptklasse erbt.

Natürlich hätte ich auch zwei komplett unterschiedliche und voneinander unabhängige Klassen definieren können.

Hauptprogramm

Das Hauptprogramm besteht nur den Selektionsparametern und aus zwei Ereignissen:

1. dem Ereignis INITIALIZATION, in dem ich den Docking-Container erzeuge und
2. dem Ereignis AT SELECTION-SCREEN, in dem ich die Daten selektiere und den Baum erzeuge

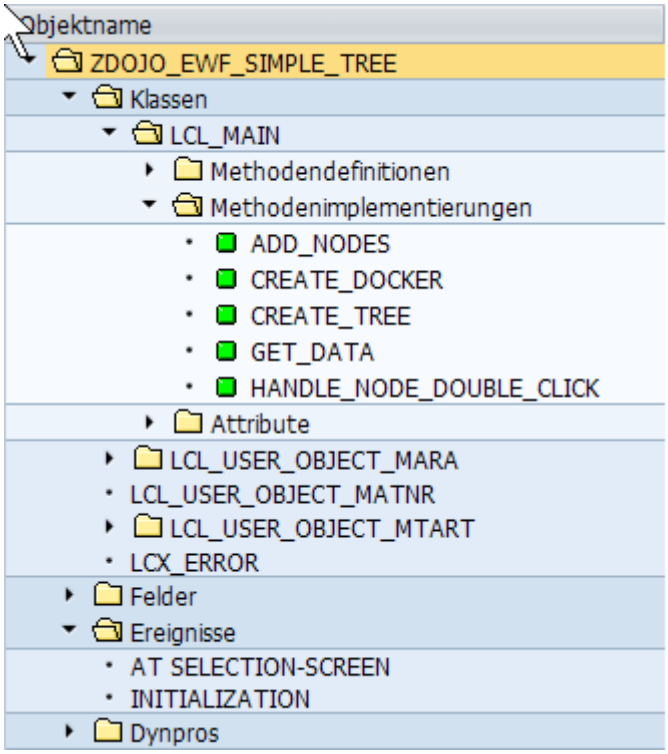
```
REPORT z_simple_tree_model.

DATA s_mara TYPE mara.

SELECT-OPTIONS s_matnr FOR s_mara-matnr.
SELECT-OPTIONS s_mtart FOR s_mara-mtart.

[...Klassendefinition...]

INITIALIZATION.
  lcl_main=>create_docker( ).

AT SELECTION-SCREEN.
  TRY.
      lcl_main=>get_data( ).
      lcl_main=>create_tree( ).
      lcl_main=>add_nodes( ).
    CATCH lcx_error.
      MESSAGE 'Fehler bei Selektion' TYPE 'I'.
  ENDTRY.
```

Hauptklasse

Die Klasse LCL_MAIN, in der die Logik des Programms vorhanden ist, besteht aus diesen vier Methoden:

1. Create_Docker
2. Create_Tree
3. Get_Data
4. Add_Nodes

Zusätzlich gibt es noch die Methode zur Ereignisbehandlung des Doppelklicks auf einen Knoten: Handle_Node_Double_Click.



Create_Docker

So simple:

```
    IF docker IS INITIAL.
      docker = NEW #( side = cl_gui_docking_container=>dock_at_bottom ratio =
50 ).
    ENDIF.
```

Get_Data

Ebenfalls nicht spektakulär:

```
    SELECT * FROM mara INTO TABLE t_mara
     WHERE matnr IN s_matnr
       AND mtart IN s_mtart.
    IF sy-subrc > 0.
      RAISE EXCEPTION TYPE lcx_error.
    ENDIF.
```

Zur Fehlerbehandlung habe ich eine eigene Exception-class erstellt:

```
CLASS lcx_error DEFINITION INHERITING FROM cx_no_check.
ENDCLASS.
```

Create_Tree

Bei der Erzeugung des CL_SIMPLE_TREE_MODEL bin ich auf die erste Hürde gestoßen, denn die Erzeugung des Control läuft etwas anders, als bei den meisten anderen GUI-Controls. Normalerweise instantiiert man das GUI-Control unter Angabe des Containers in das das Control

eingefügt werden soll (Parameter PARENT). Nicht so bei dieser Klasse. Hier wird erst das Tree-Objekt erzeugt und danach mit der Methode CREATE_TREE_CONTROL an den PARENT-Container gehängt:

```
    IF tree IS BOUND.
      tree->delete_all_nodes( ).
    ELSE.
      tree   = NEW #( node_selection_mode =
cl_simple_tree_model=>node_sel_mode_single ).
      tree->create_tree_control( EXPORTING parent = docker ).

      tree->set_registered_events(
              EXPORTING events = VALUE #(
                 ( eventid = cl_simple_tree_model=>eventid_node_double_click
appl_event = space ) ) ).

      SET HANDLER handle_node_double_click FOR tree.
    ENDIF.

    tree->add_node(
             node_key = 'Root'                                  "#EC NOTEXT
             isfolder = 'X'
             text     = 'Materialien zur Selektion' ).
```

Das Event Doppelklick wird hier ebenfalls registriert und der event handler dafür installiert. Zusätzlich wird der Hauptknoten ROOT eingefügt.

Add_Nodes

Mit der Methode ADD_NODE des Tree-Controls werden einzelne Knoten in den Baum eingehängt. Immer unter Angabe des Knoten-ID, des übergeordneten Knotens, Text und ein paar anderen. An dieser Stelle kann das User-Object übergeben werden, dass dann zur Knoten-ID zur Verfügung steht:

```
    DATA l_mtart TYPE mtart.

    SORT t_mara BY mtart matnr.

    LOOP AT t_mara INTO DATA(mara).

      IF l_mtart <> mara-mtart.
        l_mtart = mara-mtart.
        "Knoten MATERIALART
        tree->add_node(
            node_key = CONV #( mara-mtart )
            relative_node_key = 'Root'
            relationship = cl_simple_tree_model=>relat_last_child
            isfolder = 'X'
```

```
        text     = |Materialart { mara-mtart } |
        user_object = NEW lcl_user_object_mtart( i_mara = mara ) ).
  ENDIF.

  "Knoten MATERIALNUMMER
  tree->add_node(
      node_key          = |{ mara-matnr ALPHA = OUT }|
      relative_node_key = CONV #( mara-mtart )
      relationship      = cl_simple_tree_model=>relat_last_child
      isfolder          = space
      text              = |{ mara-matnr ALPHA = OUT }|
      user_object       = NEW lcl_user_object_matnr( i_mara = mara ) ).

ENDLOOP.

tree->expand_root_nodes( ).
```

Ereignisbehandler

Im Ereignisbehandler prüfen wir, welchen Objekttyp das User-Object hat, um gegebenenfalls anders reagieren zu können:

```
DATA o_user_object_matnr TYPE REF TO lcl_user_object_matnr.
DATA o_user_object_mtart TYPE REF TO lcl_user_object_mtart.
DATA o_object TYPE REF TO object.

tree->node_get_user_object( EXPORTING node_key    = node_key
                            IMPORTING user_object = o_object ).

IF o_object IS INSTANCE OF lcl_user_object_mtart.
  o_user_object_mtart ?= o_object.
  o_user_object_mtart->get_text( ).
  EXIT.
ENDIF.

IF o_object IS INSTANCE OF lcl_user_object_matnr.
  o_user_object_matnr ?= o_object.
  o_user_object_matnr->get_text( ).
  EXIT.
ENDIF.
```

In diesem Fall verwenden wir zwar für MTART und MATNR die gleiche Methode GET_TEXT, aber hier könnte man je Objekt eine andere Funktion ausführen. Falls im ABAP Release die Syntax IS INSTANCE OF noch nicht verfügbar ist, muss mit TRY – CATCH geprüft werden, ob der Cast zwischen OBJECT und User-Object erfolgreich war oder nicht:

```
TRY.
    o_user_object_matnr ?= o_object.
    o_user_object_matnr->get_text( ).
  CATCH cx_sy_move_cast_error.
ENDTRY.
```

Das komplette Programm

```
REPORT zdemo_simple_tree_model.


DATA s_mara TYPE mara.

SELECT-OPTIONS s_matnr FOR s_mara-matnr.
SELECT-OPTIONS s_mtart FOR s_mara-mtart.

CLASS lcx_error DEFINITION INHERITING FROM cx_no_check.
ENDCLASS.

CLASS lcl_user_object_mara DEFINITION ABSTRACT.
  PUBLIC SECTION.
    DATA mara TYPE mara.
    METHODS constructor IMPORTING i_mara TYPE mara.
    METHODS get_text.
ENDCLASS.

CLASS lcl_user_object_mara IMPLEMENTATION.
  METHOD constructor.
    mara = i_mara.
  ENDMETHOD.
  METHOD get_text.
    SELECT SINGLE maktx FROM makt INTO @DATA(text)
     WHERE matnr = @mara-matnr
       AND spras = @sy-langu.
    IF sy-subrc = 0.
      MESSAGE text TYPE 'I'.
    ENDIF.
  ENDMETHOD.
ENDCLASS.

CLASS lcl_user_object_mtart DEFINITION INHERITING FROM lcl_user_object_mara.
  PUBLIC SECTION.
    METHODS get_text REDEFINITION.
ENDCLASS.

CLASS lcl_user_object_mtart IMPLEMENTATION.
  METHOD get_text.
    MESSAGE |Materialart { mara-mtart }|  TYPE 'I'.
  ENDMETHOD.
ENDCLASS.
CLASS lcl_user_object_matnr DEFINITION INHERITING FROM lcl_user_object_mara.
  PUBLIC SECTION.
ENDCLASS.



CLASS lcl_main DEFINITION.
```

```
  PUBLIC SECTION.
    CLASS-METHODS get_data.
    CLASS-METHODS create_tree.
    CLASS-METHODS add_nodes.
    CLASS-METHODS create_docker.
    CLASS-METHODS handle_node_double_click
      FOR EVENT node_double_click
                  OF cl_simple_tree_model
      IMPORTING node_key.

    CLASS-DATA docker TYPE REF TO cl_gui_docking_container.
    CLASS-DATA tree   TYPE REF TO cl_simple_tree_model.
    CLASS-DATA t_mara TYPE STANDARD TABLE OF mara.
    CLASS-DATA s_mara TYPE mara.

ENDCLASS.

CLASS lcl_main IMPLEMENTATION.
  METHOD create_docker.
    IF docker IS INITIAL.
      docker = NEW #( side = cl_gui_docking_container=>dock_at_bottom ratio =
50 ).
    ENDIF.
  ENDMETHOD.

  METHOD handle_node_double_click.

    DATA o_user_object_matnr TYPE REF TO lcl_user_object_matnr.
    DATA o_user_object_mtart TYPE REF TO lcl_user_object_mtart.
    DATA o_object TYPE REF TO object.

    tree->node_get_user_object( EXPORTING node_key    = node_key
                                IMPORTING user_object = o_object ).

    IF o_object IS INSTANCE OF lcl_user_object_mtart.
      o_user_object_mtart ?= o_object.
      o_user_object_mtart->get_text( ).
      EXIT.
    ENDIF.

    IF o_object IS INSTANCE OF lcl_user_object_matnr.
      o_user_object_matnr ?= o_object.
      o_user_object_matnr->get_text( ).
      EXIT.
    ENDIF.

  ENDMETHOD.


  METHOD get_data.
```

```
    SELECT * FROM mara INTO TABLE t_mara
     WHERE matnr IN s_matnr
       AND mtart IN s_mtart.
    IF sy-subrc > 0.
      RAISE EXCEPTION TYPE lcx_error.
    ENDIF.
  ENDMETHOD.

  METHOD create_tree.

    IF tree IS BOUND.
      tree->delete_all_nodes( ).
    ELSE.
      tree   = NEW #( node_selection_mode =
cl_simple_tree_model=>node_sel_mode_single ).
      tree->create_tree_control( EXPORTING parent = docker ).

      tree->set_registered_events(
              EXPORTING events = VALUE #(
                 ( eventid = cl_simple_tree_model=>eventid_node_double_click
appl_event = space ) ) ).

      SET HANDLER handle_node_double_click FOR tree.
    ENDIF.

    tree->add_node(
             node_key = 'Root'                                "#EC NOTEXT
             isfolder = 'X'
             text     = 'Materialien zur Selektion' ).

  ENDMETHOD.

  METHOD add_nodes.

    DATA l_mtart TYPE mtart.

    SORT t_mara BY mtart matnr.

    LOOP AT t_mara INTO DATA(mara).

      IF l_mtart <> mara-mtart.
        l_mtart = mara-mtart.
        "Knoten MATERIALART
        tree->add_node(
            node_key = CONV #( mara-mtart )
            relative_node_key = 'Root'
            relationship = cl_simple_tree_model=>relat_last_child
            isfolder = 'X'
            text     = |Materialart { mara-mtart } |
            user_object = NEW lcl_user_object_mtart( i_mara = mara ) ).
      ENDIF.
```

```
      "Knoten MATERIALNUMMER
      tree->add_node(
          node_key          = |{ mara-matnr ALPHA = OUT }|
          relative_node_key = CONV #( mara-mtart )
          relationship      = cl_simple_tree_model=>relat_last_child
          isfolder          = space
          text              = |{ mara-matnr ALPHA = OUT }|
          user_object       = NEW lcl_user_object_matnr( i_mara = mara ) ).
      .

    ENDLOOP.

    tree->expand_root_nodes( ).


  ENDMETHOD.

ENDCLASS.



INITIALIZATION.
  lcl_main=>create_docker( ).

AT SELECTION-SCREEN.
  TRY.
      lcl_main=>get_data( ).
      lcl_main=>create_tree( ).
      lcl_main=>add_nodes( ).
    CATCH lcx_error.
      MESSAGE 'Fehler bei Selektion' TYPE 'I'.
  ENDTRY.
```