JSON-Import

Konfigurierbarer JSON Import

JSON Import - Agenda

1. Fest kodierter Import

2. Konfigurierbare Import
   1. JSON als TreeView darstellen
   2. Zuordnung von JSON Elementen zu Datawindow Columns
   3. Import der Daten an Hand der Zuordnung

Fest codierter Import

Google-Maps: Geo-JSON


```
{
   "results": [
      {
         "address_components": [
            {
               "long_name": "nhow Berlin",
               "short_name": "nhow Berlin",
               "types": [
                  "premise"
               ]
            },
            {
               "long_name": "3",
               "short_name": "3",
               "types": [
                  "street_number"
               ]
            },
```

```
{
               "long_name": "Stralauer Allee",
               "short_name": "Stralauer Allee",
               "types": [
                  "route"
               ]
            },
            {
               "long_name": "Bezirk Friedrichshain-Kreuzberg",
               "short_name": "Bezirk Friedrichshain-Kreuzberg",
               "types": [
                  "political",
                  "sublocality",
                  "sublocality_level_1"
               ]
            },
```

Fest kodierter Import

JSON-Elemente einzeln auslesen

```
lnvo_json.loadstring( as_json)
// Die Anzahl der Child Elemente ermitteln
ll_children = lnvo_json.getChildCount(lnvo_json.getItemarray( "/results/1/address_components"))
for ll_i = 1 to  ll_children
  ls_type = lnvo_json.getItemString( "/results/1/address_components/"+string(ll_i) +"/types/1")
  choose case ls_type
    case "street_number"// „Hausnummer"
        astr_adresse.adr_hnr1 = lnvo_json.getItemString( "/results/1/address_components/"+string(ll_i) +"/long_name")
       // astr_adresse.adr_hnr1 = lnvo_json.getItemString( "/results/1/address_components/2/long_name")
   case "route"//  „Strasse",
      astr_adresse.adr_str= lnvo_json.getItemString( "/results/1/address_components/"+string(ll_i) +"/long_name")
   case "political"      // „Ortszeil"
     astr_adresse.adr_ortsteil= lnvo_json.getItemString( "/results/1/address_components/"+string(ll_i) +"/long_name")
   case "locality„
              ……..
```

Fest kodierter Import

Anmerkungen

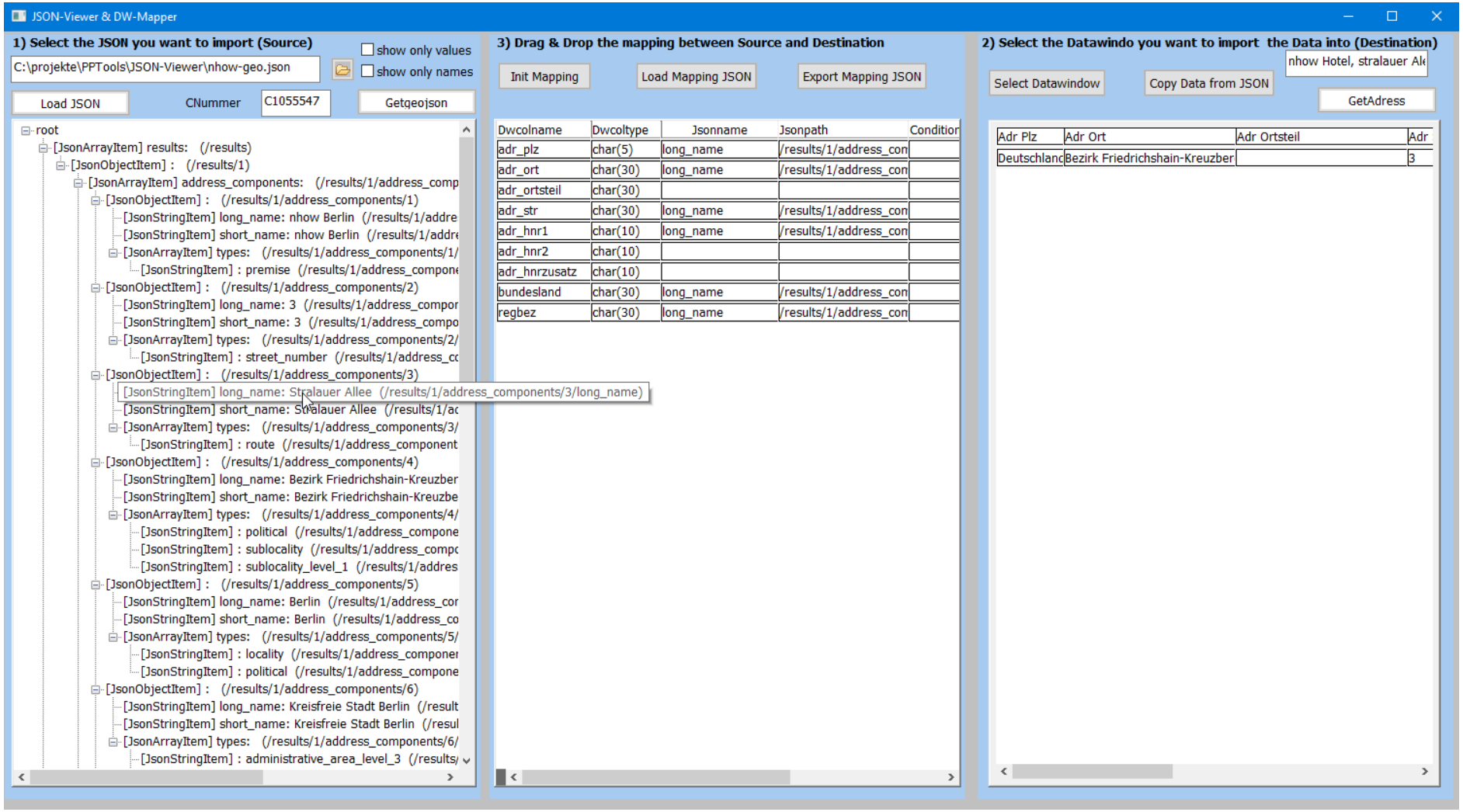
- Man muss das JSON sehr genau kennen
  - Beim Beispiel haben die Adresskomponenten keine Namen. Deshalb ist es schwierig den Objektpfad zu bestimmen
- Änderungen sind mühsam und fehleranfällig
- Tests sind dem entsprechend aufwändig

Konfigurierbare Import

Drei Schritte zum Ergebnis:

1. JSON als TreeView darstellen
2. Ein Mapping zwischen JSON und DataWindow Columns
3. Kopieren der Daten aus dem JSON in das Datawindow an Hand des Mappings

„JSON-VIEWER“


JSON-Viewer & DW-Mapper
1) Select the JSON you want to import (Source)
C:\projekte\PPTools\JSON-Viewer\nhow-geo.json
show only values
show only names
Load JSON
CNummer
C1055547
Getqeojson
root
[JsonArrayItem] results: (/results)
[JsonObjectItem] : (/results/1)
[JsonArrayItem] address_components: (/results/1/address_comp
[JsonObjectItem] : (/results/1/address_components/1)
[JsonStringItem] long_name: nhow Berlin (/results/1/addre
[JsonStringItem] short_name: nhow Berlin (/results/1/addre
[JsonArrayItem] types: (/results/1/address_components/1/
[JsonStringItem] : premise (/results/1/address_compone
[JsonObjectItem] : (/results/1/address_components/2)
[JsonStringItem] long_name: 3 (/results/1/address_compor
[JsonStringItem] short_name: 3 (/results/1/address_compo
[JsonArrayItem] types: (/results/1/address_components/2/
[JsonStringItem] : street_number (/results/1/address_co
[JsonObjectItem] : (/results/1/address_components/3)
[JsonStringItem] long_name: Stralauer Allee (/results/1/address_components/3/long_name)
[JsonStringItem] short_name: Stralauer Allee (/results/1/ac
[JsonArrayItem] types: (/results/1/address_components/3/
[JsonStringItem] : route (/results/1/address_component
[JsonObjectItem] : (/results/1/address_components/4)
[JsonStringItem] long_name: Bezirk Friedrichshain-Kreuzber
[JsonStringItem] short_name: Bezirk Friedrichshain-Kreuzbe
[JsonArrayItem] types: (/results/1/address_components/4/
[JsonStringItem] : political (/results/1/address_compone
[JsonStringItem] : sublocality (/results/1/address_compo
[JsonStringItem] : sublocality_level_1 (/results/1/addres
[JsonObjectItem] : (/results/1/address_components/5)
[JsonStringItem] long_name: Berlin (/results/1/address_cor
[JsonStringItem] short_name: Berlin (/results/1/address_co
[JsonArrayItem] types: (/results/1/address_components/5/
[JsonStringItem] : locality (/results/1/address_componer
[JsonStringItem] : political (/results/1/address_compone
[JsonObjectItem] : (/results/1/address_components/6)
[JsonStringItem] long_name: Kreisfreie Stadt Berlin (/result
[JsonStringItem] short_name: Kreisfreie Stadt Berlin (/resul
[JsonArrayItem] types: (/results/1/address_components/6/
[JsonStringItem] : administrative_area_level_3 (/results/
3) Drag & Drop the mapping between Source and Destination
Init Mapping
Load Mapping JSON
Export Mapping JSON
Dwcolname | Dwcoltype | Jsonname | Jsonpath | Condition
adr_plz | char(5) | long_name | /results/1/address_con
adr_ort | char(30) | long_name | /results/1/address_con
adr_ortsteil | char(30)
adr_str | char(30) | long_name | /results/1/address_con
adr_hnr1 | char(10) | long_name | /results/1/address_con
adr_hnr2 | char(10)
adr_hnrzusatz | char(10)
bundesland | char(30) | long_name | /results/1/address_con
regbez | char(30) | long_name | /results/1/address_con
2) Select the Datawindo you want to import the Data into (Destination)
nhow Hotel, stralauer Al
Select Datawindow
Copy Data from JSON
GetAdress
Adr Plz | Adr Ort | Adr Ortsteil | Adr
Deutschlanc | Bezirk Friedrichshain-Kreuzber | | 3

JSON als TreeView darstellen

Kurze Wiederholung: Was ist JSON (JavaScript Object Notation)

JSON besteht aus einer Liste von Elementen.
Es gibt die folgenden Typen von Elementen.

Nullwert

wird durch das Schlüsselwort ***null*** dargestellt.

Boolescher Wert

wird durch die Schlüsselwörter ***true*** und ***false*** dargestellt. Dies sind keine Zeichenketten. Sie werden daher, wie null, nicht in Anführungszeichen gesetzt.

Zahl

ist eine Folge der Ziffern 0–9. Diese Folge kann durch ein negatives Vorzeichen - eingeleitet und durch einen Dezimalpunkt . unterbrochen sein. Die Zahl kann durch die Angabe eines Exponenten e oder E ergänzt werden, dem ein optionales Vorzeichen + oder - und eine Folge der Ziffern 0–9 folgt.

Zeichenkette

beginnt und endet mit doppelten geraden Anführungszeichen ("). Sie kann Unicode-Zeichen und durch \ eingeleitete Escape-Sequenzen enthalten.

Array

beginnt mit [und endet mit]. Es enthält eine durch Kommata geteilte, indizierte Liste von Elementen gleichen oder verschiedenen Typs. Leere Arrays sind zulässig.

Objekt

beginnt mit { und endet mit }. Es enthält eine durch Kommata geteilte, ungeordnete Liste von Eigenschaften. Objekte ohne Eigenschaften („leere Objekte“) sind zulässig.

besteht aus einem Schlüssel und einem Wert, (Schlüssel : Wert).

der Schlüssel ist eine Zeichenkette.

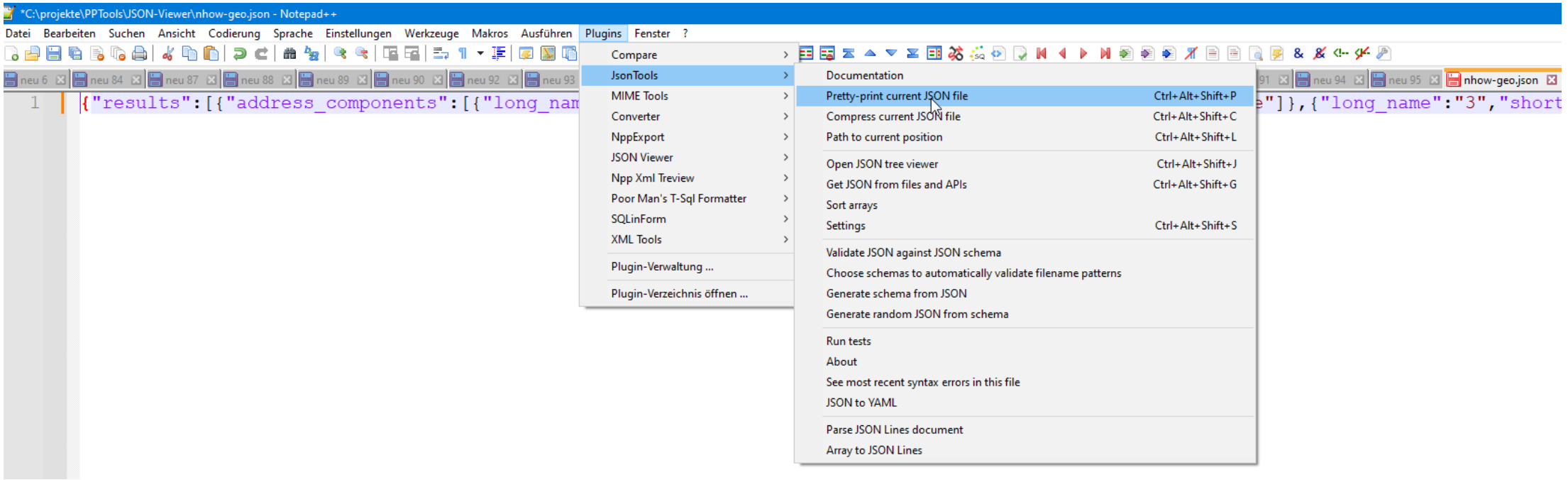
der Wert ist ein beliebiges Element.

JSON als TreeView darstellen

Beispiel

Ein JSON mit Strukturelementen lässt sich aus Baum / TreeView darstellen.

- Editoren, wie Notepad++ haben das als Plugin.



JSON TreeView mit PowerBuilder

Das will ich auch mit **PowerBuilder** machen!

Aufgabe:

- Das JSON Element für Element zu durchlaufen.
- Bei jedem Element entscheiden, ob es ein *einfaches* oder *strukturiertes* Element ist.
- Einfache Elemente werden im Baum als Geschwisterkind (sibling) auf der gleichen Ebene zugefügt.
- Strukturierte Elemente als Kind (child) **eine Ebene tiefer**. (Rekursiver Aufruf)

JSON TreeView mit PowerBuilder

Verwenden des JSONParser Objects

Initialisieren

```
inv_JsonParser.loadstring( as_json )
```

```
// Get JSON Root
ll_RootObject = inv_JsonParser.GetRootItem()

// Examine Tree
of_parse_child(ll_rootobject, ll_tvitem)
```

JSON Parsen

```
of_parse_child(long al_parent_js, long al_parent_tvi)

ll_childcnt = inv_JsonParser.GetChildCount(al_parent_js)
// Examine each Child
For ll_i = 1 to ll_childcnt

  lb_children = FALSE
  // GetItemkey  (Name of the Item)
  ls_ItemKey =  inv_JsonParser.getChildKey(al_parent_js, ll_i)
  // GetChildItem (the handle of the child item)
  ll_Childitem = inv_JsonParser.getChildItem(al_parent_js, ll_i)
  if ls_Itemkey <> "" then
    ls_jsonpath = is_jsonpath+ "/"+ls_ItemKey
  else
    ls_jsonpath = is_jsonpath+ "/"+string(ll_i)
  end if
  // GetItemValue or process object depending on type
  l_jitype = inv_JsonParser.GetItemType(ll_Childitem)
  choose case  inv_JsonParser.GetItemType(ll_Childitem)
```

Die JSON-Elemente untersuchen

```
choose case  inv_JsonParser.GetItemType(ll_Childitem)
   case JsonObjectItem!
   ls_type = "JsonObjectItem"
   ls_ItemValue=""
   if inv_JsonParser.GetChildCount(ll_Childitem) > 0 then
      lb_children = TRUE
   end if
   case JsonArrayItem!
      ls_type = "JsonArrayItem"
      ls_ItemValue=""
      if inv_JsonParser.GetChildCount(ll_Childitem) > 0 then
         lb_children = TRUE
      end if
   case JsonStringItem!
   ls_type = "JsonStringItem"
   ls_ItemValue = inv_JsonParser.GetItemString(ll_Childitem)
case JsonNumberItem!
```

```
   case JsonNumberItem!
      ls_type = "JsonNumberItem"
      ls_ItemValue =
         string(inv_JsonParser.GetItemNumber(ll_Childitem))
   case JsonBooleanItem!
      ls_type = "JsonBooleanItem"
      ls_ItemValue =
        string(inv_JsonParser.GetItemBoolean(ll_Childitem))
   case JsonNullItem!
      ls_type = "JsonNullItem"
      ls_ItemValue = 'Null‘
end choose
```

Den TreeView erstellen

```
//Build label for Treeview
 if ib_names_only then
   ls_label = ls_ItemKey
else
    if ib_values_only then
        ls_label = ls_ItemValue
   else // show structure with naes and values
        ls_label = "[" + ls_type + "] " + ls_ItemKey + ": "  +
            ls_ItemValue + "  ("+ls_jsonpath +")"
    end if
end if
```

```
// Build the treeview Item
lnvo_iteminfo.is_itemkey = ls_ItemKey
lnvo_iteminfo.is_itemvalue = ls_ItemValue
lnvo_iteminfo.is_itempath = ls_jsonpath
lnvo_iteminfo.is_itemtype = ls_type

ltvi_tvi.Label = ls_label
ltvi_tvi.Children = lb_children
ltvi_tvi.data = lnvo_iteminfo

// Paint the knot
ll_tvi_handle = this.insertitemlast( al_parent_tvi,
ltvi_tvi)
```

Strukturen rekursive bearbeiten

```
   // Ist das JSON Element eine Strukture: parse it!
   choose case  inv_JsonParser.GetItemType(ll_Childitem)
      case JsonObjectItem! ,JsonArrayItem!
         // Diese Funktion recursive aufrufen
         is_jsonpath = ls_jsonpath  // Den ItemPath für dieses Level  in einer Instanzvariable speichern und …
         of_parse_child( ll_Childitem, ll_tvi_handle)
         // …. nach Rückkehr aus Rekursion den ItemPath zurücksetzten
         is_jsonpath = left(is_jsonpath,lastpos(is_jsonpath,"/") -1)
      end choose
next
```

Mapping von JSON-Elementen mit Datawindow-Columns

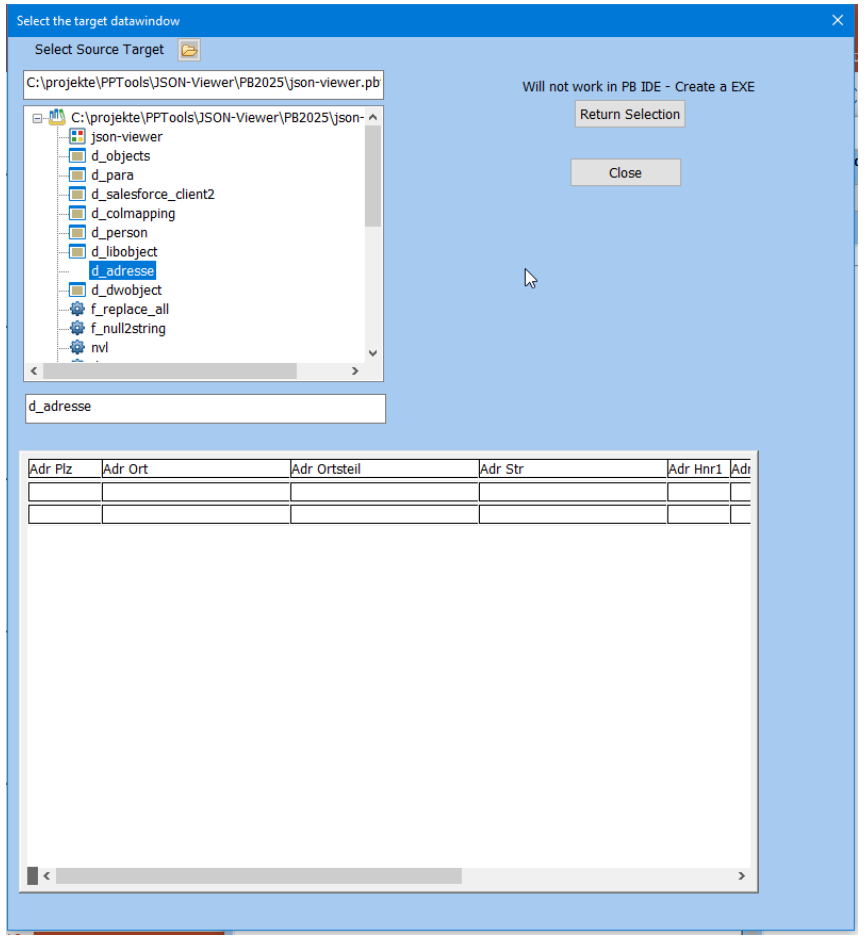
Ziel (Wiederholung):

Die Daten aus dem JSON in eine Datawindow importieren:

1. Das Ziel-Datawindow festlegen
2. Mit den Columns dieses Datawindows eine Mappingtabelle initialisieren
3. Per Drag&Drop aus dem JSON Tree die Zuordnung machen

Ziel-Datawindow festlegen

- Auswahl über EXE
  (Von Bruce Armstrong)



Mappingtabelle initialisieren

Dwcolname	Dwcoltype	Jsonname	Jsonpath	Condition	Jsontype

Dwcolname	Dwcoltype	Jsonnar
adr_plz	char(5)	
adr_ort	char(30)	
adr_ortsteil	char(30)	
adr_str	char(30)	
adr_hnr1	char(10)	
adr_hnr2	char(10)	
adr_hnrzusatz	char(10)	
bundesland	char(30)	
regbez	char(30)	

```
stringls_objects[],&
ls_coltypes[]
longll_rc ,&
ll_row

datastorelds_ds
lds_ds = create datastore

dw_mapping.reset()
lds_ds.dataobject = dw_preview.dataobject

ll_rc = of_getobjects( ls_objects[], ls_coltypes[], "column", "detail", true, lds_ds)
for ll_row = 1 to upperbound(ls_objects)
   ll_rc = dw_mapping.insertrow(0)
   dw_mapping.setItem(ll_row,"dwcolname",ls_objects[ll_row])
   dw_mapping.setItem(ll_row,"dwcoltype",ls_coltypes[ll_row] )
next
```

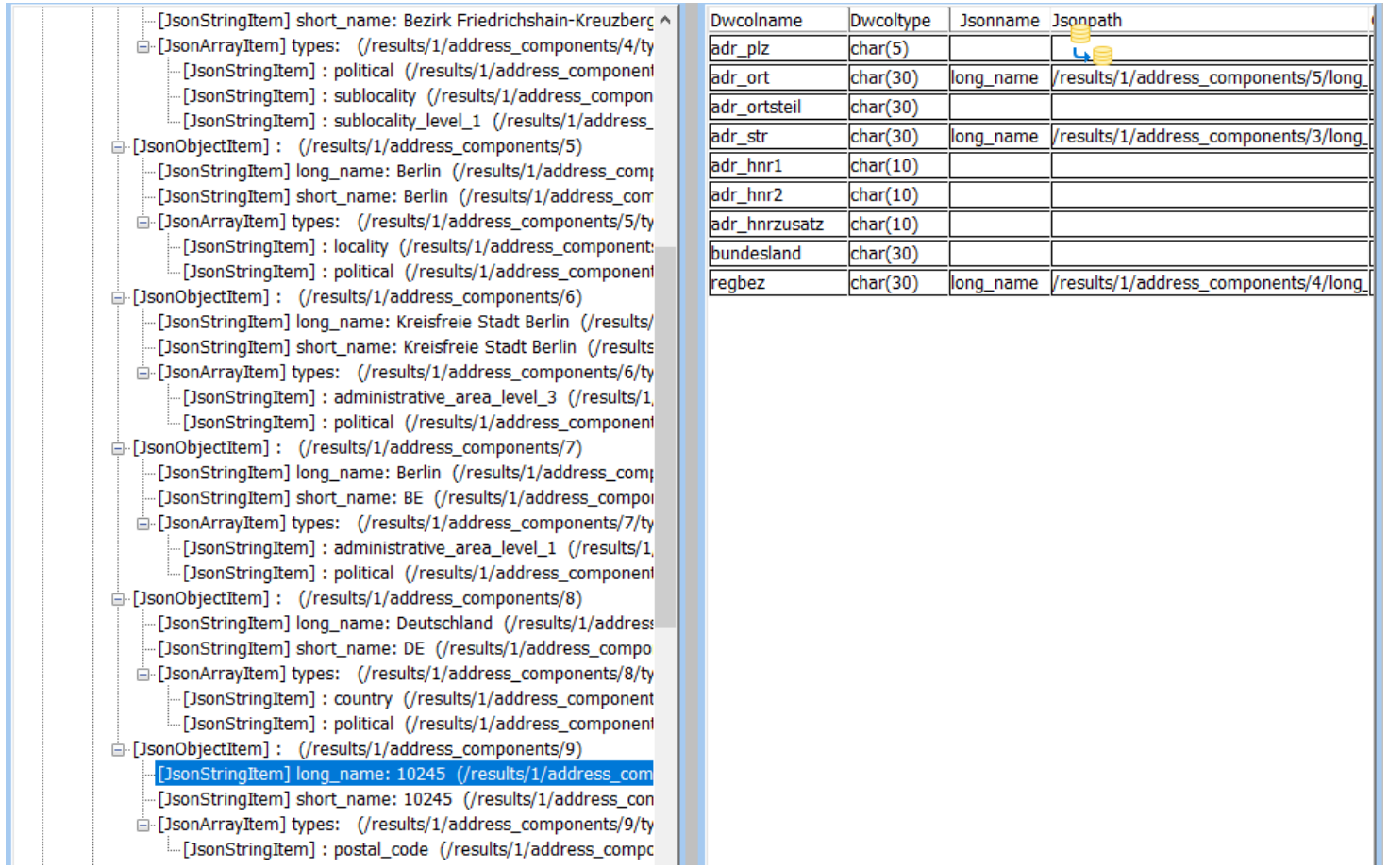
Mapping

Drag & Drop

Kopiert werden:

- Der JSON Name
- JSON Path
- JSON Type

Export als JSON



Dwcolname	Dwcoltype	Jsonname	Jsonpath
adr_plz	char(5)		
adr_ort	char(30)	long_name	/results/1/address_components/5/long_
adr_ortsteil	char(30)		
adr_str	char(30)	long_name	/results/1/address_components/3/long_
adr_hnr1	char(10)		
adr_hnr2	char(10)		
adr_hnrzusatz	char(10)		
bundesland	char(30)		
regbez	char(30)	long_name	/results/1/address_components/4/long_

Daten Kopieren

lnvo_n_cst_jsontree.

of_parse_json2ds(ls_responsejson,lds_ds, ls_mappingjson)

```
// Mapping als JSON
ll_rc  = ids_mapping.importJSON( as_mappingrule,ls_error)

//ll_rc = of_loadfile(as_jsonpath)
ll_rc = of_loadstring(as_jsonpath)

// Get JSON Root
ll_RootObject = inv_JsonParser.GetRootItem()

of_parse_child_dsfirst(ll_rootobject , ads_ds)
```

```
of_parse_child_dsfirst(long al_parent_js , datastore ads_ds)
...
ll_childcnt = inv_JsonParser.GetChildCount(al_parent_js)
For ll_i = 1 to ll_childcnt
   ll_valuecnt = 0
   ll_newrow = ads_ds.insertrow(0)
   // Für alle Columns aus dem Mapping (Sollte dem Ziel DS
        entsprechen) nach Werten im JSON suchen
   for ll_row = 1 to rowcount(ids_mapping)
      setnull(la_value)
      ls_jsonpath =  ids_mapping.getItemString(ll_row, "jsonpath")
      if trim(ls_jsonpath) = "" or isnull(ls_jsonpath) then
         continue
      end if
      ls_jsonname = ids_mapping.getItemString(ll_row, "jsonname")
      ls_dwcolname = ids_mapping.getItemString(ll_row, "dwcolname")
    // Vor dem Kopieren aus JSONParser in DW, den Pfad für das
         aktuelle "ROW" im JSON anpassen ->
         jsonpath das erste   /1 gegen /<ll_i> tauschen
     ls_jsonpath = of_setpath4row(ls_jsonpath,ll_i)
```

```
    // Jetzt den Wert holen
   la_value = of_getItemValue(ls_jsonpath)
   if not isnull(la_value) then
      ads_ds.setItem(ll_newrow, ls_dwcolname,la_value)
      ll_valuecnt ++
   end if
   next
   if ll_valuecnt = 0 then// Wenn keine Werte eingetragen
werden, das Row wieder löschen
      ads_ds.deleterow(ll_newrow)
   end if
   next
return ll_valuecnt
```

Fragen

Noch Fragen?

Das Ende naht

Danke, tschüss, auf Wiedersehen!