Generating a Listbox using Dynamic forms

By Vance Villanueva, Technical Services Engineer, 4D Inc.

Technical Note 18-16

Table of Contents

Table of Contents 2
Introduction 3
Overview 3
Conventional form vs Dynamic form design 3
  Conventional Approach 3
    Setting up the Listbox in the form 3
      Current/Name selection based 4
      Array based 4
      Collection/Entity selection based 4
    Programming the Listbox 5
      Current/Name selection based 5
      Array based 5
      Collection/Entity selection based 5
    Opening the form 6
  Dynamic approach 6
    JSON definition 6
  Form definition with one object 7
Implementation of the Dynamic Listbox 8
  Listbox definition 8
  Listbox column definition 9
  Listbox data source configurations 9
    Selection based 9
    Array based 11
    Collection based 12
    Entity Selection based 13
Sample database 14
  Setup Listbox Form definition Method 15
    Calling Selection based 17
    Calling Array based 17
    Calling Collection based 18
    Calling Entity Selection based 18
  Displaying using Subform 18
  Displaying JSON definition 19
Conclusion 20
Reference 20

Introduction

The ability to create Dynamic forms is a feature that is implemented starting in v16R6. An interesting part of this feature is that one can dynamically build each form object definition with fewer pieces of code and without a form design. For a Listbox, the conventional design approach is to setup a form as well as set the data source to bind the data in each column through code or in the Listbox property settings via form editor. But with Dynamic forms, there is an efficient process which can be built through a JSON definition. By simply creating a JSON form definition through code or in a JSON file, 4D can easily display it. The power of generating a fully rich Listbox without the need of designing through a form editor can be very useful as well as an efficient practice for existing and new databases. This technical note will breakdown the usage of generating Dynamic forms containing a Listbox as well as demonstrate various implementations through a sample database.

Overview

As a 4D developer, designing a form that contains data from a database and/or other sources undergoes a series of steps to display at runtime. A Listbox object for example needs to be dropped into a form through a form editor. To display contents in a Listbox, the data needs to be binded in a column through the Listbox property settings or via 4D code. Then finally to display the form, 4D code is called. This whole process can be minimized through only 4D code thanks to the feature of Dynamic forms of executing a single JSON definition, which possesses many benefits. The document will dive into the conventional design to a Dynamic form design breakdown and will be demonstrated through a sample database.

Note: Database must be in v16R6 or greater to use the new features.

Conventional form vs Dynamic form design

The following section will look at the conventional approach of creating a Listbox to the new feature of creating it simply by definition in a Dynamic form!

Conventional Approach

Upon displaying a Listbox, it needs to be placed onto a form. Once a Listbox is placed in the form, it will require setting up the properties in the property list and code to display the data.

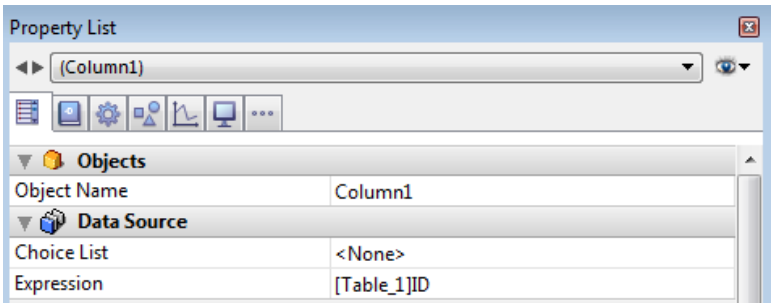
Setting up the Listbox in the form

When setting the Listbox, one must decide what kind of data source type to use. The standard types are current/named selection or arrays. But in v16R6 and greater, there

are other options with collection and entity selection. Depending on the data source type, one must setup the variables and columns accordingly. The next areas will discuss the setup on various data source type for the columns:

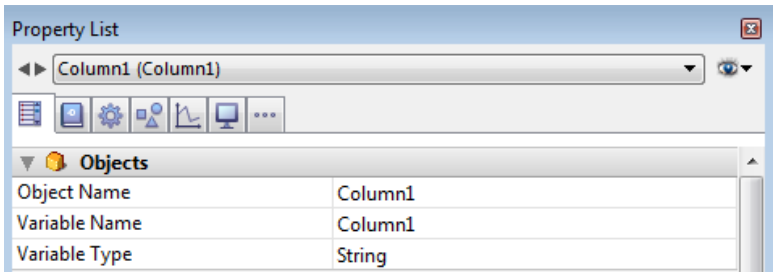
Current/Name selection based

This type of Listbox requires an “Expression” for each column. An example below sets up the column to read from “[Table_1]ID”:



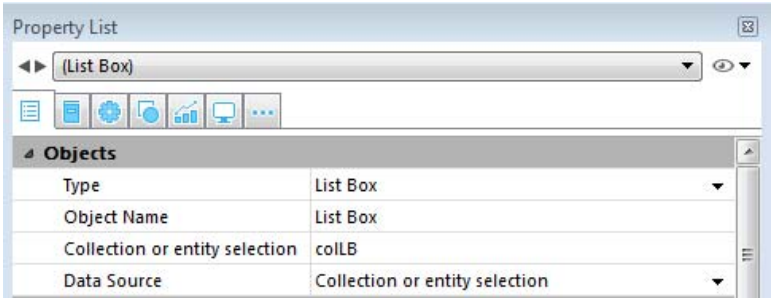
Array based

This type of Listbox requires an “Object/Variable” name for each column. An example below sets up the column to read in “Column1” array.

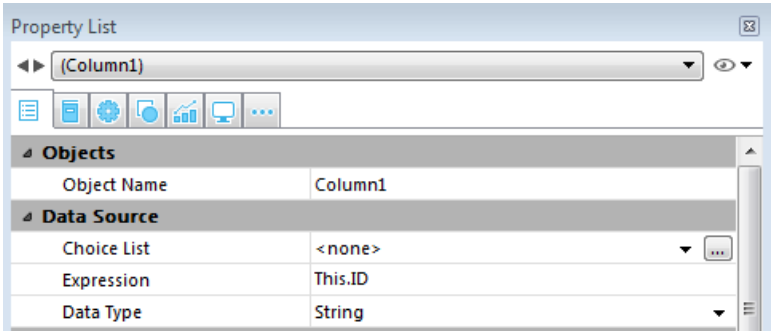


Collection/Entity selection based

This type of Listbox requires a collection or object variable in the Listbox properties. If the variable is an object, then using ORDA for Entity selection can be referenced. The example below uses the “colLB” variable:



To display columns, the key word “This” is used with the property to be accessed in a collection containing 4D objects or field from an Entity selection referenced into an object. The example below assigns a column:



Programming the Listbox

The focus of programming the Listbox is to explain how the data is binded to the Listbox columns. The areas below will describe how each data source connects to the columns.

Current/Name selection based

Since this is record based, a query is performed to get the selection such as 4D commands like **QUERY** and **ALL RECORDS**.

Array based

Each column can be referenced by the array that is initialized. An example is the following:

```
ARRAY TEXT(Column1;0)

APPEND TO ARRAY(Column1;"Tom")
APPEND TO ARRAY(Column1;"Bob")
```

Collection/Entity selection based

The example above used “colLB” for collection variable. A collection is used with set of objects. Here is an example of setting up a collection variable containing objects:

```
C_COLLECTION(colLB)
colLB:=New collection(New object("name";"Tom");New object("name";"Bob"))
```

For Entity selection based, an object is created from the records that are queried with ORDA. Here is an example of using “eSel”:

```
C_OBJECT(eSel)
eSel:=ds.Table_1.all()
```

Opening the form

When a Listbox has been setup on the property list and/or through code, the form would need to be opened for display. Typically, it is opened using **Open form window** or **Open window**.

This gives a general idea of how a Listbox is setup typically through a property list and with code. The next section will discuss a dynamic approach of building a Listbox purely on code only.

Dynamic approach

When building a Listbox originally, the property list is set and code is required to bind the data as well as open the form. To build a Listbox dynamically in a form it does not require creating a form and placing a Listbox object. To open a form it takes a JSON definition in the **Open form window** command.

JSON definition

In v16R6, Dynamic forms can be created with this basic JSON form definition.

```
{
    "pages": [
        null,
        {
            "objects": {}
        }
    ]
}
```

The same definition can be created as a 4D object in code as the following:

```
C_OBJECT($page;$form)

$page:=New object("objects";New object)
$form:=New object("pages";New collection(Null;$page))
```

Form definition with one object

In taking this a step further, under the “objects” property, a text object is placed in the form as a simple example.

```
{
    "pages": [
        null,
        {
            "objects": {
                "TEXT": {
                    "type": "text",
                    "text": "test",
                    "left": 70,
                    "top": 70,
                    "width": 100,
                    "height": 100
                }
            }
        }
    ]
}
```

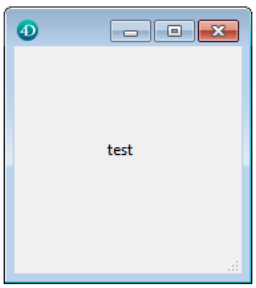
The same definition can be created as a 4D object in code as the following:

```
C_OBJECT($obj;$page;$form)
C_LONGINT($window)

$obj:=new
object("type";"text";"text";"test";"left";70;"top";70;"width";100;"height";100)
$page:=New object("objects";New object("TEXT";$obj))
$form:=New object("pages";New collection(Null;$page))

$window:=Open form window($form)
DIALOG($form)
CLOSE WINDOW($window)
```

When it is displayed, it will be as the following using **Open form window** by taking an object as the parameter:



Overall both approaches have their pros and cons to implementation. The focus of this document is to examine the Dynamic interface aspect as the next section will explain the implementation in depth.

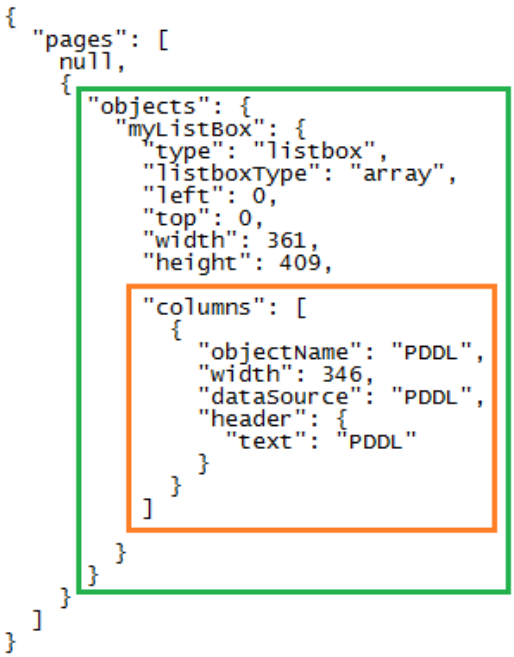
Implementation of the Dynamic Listbox

From the basic form definition mentioned above it is possible to build a Listbox object definition very easily with simple properties. These examples are the very minimum requirements to display a Listbox with data. Other properties can be set to enhance the object. Please refer to the reference below. This information is a basic example to build upon. It is broken down into three sections:

- Listbox definition
- Listbox column definition
- Listbox data source configurations

Listbox definition

The Listbox definition in “green” is the minimum configuration in setting up the Listbox. The next section will describe the column definition in “orange”. For this area, the Listbox type “listboxType” property needs to be specified to determine the type whether it is “Selection”, “Array”, “Collection”, or “Entity Selection”. Because this Listbox will be opened in a Dynamic form, the object’s coordinates and size require a location with “left”, “top”, “width”, and “height” respectfully as shown below:



Listbox column definition

Once a Listbox type, data source, location, and size are determined, a column "in orange" can be defined. A column will need a header title, data source, and an object name property at the minimum as shown above. The example above is for an array based Listbox where the "objectName" and "dataSource" are specified with the array used. This will be different on the data source type used.

When selecting the Listbox data source type, the configuration of the Listbox and Listbox column definition will be different on binding the data. The next section will explain those types.

Listbox data source configurations

The following data source configurations are covered:

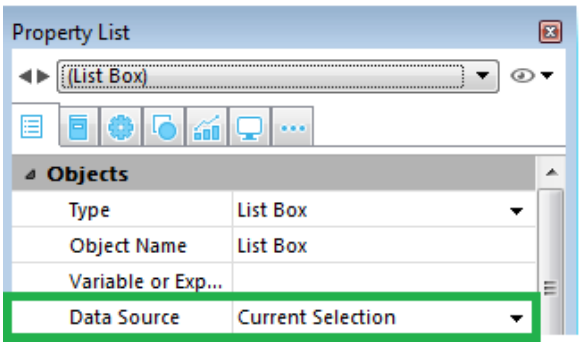
- Selection based
- Array based
- Collection based
- Entity Selection based

Selection based

The following definition is an example of one column that requires a table and field name:

```
{
    "pages": [
        null,
        {
            "objects": {
                "myListBox": {
                    "type": "listbox",
                    "listboxType": "currentSelection",
                    "table": {
                        "$ref": "#/info/table"
                    },
                    "left": 0,
                    "top": 0,
                    "width": 300,
                    "height": 270,
                    "columns": [
                        {
                            "objectName": "colID",
                            "width": 283,
                            "dataSource": {
                                "$ref": "#/info/field1"
                            },
                            "header": {
                                "text": "ID1"
                            }
                        }
                    ]
                }
            }
        }
    ],
    "info": {
        "table": "Table_1",
        "field1": "[Table_1]ID"
    }
}
```

The property “info” contains the values for the table and field. A “table” property must be specified in the Listbox definition using “info”. For each column, a “dataSource” property is reference by field using “info”. This is like setting the Data Source in the property list as the following:



Another way to reference the “table” and “dataSource” properties can be naming the table and field name as shown below:

```
{
    "pages": [
        null,
        {
            "objects": {
                "myListBox": {
                    "type": "listbox",
                    "listboxType": "currentSelection",
                    "table": "Table_3",
                    "left": 0,
                    "top": 0,
                    "width": 300,
                    "height": 270,
                    "columns": [
                        {
                            "objectName": "colID",
                            "width": 283,
                            "dataSource": "[Table_1]ID",
                            "header": {
                                "text": "ID1"
                            }
                        }
                    ]
                }
            }
        }
    ]
}
```

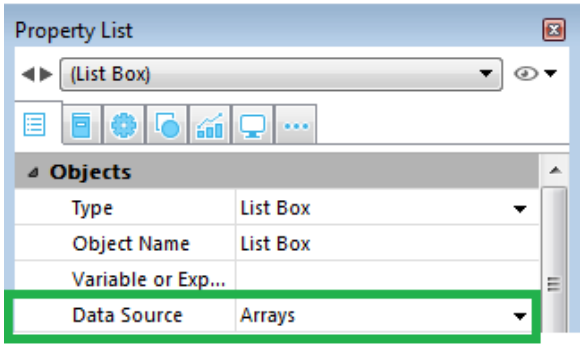
Array based

The following definition is an example of two columns that require two arrays specified:

```
{
    "pages": [
        null,
        {
            "objects": {
                "myListBox": {
                    "type": "listbox",
                    "listboxType": "array",
                    "left": 0,
                    "top": 0,
                    "width": 361,
                    "height": 409,
                    "columns": [
                        {
                            "objectName": "PDDL",
                            "width": 173,
                            "dataSource": "PDDL",
                            "header": {
                                "text": "PDDL"
                            }
                        },
                        {
                            "objectName": "PDDL1",
```

```
                        "width": 173,
                        "dataSource": "PDDL1",
                        "header": {
                            "text": "PDDL1"
                        }
                    }
                ]
            }
        }
    }
]
}
```

For each column, “objectName” and “dataSource” properties are referenced by the array created. In this example, the “PDDL” and “PDDL1” arrays were created of text type. This is like setting the Data Source in the property list as the following:



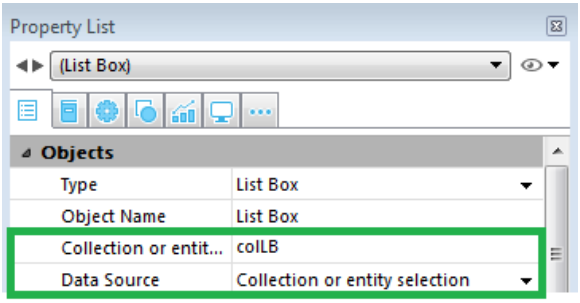
Collection based

The following definition is an example of one column that is a collection:

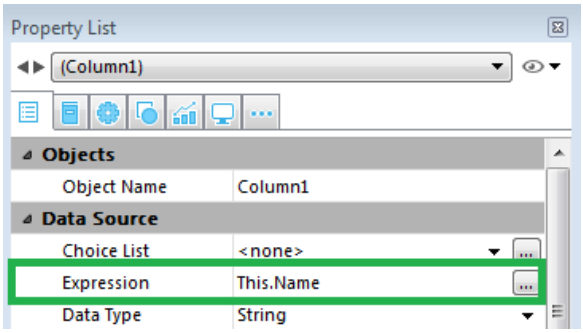
```
{
    "pages": [
        null,
        {
            "objects": {
                "myListBox": {
                    "type": "listbox",
                    "listboxType": "collection",
                    "dataSource": "colLB",
                    "left": 0,
                    "top": 0,
                    "width": 361,
                    "height": 409,
                    "columns": [
                        {
                            "objectName": "Name",
                            "width": 346,
                            "dataSource": "This.name",
                            "selectedItemsSource": "colLB",
                            "header": {
                                "text": "Name"
                            }
                        }
                    ]
```

```
                }
            }
        }
    ]
}
```

For a Collection based Listbox, a data source property needs to be specified similar to the property list below where the “colLB” variable is declared:



For each column, “objectName”, “dataSource”, and “selectedItemsSource” properties are referenced with “Name”, “This.name”, and “colLB” respectfully in green as mentioned above. The Listbox “Expression” for each column would be similar to setting in the Property List:



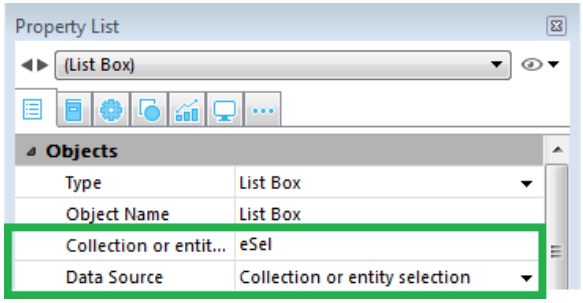
Entity Selection based

The following definition is an example of one column that is an Entity Selection:

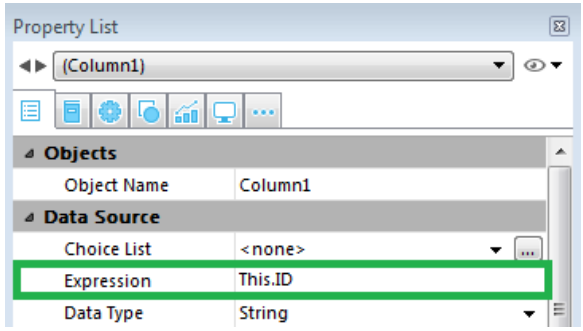
```
{
    "pages": [
        null,
        {
            "objects": {
                "myListBox": {
                    "type": "listbox",
                    "listboxType": "collection",
                    "dataSource": "eSel",
                    "left": 0,
                    "top": 0,
                    "width": 361,
                    "height": 409,
                    "columns": [
```

```
                    {
                        "objectName": "ID",
                        "width": 346,
                        "dataSource": "This.ID",
                        "header": {
                            "text": "ID"
                        }
                    }
                ]
            }
        }
    }
]
}
```

For an Entity based Listbox, a data source property needs to be a specified object variable similar to the property list below where the “eSel” variable is declared:



For each column, “objectName” and “dataSource” properties are referenced with “ID”, “This.ID” respectfully. The Listbox “Expression” for each column would be similar to setting in the Property List:

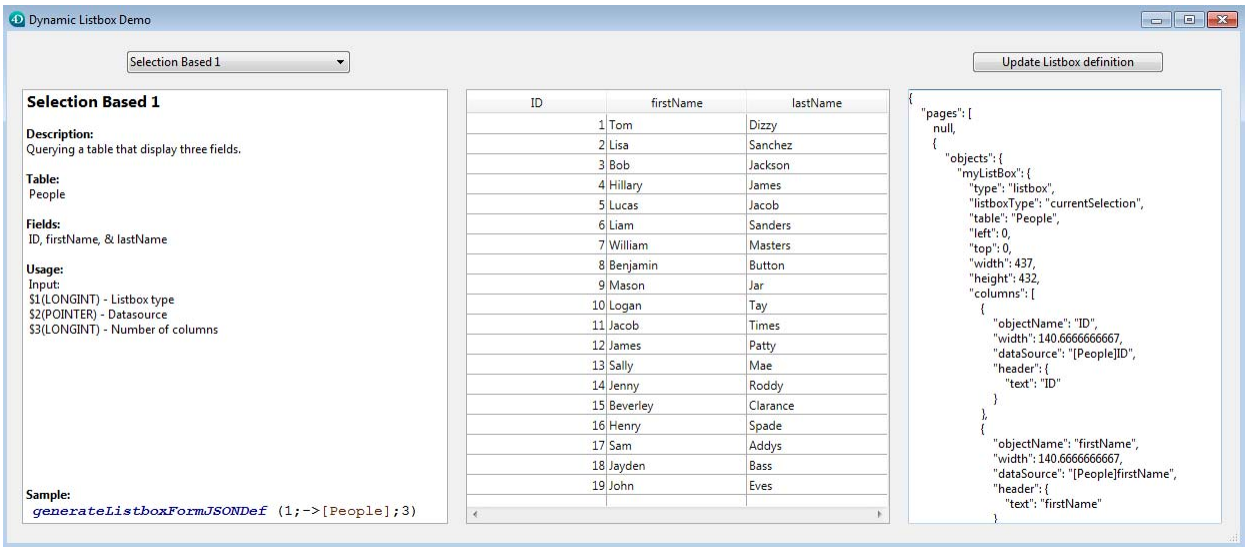


With this basic understanding on how to construct a Listbox dynamically with the definitions for the various types of Listboxes, the next section will demonstrate in a sample database.

Sample database

By simply creating a JSON definition of a Listbox, it is easily viewable with Dynamic forms. This section will demonstrate an ability to generate a Listbox JSON definition simply by calling a

method and then display through a subform. The following screen shot below is a make up of the demonstration (Demo is found in the menu as File->"Dynamic Listbox Demo"):



A popup drop down menu contains a selection of creating different Listbox data source types of selection, array, collection, and entity selection. When a item is selected, the text box below the drop down will give a description of a method that will be called to generate the JSON definition. When the method is called, the middle object which is the subform will output the JSON form definition with **OBJECT SET FORM**. The far right text box in the form will display the equivalent JSON definition. A button above can update the subform for manual manipulation. The next section will discuss the method that generates the JSON definition behind the scenes.

Setup Listbox Form definition Method

The following method below can generate a Listbox JSON definition of all data source types. By specifying the Listbox type, data source, and the number of columns, a 4D object is returned that can be used with commands like **OPEN FORM WINDOW** and **OBJECT SET FORM** to display Dynamic forms. With the use of **EXECUTE FORMULA**, variables can be dynamically created to individually create new objects. This method used the base 4D code structure mentioned in the beginning of the document. From there it is able to generate the bare minimum JSON definition to display a Listbox dynamically.

```
// ----------------------------------------------------------------
// Name: generateListboxFormJSONDef
// Description: Method will create a JSON form definition containing
// a Listbox object specifying the type and data source.
//
// Input Parameters:
// $1 (LONGINT) - Choice of Listbox type
//                1 - Selection based
//                2 - Array based
//                3 - Collection based
//                4 - Entity Selection based
// $2 (POINTER) - Data source (Table selection, Array,
```

```
  //                            Collection, or Entity Selection)
  // $3 (POINTER) - Number of columns or Array of columns
  // Output:
  // $0 (OBJECT) - JSON form definition
  // ----------------------------------------------------------------
C_LONGINT($1;$choice)
C_POINTER($2;$dataPtr)
C_POINTER($3;$dataCol)
C_LONGINT($nCol;$nFields;$tableName;$fieldName;$width;$height;$i)
C_OBJECT($0;$page;$form;$obj)
C_OBJECT($colObj)
C_TEXT($colName;$varName)
ARRAY OBJECT($arrCol;0)

If (Count parameters>1)

 $choice:=$1
 $dataPtr:=$2
 $dataCol:=$3

  Case of
    : ($choice=1)  // Selection based
      ARRAY TEXT($fldTitles;0)
      ARRAY LONGINT($fldNum;0)
      GET FIELD TITLES($dataPtr->;$fldTitles;$fldNum)

      $nFields:=Size of Array($fldTitles)
      $nCol:=$dataCol->

      If ($nCol>$nFields)
        $nCol:=$nFields
      End if

    : ($choice=2)  // Array based
      $nCol:=Size of array($dataPtr->)

    : (($choice=3) | ($choice=4))  // Collection or Entity Based Collection
      $nCol:=Size of array($dataNames->)
  End case

  OBJECT GET COORDINATES((OBJECT Get pointer(Object named;"Subform"))->;$left;$top;\
  $right;$bottom)

  $width:=$right-$left
  $height:=$bottom-$top

  // Creating the Listbox column definition
  For ($i;1;$nCol)

    Case of
      : ($choice=1)
        $colObj:=New object("objectName";$fldTitles{$i};"width";\
         Round(Num($width/$nCol);0)\;"dataSource";"["+Table name($dataPtr)+\
        "]"+$fldTitles{$i};"header"; New object("text";
        $fldTitles{$i});"textAlign";"center")

      : ($choice=2)
        $varName:=$dataPtr->{$i}


        $colObj:=New object("objectName";$varName;"width";Round(Num($width/$nCol);\
        0);"dataSource";$varName;"header";New object("text";$varName);\
        "textAlign";"center")
```

```
      : (($choice=3) | ($choice=4))
        $varName:=$dataNames->{$i}

        $colObj:=New object("objectName";$varName;"width";Round(Num($width/$nCol);0)\
        ;"dataSource";$varName;"header";New object("text";$varName);\
        "textAlign";"center")

    End case

    APPEND TO ARRAY($arrCol;$colObj)

 End for

 // Creating the Listbox definition
 Case of
   : ($choice=1)
     $obj:=New object("type";"listbox";"listboxType";"currentSelection";\
      "table";Table name($dataPtr);\
      "left";0;"top";0;"width";$width+15;"height";$height)

   : ($choice=2)
      $obj:=New object("type";"listbox";"listboxType";"array";\
      "left";0;"top";0;"width";$width+15;"height";$height)

   : (($choice=3) | ($choice=4))
      RESOLVE POINTER($dataPtr;$colName;$tableName;$fieldName)
      $obj:=New object("type";"listbox";"listboxType";"collection";\
      "dataSource";$colName;"left";0;"top";0;"width";$width+15;"height";$height)
  End case

  OB SET ARRAY($obj;"columns";$arrCol)
  $page:=New object("objects";New object("myListBox";$obj))
  $form:=New object("pages";New collection(Null;$page))

  $0:=$form
End if
```

The next sections will demonstrate the ***generateListboxFormJSONDef*** method for the different Listbox types.

Calling Selection based

To generate a Selection based Listbox, a queried table can be passed as a pointer with the number of columns specified as a pointer.

```
C_LONGINT($nCol)
ALL RECORDS([People])
$nCol:=3 // 3 Columns
$form:=generateListboxFormJSONDef (1;->[People];->$nCol)  // Selection based
```

Calling Array based

To generate an Array based Listbox, an array (arrUsed) needs to be created for the arrays that will be displayed in the columns. The example below has two arrays “PDDL” and “PDDL1” that contains data to display.

```
ARRAY TEXT(arrUsed;0)
APPEND TO ARRAY(arrUsed;"PDDL")
APPEND TO ARRAY(arrUsed;"PDDL1")
$form:= generateListboxFormJSONDef (2;->arrUsed)  // Array based
```

Calling Collection based

To generate a Collection based Listbox, a collection is passed as a pointer with a array containing the appropriate properties using “This” to specify each column to display. The collection is a collection of objects that contain “fname”, “lname”, and “age” that are columns to display.

```
C_COLLECTION(colPeople)
ARRAY TEXT(colNames;0)
CLEAR VARIABLE(colNames)
APPEND TO ARRAY(colNames;"This.fName")
APPEND TO ARRAY(colNames;"This.lName")
APPEND TO ARRAY(colNames;"This.age")

colPeople:=New collection(New object("fname";"Bob";"lname";"Smith";"age";39);New
object("fname";"Jim";"lname";"Stark";"age";38))
$form:= generateListboxFormJSONDef (3;->colPeople;->colNames)  // Collection
```

Calling Entity Selection based

To generate an Entity Selection based Listbox, an object returned from querying through ORDA is passed as a pointer with a array containing the appropriate properties using “This” to specify each column to display similar to the Collection example above.

```
C_OBJECT(eSel)
ARRAY TEXT(colFieldNames;0)
APPEND TO ARRAY(colFieldNames;"This.ID")
APPEND TO ARRAY(colFieldNames;"This.firstName")
APPEND TO ARRAY(colFieldNames;"This.lastName")

eSel:=ds.People.all()
$form:= generateListboxFormJSONDef (4;->eSel;->colFieldNames)  // Entity Collection
```

Displaying using Subform

The ***generateListboxFormJSONDef*** returns the JSON definition as an 4D object. **OBJECT SET FORM** can simply be called as the following:

```
OBJECT SET SUBFORM(*;"Subform";$form)
```

A Listbox can be displayed as the following:

ID	firstName	lastName
1	Tom	Dizzy
2	Lisa	Sanchez
3	Bob	Jackson
4	Hillary	James
5	Lucas	Jacob
6	Liam	Sanders
7	William	Masters
8	Benjamin	Button
9	Mason	Jar
10	Logan	Tay
11	Jacob	Times
12	James	Patty
13	Sally	Mae
14	Jenny	Roddy
15	Beverley	Clarance
16	Henry	Spade
17	Sam	Addys
18	Jayden	Bass
19	John	Eves

Displaying JSON definition

The JSON definition is simply using the following sample command:

```
jsonText:=JSON Stringify($form;*)
```

It can be displayed as the following:

```
{
   "pages": [
      null,
      {
         "objects": {
            "myListBox": {
               "type": "listbox",
               "listboxType": "currentSelection",
               "table": "People",
               "left": 0,
               "top": 0,
               "width": 437,
               "height": 432,
               "columns": [
                  {
                     "objectName": "ID",
                     "width": 140.6666666667,
                     "dataSource": "[People]ID",
                     "header": {
                        "text": "ID"
                     },
                     "textAlign": "center"
                  },
                  {
                     "objectName": "firstName",
                     "width": 140.6666666667,
                     "dataSource":
```


Conclusion

This Technical Note introduced a technique of displaying a Listbox dynamically without the need to develop in a form editor as well as minimizing excess code. With 4D's new feature of Dynamic forms, it is possible to prototype a Listbox very easily through a JSON definition and update very easily. This technique as an example will be very handy on generating Listbox data on the fly when a database is in runtime without disturbing normal operations. With the current features mentioned starting from v16R6, many databases can take advantage of this technique moving forward in 4D.

Reference

Dynamic Forms - http://doc.4d.com/4Dv17/4D/17/Dynamic-Forms.300-3743749.en.html