

# Using 4D v11hLists to manage your Objects

Presented by: **Luis Piñeiros - 4D, Inc.**

## INTRODUCTION

4D v11 SQL has extended the capabilities of hierarchical list items to allow storage of Object-like structures. The same hierarchical list item may have values of different types; i.e. numerical, string or boolean. In other words, you can associate a dictionary with each list item.

Using the commands SET LIST ITEM PARAMETER and GET LIST ITEM PARAMETER, you can store Objects using native 4D v11 SQL Commands.

Using the new hierarchical lists architecture in 4D v11 SQL, you can now tag lists, data, name pair values, etc. without the use of plug-ins. With 4D v11 SQL native commands you can store different types of data in Objects that can be used to manage parameter lists, store preferences, manage sessions, send a record to a different process, etc.

Objects can add flexibility and allow you to improve the way your code is designed and executed.

## OBJECTS OVERVIEW

Objects are a single entity that can store and retrieve different types of data. Although similar to BLOBs, Objects have the advantage of allowing data to be stored and retrieved in any order, as opposed to BLOBs that must be written and read in the same order.

Objects carry out their functions as an unordered dictionary. A dictionary (associative container, map, hash) is an abstract data type composed of a collection of keys and values, where each key is associated with one value. The operation of finding the value associated with the key is called indexing or lookup. Keys in Objects are called references or tags.

## BENEFITS OF USING OBJECTS

### 4D v11 SQL Native Commands

You can take advantage of Objects in your code without having to install additional plug-ins.

### Use of hierarchical lists

Objects can be created and managed using 4D v11 SQL hierarchical list theme commands.

### Storing data as named items

Objects can store and retrieve data with items that have a name.

### Modifiable data in items

The data in an item can be replaced, deleted, copied or created without recreating the Object.

### Saving Objects

Objects created using hierarchical lists can be saved in a BLOB. Using the LIST TO BLOB command you can store the list on a BLOB and save it. Using the BLOB TO LIST command you can retrieve the list.

## **USES**

### **Preferences**

Configuration and preferences information can be saved and restored using Objects.

### **Session Management**

Web based systems can store and manage sessions using Objects.

### **Replacing the use of Variables**

Objects can store and retrieve information that would normally take large numbers of variables.

### **Hierarchically structured data**

Objects can manage complex hierarchically structured information.

### **Parameter lists**

Objects can be used to store parameter lists for subroutines.

### **Sending a Record to Another Process**

Objects can be used to store complete records and send them to another process.

### **Sending a Record to Another Computer**

Objects can be used to store complete records and send them to another computer.

## **WHAT ABOUT BLOBS INSTEAD OF OBJECTS?**

4D has supported the BLOB (Binary Large Object) data type since version 6. BLOBs can be fields or variables.

Data can be stored and retrieved from BLOBs using all the BLOB theme commands. Advanced commands allow you to read and write data to a location within the BLOB; however, BLOB commands do not use tabs and pointers, in addition, arrays of pointers cannot be stored in a BLOB.

## **OBJECTS IN 4D V11 SQL**

4D v11 SQL supports Objects with the introduction of new options in its hierarchical list theme commands. Particularly, the SET LIST ITEM PARAMETER and GET LIST ITEM PARAMETER. With the use of these new commands, you can create Object tree-like structures in your 4D programming code. In addition, using BLOBs you can store and retrieve Objects. All this functionality is now possible using 4D v11 SQL native commands.

### **Hierarchical lists**

As defined in the 4D v11 SQL Language Reference, hierarchical lists are form objects that can be used to display data as lists with one or more levels that can be expanded or collapsed. It also specifies that a hierarchical lists is both a language object and a form object. The language object is referenced by a unique internal ID (listRef). The ID of type Longint, is generated by commands that are used to create lists: New list, Copy list, Load list and BLOB to list. There's one instance of the language object for the list in memory.

Each item of a hierarchical list has a reference number as well, (ItemRef). This value is only intended for your own use.

## Creating lists

There are a few different ways of creating a new list:

**New list -> listRef:** Creates a new, empty hierarchical list in memory and returns its unique list reference number.

**Copy list (list) -> listRef:** Duplicates the list whose reference number you pass in listName, and returns the list reference number of the new list.

**Load list (listName) -> listRef:** Creates a new hierarchical list whose contents are copied from the list and whose name you pass in listName. It returns the list reference number to the newly created list.

**BLOB to list(blob{;offset}) -> listRef:** Creates a new hierarchical list with the data stored within the BLOB at the byte offset (starting at zero) specified by offset and returns a list Reference number for the new list.

After you have created a hierarchical list you can:

Add items to the list, using the command APPEND TO LIST or INSERT in LIST.

Delete items from the list, using the command DELETE FROM LIST.

## COMMANDS TO IMPLEMENT OBJECT STORAGE

### Set list Item Parameter

**SET LIST ITEM PARAMETER ({\*; }list; itemRef | \*; selector; value)**

Parameter	Type		Description
*	*	→	If specified, list is an object name (string) If omitted, list is a list reference number
list	listRef   String	→	list reference number (if * omitted) or Name of list type object (if * passed)
itemRef   *	Longint   *	→	Item reference number or 0 for the last item appended to the list or * for the current list item
selector	String	→	Parameter constant
value	String Boolean Num	→	Current value of parameter

The SET LIST ITEM PARAMETER command can be used to modify the selector parameter for the itemRef item of the hierarchical list whose reference or object name is passed in the list parameter.

If you pass the first optional \* parameter, you indicate that the list parameter is an object name (string) corresponding to a representation of the list in the form. If you do not pass this parameter, you indicate that the list parameter is a hierarchical list reference (listRef). If you only use a single representation of the list or work with structural items (the second \* is omitted), you can use either syntax. Conversely, if you use several representations of the same list and the second \* is passed, the syntax based on the object name is required since each representation can have its own current item.

You can pass a reference number in itemRef. If this number does not correspond to an item in the list, the command does nothing. You can also pass 0 in itemRef to indicate the last item added to the list (using APPEND TO LIST).

Lastly, you can pass \* in itemRef: in this case, the command will be applied to the current item of the list. If several items are selected manually, the current item is the last one that was selected. If no item is selected, the command does nothing.

In selector, you can pass the Additional text constant (found in the “hierarchical lists” theme) or any custom value:

- Additional Text: This constant is used to add text to the right of the itemRef item. This additional title will always be displayed in the right part of the list, even when the user moves the horizontal scrolling cursor. When you use this constant, pass the text to be displayed in value.
- 
- Custom selector: You can also pass custom text and associate it with a value of the Text, Number or Boolean type in selector. This value will be stored with the list item and may be retrieved using the GET LIST ITEM PARAMETER command. This lets you set up any type of interface associated with hierarchical lists. For example, in a list of customer names, you can store the age of each person and only display it when the corresponding item is selected.

This is one of the key commands to implement Objects using 4D v11 SQL native commands.

## GET LIST ITEM PARAMETER

**GET LIST ITEM PARAMETER ({\*; }list; itemRef | \*; selector; value)**

Parameter	Type		Description
*	*	→	If specified, list is an object name (string)  If omitted, list is a list reference number
list	listRef   String	→	list reference number (if * omitted) or  Name of list type object (if * passed)
itemRef   *	Longint   *	→	Item reference number or  0 for the last item appended to the list or  * for the current list item
selector	String	→	Parameter constant
value	String Boolean Num	←	Current value of parameter

The GET LIST ITEM PARAMETER command is used to find out the current value of the selector parameter for the itemRef item of the hierarchical list whose reference or object name is passed in the list parameter.

If you pass the first optional \* parameter, you indicate that the list parameter is an object name (string) corresponding to a representation of the list in the form. If you do not pass this parameter, you indicate that the list parameter is a hierarchical list reference (listRef). If you only use a single representation of the list or work with structural items (the second\* is omitted), you can use either syntax. Conversely, if you use several representations of the same list and the second \* is passed, the syntax based on the object name is required since each representation can have its own current item.

*Note: If you use the @ character in the object name of the list and the form contains several lists that match this name, the GET LIST ITEM PARAMETER command will be applied to the first object whose name corresponds.*

You can pass a reference number in itemRef. If this number does not correspond to an item in the list, the command does nothing. You can also pass 0 in itemRef to indicate the last item added to the list (using APPEND TO LIST).

Lastly, you can pass \* in itemRef: in this case, the command will be applied to the current item of the list. If several items are selected manually, the current item is the last one that was selected. If no item is selected, the command does nothing.

In selector, you can pass the Additional text constant (found in the “hierarchical lists” theme) or any custom value. For more information about the selector and value parameters, please refer to the description of the SET LIST ITEM PARAMETER command.

This is another key command to implement Objects using 4D v11 SQL native commands.

## GET LIST ITEM

**GET LIST ITEM ({\*; }list; itemPos | \*; itemRef; itemText{; sublist{; expanded}})**

Parameter	Type		Description
*	*	→	If specified, list is an object name (string)  If omitted, list is a list reference number
list	listRef   String	→	list reference number (if * omitted), or  Name of list type object (if * passed)
itemPos   *	Number   *	→	Position of item in expanded list(s)  or * for the current item in the list
itemRef	Longint	←	Item reference number
itemText	String	←	Text of the list item
sublist	listRef	←	Sublist list reference number (if any)
expanded	Boolean	←	If a sublist is attached:  TRUE = sublist is currently expanded  FALSE = sublist is currently collapsed

The GET LIST ITEM command returns information about the item specified by itemPos of the list whose reference number or object name is passed in list.

If you pass the first optional \* parameter, you indicate that the list parameter is an object name (string) corresponding to a representation of the list in the form. If you do not pass this parameter, you indicate that the list parameter is a hierarchical list reference (listRef). If you only use a single representation of the list, you can use either syntax. Conversely, if you use several representations of the same list, the syntax based on the object name is required since each representation can have its own expanded/collapsed configuration and its own current item.

*Note: If you use the @ character in the name of the list object and the form contains several lists that match with this name, the GET LIST ITEM command will only apply to the first object whose name corresponds.*

The position must be expressed relatively, using the current expanded/collapsed state of the list and its sublist. You pass a position value between 1 and the value returned by Count list items. If you pass a value outside this range, GET LIST ITEM returns empty values (0, "", etc.).

After the call, you retrieve:

- The item reference number of the item in itemRef.
- The text of the item in itemText.
- If you passed the optional parameters sublist and expanded:
- sublist returns the list reference number of the sublist attached to the item. If the item has no sublist, sublist returns zero (0).
- If the item has a sublist, expanded returns TRUE if the sublist is currently expanded, and FALSE if it is collapsed.

## LIST TO BLOB

### LIST TO BLOB (list; blob{; \*})

Parameter	Type		Description
list	listRef	→	hierarchical list to store in the BLOB
blob	BLOB	→	BLOB to receive the hierarchical list
		←	New offset after reading
Function result	listRef	←	BLOB to receive the hierarchical list

The LIST TO BLOB command stores the hierarchical list list in the BLOB blob.

If you specify the \* optional parameter, the hierarchical list is appended to the BLOB and the size of the BLOB is extended accordingly. Using the \* optional parameter, you can sequentially store any number of variables or lists (see other BLOB commands) in a BLOB, as long as the BLOB fits into memory.

If you do not specify the \* optional parameter, the hierarchical list is stored at the beginning of the BLOB, overriding its previous contents; the size of the BLOB is adjusted accordingly.

Wherever the hierarchical list is stored, the size of the BLOB will be increased if necessary according to the specified location (plus up to the size of the list if necessary). Modified bytes (other than the ones you set) are reset to 0 (zero).

**Warnig:** If you use a BLOB for storing lists, you must later use the command BLOB to list for reading back the contents of the BLOB, because lists are stored in BLOBs using a 4D internal format.

After the call, if the list has been successfully stored, the OK variable is set to 1. If the operation could not be performed, the OK variable is set to 0; for example, if there was not enough memory.

## BLOB TO LIST

**BLOB to list (blob{; offset}) → listRef**

Parameter	Type		Description
blob	BLOB	→	BLOB to containing a hierarchical list
offset	Number	→	Offset within the BLOB (expressed in bytes)
*	*	→	* to append the value

The BLOB to list command creates a new hierarchical list with the data stored within the BLOB blob at the byte offset (starting at zero) specified by offset and returns a list Reference number for that new list.

The BLOB data must be consistent with the command. Typically, you will use BLOBs that you previously filled out using the command LIST TO BLOB.

If you do not specify the optional offset parameter, the list data is read starting from the beginning of the BLOB. If you deal with a BLOB in which several variables or lists have been stored, you must pass the offset parameter and, in addition, you must pass a numeric variable.

Before the call, set this numeric variable to the appropriate offset. After the call, that same numeric variable returns the offset of the next variable stored within the BLOB.

After the call, if the hierarchical list has been successfully created, the OK variable is set to 1. If the operation could not be performed, the OK variable is set to 0; for example, if there was not enough memory.

**Note regarding Platform Independence:** BLOB to list and LIST TO BLOB use a 4D internal format for handling lists stored in BLOBs. As a benefit, you do not need to worry about byte swapping between platforms when using these two commands. In other words, a BLOB created on Windows using those two commands can be reused on Macintosh and vice-versa.

## PUTTING IT ALL TOGETHER

Lets look at an example to see how to use these commands to create Objects.

In this case we want to create an Object for the Preferences selected by a User in our 4D v11 SQL application.

Instead of creating fields for the User's preferences in the Accounts we're going to save all the preferences in a BLOB field in the Accounts table. The BLOB will contain our Object. We will then retrieve the Object from the BLOB, and retrieve the preferences from the Object.

With this method we can continue to add preferences without the limitation of having to create fields or variables to store them

```

`.....

`luis piñeiros
`technical services team member
`4D, Inc.

`8.2008

`.....

`Account_Preferences

` Called by:  Account_Init

` Parameters:  none

` Returns: none

`Saves and Retrieves User's Preferences in a BLOB field/Accounts Table
`Creates new Accounts record

`.....

C_LONGINT(vObj_listRef;vObj_ItemRef)
C_LONGINT(vObj_listRef_Get;vObj_ItemRef_Get)
C_BLOB(vObj_Blob)

C_TEXT($user_dep;$user_access;$language;$system)
C_BOOLEAN($show_all)
C_LONGINT($row_color)

`Create a new list
vObj_listRef:=New list

`Reference No for the list item
vObj_ItemRef:=1

`Add a list item to the new list
`Put "preferences" on the text of the list
APPEND TO LIST(vObj_listRef;"preferences";vObj_ItemRef)

`Store a combination of preferences based on the type of user

$user_dep:="IT" `Information Technolgy
$user_access:="admin" `Administrator Access Level
$language:="US English" `Preferred language
$system:="Mac OS X 10.5" `Operating System
$show_all:=True `Show all records as a default
$row_color:=15594494 `Light blue

SET LIST ITEM PARAMETER(vObj_listRef;vObj_ItemRef | *;"user_dep";$user_dep)
SET LIST ITEM
PARAMETER(vObj_listRef;vObj_ItemRef | *;"user_access";$user_access)
SET LIST ITEM PARAMETER(vObj_listRef;vObj_ItemRef | *;"language";$language)
SET LIST ITEM PARAMETER(vObj_listRef;vObj_ItemRef | *;"system";$system)
SET LIST ITEM PARAMETER(vObj_listRef;vObj_ItemRef | *;"show_all";$show_all)
SET LIST ITEM PARAMETER(vObj_listRef;vObj_ItemRef | *;"row_color";$row_color)

```



```

`Store the list on a BLOB
LIST TO BLOB(vObj_listRef;vObj_Blob)

`Save the BLOB on the Accounts Preferences BLOB field

READ WRITE([Accounts])
CREATE RECORD([Accounts])
[Accounts]Accounts_ID:=Sequence number([Accounts])
[Accounts]User_First_Name:="Luis"
[Accounts]User_Last_Name:="Piñeiros"
[Accounts]Preferences:=vObj_Blob
SAVE RECORD([Accounts])

`Clear memory
CLEAR LIST(vObj_ItemRef;*)

`Now lets retrieve the Object

vObj_listRef_Get:=BLOB to list([Accounts]Preferences)
vObj_ItemRef_Get:=1

GET LIST ITEM
PARAMETER(vObj_listRef_Get;vObj_ItemRef_Get | *;"user_dep";$user_dep)
GET LIST ITEM
PARAMETER(vObj_listRef_Get;vObj_ItemRef_Get | *;"user_access";$user_access)
GET LIST ITEM
PARAMETER(vObj_listRef_Get;vObj_ItemRef_Get | *;"language";$language)
GET LIST ITEM PARAMETER(vObj_listRef_Get;vObj_ItemRef_Get | *;"system";$system)
GET LIST ITEM
PARAMETER(vObj_listRef_Get;vObj_ItemRef_Get | *;"show_all";$show_all)
GET LIST ITEM
PARAMETER(vObj_listRef_Get;vObj_ItemRef_Get | *;"row_color";$row_color)

`We get the values that we stored back

`$user_dep is "IT"
`$user_access is "admin"
`$language is "US English"
`$system is "Mac OS X 10.5"
`$show_all is True
`$row_color is 15594494

`.....

```

## CONCLUSION

Objects provide a powerful alternative to store and retrieve information. They also contribute to a more object-oriented style of programming. Objects allow you to store and retrieve information using tags (keys) associated with values.

## QUESTIONS?