# Introduction to the Architectural Tutorials

Architectural commands are commonly referred to as "Flat File" commands. However, Jovis provides several enhancements and capabilities which make these commands far more useful and beneficial than ordinary "Flat File" utility commands.

In addition to text, Architectural commands are used for storing and retrieving 'PICT', 'snd ', and 'Blob' (Binary Large OBjects) resources. Because these commands can be used within a "Relational" data file, they provide additional features to a Relational database. For example, you can "link" Relational records to an Architectural index of pictures, sounds, or binary objects.

Architectural commands are very fast because there is virtually no overhead in processing; you simply find a key and read its attached record. The commands provide all the "nuts and bolts" needed to build a customized database, giving you the opportunity to design a database from the ground up, or to enhance a Relational database.

Architectural data files do not accept Relational commands; however, as we have previously mentioned, Relational data files do accept Architectural commands. If you are not planning on using the Relational commands, you should create an Architectural data file.

Just as with the Relational commands, the Architectural commands require you to initialize a shell application global to be used as a global file identifier. You should use the error and warning function handlers as presented in the second Relational tutorial. (Note that 'DBInit' and 'DBOpen' do not use these handlers.) You can also use the "JovisErrorCode" global in the same way as you would for the Relational commands. The Architectural commands provide a third and unique way of returning error code information. For each command, it is possible to have an error code prefixed as an item to the return information. For example:

```
put Jovis("FindAt","myDB","Jackson") into returnResult
```

The local variable "returnResult" would contain the following:

```
"0,Jackson"
```

where the first item is the error code and all subsequent information is data returned by the commands. Here, the error code is zero, which indicates no errors occurred. This type of error code system is useful while you are creating and debugging your applications; it is rarely of any use to the end-user.

## Goal

The primary goal of these tutorials is to demonstrate the use of several of the Architectural commands by extending the capabilities of the "DASCO Database". We will provide suggestions and tips on using the these commands for various situations that may not require a Relational database. While not all of the Architectural commands will be discussed, enough information will be provided so that you can begin using any of the commands without difficulty.

The 'StartUp' and 'ShutDown' commands are used in exactly the same way for the Architectural commands as they are for the Relational commands. The 'AutoSave' property is also used in the same way for both types of commands.

# Tutorial 1

## Architectural Data Files: Creating, Opening, Closing, and Defining Structures

### Definitions:

Keys are units of information. They can point to records, but are not required to.

Keysets are a series of keys in an organized sorted order. The terms index and keyset are interchangeable.

### Before You Begin

This tutorial provides information for getting started with the Architectural commands. We begin by introducing the 'DBInit', 'DBOpen', and 'DBClose' commands. Next, we explain the 'DefineKeyset' command, as well as the 'GetKeysetName' and 'GoToKeysetName' commands. This tutorial assumes that you have completed all of the relational tutorials, and that the "DASCO Database" is completely built and functional. The error and warning function handlers that you already installed in your project or stack work exactly the same for the Architectural commands. The only exception to this rule, is the 'DBInit' and 'DBOpen' commands which use the 'JovisErrorCode' global.

If you have not already done so, you should read the "Introduction to the Architectural Tutorials". It provides several important concepts specifically related to the Architectural commands.

### Overview

The first part of this tutorial will explain and provide an example of using the 'DBInit', 'DBOpen', and 'DBClose' commands. Even though we will be creating an Architectural data file, it will not be put to use in the subsequent tutorials. Instead, we will continue to use the "DASCO Database" for demonstrating the 'DefineKeyset', 'GetKeysetName' and 'GoToKeysetName' commands. The 'DefineKeyset' command is used for creating keysets structures; the 'GetKeysetName' and 'GoToKeysetName' commands are used for switching between one keyset structure and another, thereby altering the current keyset structure. Our next tutorial will go into greater depth about "Currency", and provide several examples to

 

help you understand it. In this tutorial, we will introduce this important concept and how it functions with respect to the 'GetKeysetName' and 'GoToKeysetName' commands.

## Getting Started

### DBInit

The process for creating an Architectural data file is identical to that for creating a relational data file. You need to initialize a shell application global. You must put the word "Jovis" into a shell application global and then pass the name of the global in quotes as parameter two of the 'DBInit' command. In this way, you are establishing a global file identifier that will be used by all of the other Architectural commands. Here is a script for initializing a global and calling the 'DBInit' command:

```
1: function CreateArchitecturalFile
2:    global myArchDB, JovisErrorCode
3:    --
4:    if myArchDBis empty then
5:       --
6:       put "Jovis" into myArchDB
7:       --
8:       get Jovis("DBInit","myArchDB","","Select File:"JovisErrorCode")
9:       if item 1 of JovisErrorCode = "error" then
10:            answer JovisErrorCode
11:            put empty into myArchDB
12:            return "false"
13:         else if item 1 of JovisErrorCode = "Warning" then
14:            put empty into myArchDB
15:            return "false"
16:         end if
17:         --
18:      else
19:         return "false"
20:      end if
21:      --
22:      return "true"
23:      --
24:   end CreateArchitecturalFile
```

When this script is executed, it brings up the Macintosh 'Standard Put File Dialog', by way of the 'DBInit' command, to let you enter a name and set the location for the new data file.

 

The script has several important points that you need to be aware of:

1. Line 2 has declared two globals: 'myArchDB' and 'JovisErrorCode'. In this example, 'myArchDB' will be used as the global file identifier. This identifier tells Jovis which data file you want to work with, as well as where in memory the relevant information is located. (When you move beyond the tutorials, you can, of course, call your global file identifier anything you want.)

Remember to declare both globals and use the 'JovisErrorCode' global whenever you create or open a data file using the 'DBInit' and 'DBOpen' commands.

2. Line 4 of our 'CreateArchitecturalFile' example tests whether 'myArchDB' is empty. This is important; if this global is already in use (not empty), the next line in our script will re-initialize the global, and we will lose the original identifier. Once we are sure that 'myArchDB' is empty, we can initialize it.

3. In Line 6, we initialize 'myArchDB' by putting the name "Jovis" into it. The initialization means that when the 'DBInit' and 'DBOpen" commands "call back" to the shell application, Jovis knows which data file is being referred to. (What happens is that you are passing the name of the global to Jovis. Jovis then examines the global to make sure it contains the word "Jovis", and then sets up the link to the data file.)

NOTE: If you inadvertently change what this global contains, you lose the ability to access the data file no matter what command you try to use, including the 'DBClose' command! If this happens, your only alternative is to quit the shell application, relaunch it and then re-open the database. Treat your global file identifiers with special care and everything will be fine. Creating or opening a second data file is simply a matter of using a new and different global name. The only limitation is how much memory you have, and the size of your hard drive.

4. Line 8 is our 'DBInit' call. Notice that the global identifier is in quotes. This means that we are passing the name of the global to Jovis, NOT the contents of the global. Every call to Jovis requires that the global be in quotes.

Once the 'DBInit' command has the name of the global, it will handle the details of setting up the identifier for future use by any of the other commands. All you have to do is remember to pass the name of the global in quotes.

 

5. Lines 9 through 12 demonstrate how to handle errors that could occur. Since the 'DBInit' command has been called, we need to check for errors that may have occurred. This is done by checking the first item of the global 'JovisErrorCode'. If there is an error, item 1 of the 'JovisErrorCode' global will equal "error". The Answer dialog box will display the error message contained in the global 'JovisErrorCode' (line 10). In line 11, 'myArchDB' is reset to empty, and we exit from the script by returning "false". If there is no error, item 1 of the 'JovisErrorCode' global will equal "no error".

Similarly, in lines 13 through 16 we check for warnings. If item 1 of the 'JovisErrorCode' global equals "warning", the Answer dialog box displays the warning message, "myArchDB" is emptied, and we exit the script by returning "false". The most frequent warning message is issued when the user clicks the cancel button from the standard 'File Put Dialog'. If there is no warning, item 1 of the 'JovisErrorCode' global will equal "no error".

The various error and warning messages that may occur are listed later in this Tutorial.

This completes everything concerning our 'CreateArchitecturalFile' example. The 'DBOpen' command is done similarly. That script is provided in the "Script Examples" section of this Tutorial.

Both the 'DBInit' and 'DBOpen' commands have several other parameters, which are all optional. These are explained in the Syntax Reference section of the Jovis manual.

## DBClose

The correct way to close an Architectural data file requires the use of the 'DBClose' command. (DO NOT use this command to close a relational data file. That must only be done using the 'CloseCollection' command.) Here is our sample script for using 'DBCLose':

```
1:    on CloseDBFile
2:       global myArchDB,JovisErrorCode
3:       if myArchDBis not empty then
4:          if myArchDBis not "Jovis" then
5:             --
6:             get Jovis("DBClose","myArchDB")
7:             if item 1 of JovisErrorCode = "error" then
8:             answer JovisErrorCode
9:             exit CloseDBFile
10:            end if
11:            --
12:            if myArchDB= "Jovis" then
13:               put empty into myArchDB
14:            end if
15:            --
16:         end if
17:      end if
18:   end CloseDBFile
```

 

The above script will close your data file without shutting down Jovis.

In line 2, we have once again declared our global file identifier 'myArchDB' just as in the 'DBInit' example above.

Lines 3 and 4 check to make sure that the identifier is not empty, and that it has not already been closed. When you call the 'DBClose' command (line 6), Jovis will save any last minute changes and then close the file. It then resets the global identifier to the name "Jovis".

In line 12, we test that no errors occurred by making sure that the identifier now equals "Jovis". If so, line 13 puts empty into the global. This prepares it for reuse if necessary. Once a data file has been closed, it cannot be accessed by any of the Jovis commands, until, of course, it is reopened.

NOTE: You will want to be sure to shut down Jovis before closing your stack or quitting your shell application. See the 'ShutDown' command for further instructions. If any serious errors occur, the 'Jovis' error and warning handlers will catch them.

## Keysets

Before you can begin working with the Jovis Architectural commands, you need to understand that, in Jovis, an index is referred to as a keyset. Architectural data files consist of a series of data structures and keysets. (In computer science, an index is a type of binary tree. If this is of interest to you, consider reading some of the many computer science books that discuss data structures and various types of indexes.)

A keyset maintains a series of keys in an organized sorted order. Keys are units of information, such as a zip code or a customer ID number. In the example below, the six zip codes are in ascending order, and each has 5 digits. Each of these zip codes is a key.

```
06615
60432
60504
67701
97832
99337
```

A key can be made up of a single unit such as a zip code, or of multiple units (i.e. Part ID and Part Quantity concatenated with a delimiter like this: "T43Δ75"). In the case of multiple units, the key length is considered the combined length of all the units and delimiters.

You use the 'DefineKeyset' command to create a new keyset. If you have not done so, you should open the 'DASCO Database' file, using 'OpenCollection'. We will now begin extending our database by defining a keyset which we will name "ItemPicture". Each key in this keyset will "point" to a picture record which is a 'PICT' image of the customer's purchased item. (We will go into greater detail concerning keys in the tutorial about keys and records.)

Our routine to create the new keyset is this:

```
on CreateKeyset
   get Jovis("DefineKeyset","myDB","5","False","Text","ItemPicture")
end CreateKeyset
```

If the error and warning function handlers did not report any problems, we have created the new keyset successfully. The 'DefineKeyset' command requires 6 parameters:

Parameter three is the maximum length of each key. The maximum allowed by Jovis is 128 characters.

Parameter four should be set to "True" if you will be allowing duplicate keys, otherwise set this to "False".

Parameter five is the data type for the keys, and must be either "Text", "Date", or "Number". These are the same data types used with the relational commands

Parameter six of the 'DefineKeyset' command must be a unique name for the keyset.

As a simple test, let's ask Jovis for the name of the keyset we just created.

```
on CurKeysetName
   put Jovis("GetKeysetName","myDB") into msg
end CurKeysetName
```

The above script puts "ITEMPICTURE" into the message box. Because we just created this keyset, we can say that it is the "Current" keyset. (Our next tutorial will explain the concept of "Currency" in greater detail. At this point, we simply want to show how to get the name of the current keyset.)

If the current keyset is not "ITEMPICTURE", we use the 'GoToKeysetName' command to make it current. For example:

```
on SetKeysetName
   get Jovis("GoToKeysetName","myDB","ItemPicture")
end SetKeysetName
```

In the above script, the command 'GoToKeysetName' makes the keyset whose name is provided

 

in parameter three the current keyset.

## Script Examples

Here is the 'DBOpen' script that should be used for opening an Architectural data file:

```
on DoOpenDBFile
   global myDB,JovisErrorCode
   -- initialize by putting the name "Jovis" into a global
   if myDB = empty then
      put "Jovis" into myDB
      get Jovis("DBOpen","myDB","","Select File:","false","JovisErrorCode")
      if item 1 of JovisErrorCode = "error"  then
         answer "JovisErrorCode"
         put empty into myDB
      end if
      --> did user click 'cancel' btn?
      if item 1 of JovisErrorCode = "warning" then
         put empty into myDB
         exit DoOpenDBFile
      end if
   else
      beep
      answer "Error: Invalid global identifier."
   end if
end DoOpenDBFile
```

## Error Messages That You May Encounter

Failed to initialize cache system
Not enough parameters
Global name missing or invalid

Keyset Not Found
Invalid keyset name
Duplicate Keyset Name

## What You Should Have Learned

For those situations when you will not be using the relational commands, you should create an Architectural data file using 'DBInit'. You can use 'DBOpen' to subsequently open the file. We

also provided information and an example of using the 'DBClose' command for closing the file. Additionally, you should refer to the 'ShutDown' command for how to shut down Jovis.

You should by now understand how to create a keyset using the 'DefineKeyset' command. We briefly introduced the concept of "Currency" by introducing the 'GetKeysetName' and 'GoToKeysetName'. ("Currency" is explained in the next tutorial.)

## What to learn Next

Our next tutorial is among the most important. It explains in detail the concept of "Currency", and how it applies to the Architectural commands. Once you have a firm understanding of this concept, you will be ready to create keys and records.

# Tutorial 2

## The Concept of Currency

### Before You Begin

This tutorial explains the concept of currency. It provides examples, as well as information required to work with the Architectural commands. If you have not already done so, you should have the "DASCO Database" open. Also, you should have created the "ItemPicture" keyset as it was demonstrated to you in the previous tutorial. As always, we expect the error and warning function handlers to be installed.

### Overview

In order to explain the concept of currency we begin with a simple analogy:

If someone were to ask, where are you? You might answer "I am currently at my office located downtown in New York City". This answer is very specific, it provides three pieces of information, what city, a location within the city, and where within that location. All three pieces of information tell us something about where you are currently located. If you had simply answered, "I am in New York City". Your answer would, of course, have been correct, but it would not have provided enough information, and we would not know exactly where you are.

Using the Jovis Architectural commands requires a similar understanding of "knowing where you are". You must be located in the correct place in the database in order to search, or perform other Architectural operations. For example, you would not try to find fishing equipment if you were at a grocery store. If you wanted fishing equipment, you would obviously "go to" a fishing store. With Jovis, you use the 'GoToKeysetName' command in order to position yourself at the correct keyset for what you need to do, such as creating a key or reading a record.

In the next section, we provide examples and information on how to "be in the right place".

## Getting Started

Jovis keeps track of its current position within each Architectural database by maintaining three indicators:

- Current Keyset
- Current Key
- Current Record

### Current Keyset

From our previous tutorial we learned to set the current keyset using the 'GoToKeysetName' command. Here is our example once again:

```
on SetKeysetName
   get Jovis("GoToKeysetName","myDB","ItemPicture")
end SetKeysetName
```

Keysets have unique ID numbers. In order to know the ID of the keyset "ItemPicture," you would call the 'ListKeysetNames' command.

```
on mouseUp
   put Jovis("ListKeysetNames","myDB","0") into cd fld "Display"
end mouseUp
```

Assuming that no error occurred, the card field "Display" would now contain the names of all of the keysets and/or indexes for the entire "DASCO Database", including the relational indexes. The keysets for the Relational commands always begin with at least one asterisk. (The list will include several keysets that are for internal use.)

```
**TOP_LEVEL_KEYSET,1
**CUSTOMERS_IDXLT,2
**CUSTOMERS_ID,3
**CUSTOMERS_LAST_NAME_IDX,4
**CUSTOMERS_ZIP_IDX,5
**CUSTOMERS_CUSTOMER_ID_IDX,6
**PURCHASES_IDXLT,7
**PURCHASES_ID,8
**PURCHASES_CUSTOMER_ID_IDX,9
**PURCHASES_PURCH_DATE_IDX,10
**PURCHASES_STOCK_ITEM_REF_IDX,11
ITEMPICTURE,12
```

 

The second item in each line is the keyset ID. With this ID number you can use the 'GoToKeysetID' command to make a keyset current. For example, in this next script we can make the keyset "**CUSTOMERS_LAST_NAME_IDX" the current keyset. This keyset is one of the relational indexes, but we can also use it at the Architectural level. (Note that if the current keyset is a relational index, you cannot make changes to the index using the 'WriteKey', 'RewriteKey', or 'DelKey' commands.)

```
on GetKeysetName
   get Jovis("GoToKeysetID","myDB","4")
end GetKeysetName
```

### Current Key

Now that our "Last_Name" index is the current keyset, we can use the 'FindAt' command to find a particular key in this keyset. For example:

```
on mouseUp
   put Jovis("FindAt","myDB","Jackson") into cd fld "Display"
end mouseUp
```

The card field "Display" should now contain "Jackson".

Several different commands can be used to set the current key:

- When you execute the commands 'GoToTop', or 'DBOpen', the first key in the first created keyset becomes the current key.
- When you execute the command 'WriteKey', the key just created becomes the current key.
- When you execute one of these commands, 'FindFirst', 'FindNext', 'FindLast', 'FindPrior', 'FindAtCurRec', 'FindCurRec', or 'GoToKey,' the key requested becomes the current key.
- When you use the 'FindAt' command, the key with the specified value becomes the current key.

1. If the key you are searching for is less than the first key in the keyset, a "Not exact match" warning (error code ID 11) is issued, and the first key in the keyset is made the current key. For example:

```
put Jovis("FindAt","myDB","! Jones") into cd fld "Display"
```

Because an exclamation character has an ASCII value less than the first character of the first key in the keyset, i.e. the "A" in "Abbado", the current key becomes the first key.

2. On the other hand, if the key you are searching for is greater than the last key in the keyset, an "End of keyset" error (error code ID 53) is issued, and the last key in the keyset is made the current key, which in the "DASCO Database" is "Wesley".

```
put Jovis("FindAt","myDB","z Jones") into cd fld "Display"
```

This is because the character "z" has an ASCII value greater than the first character of the last key in the keyset, i.e. the "W" in "Wesley", so the current key becomes the last key.

3. Finally, if the key you are searching for is within the range of the keyset, but no match is found, a "Not exact match" warning is issued, and the key that is the next greater to what you are searching for is made the current key.

```
put Jovis("FindAt","myDB","N Jones") into cd fld "Display"
```

The card field "Display" will now contain "O'Neil", because there is no last name that starts with "N Jones", so the next greater key, "O'Neil", is returned.

## Current Record

The current record indicator is the third and last of our indicators. Once you have established a current keyset and key, you can then read a record from the database. By using the 'ReadText' command to read a record, you will also be making it the current record. When you go to a different key and/or keyset, the current record remains unchanged until the next 'ReadText' call is made. (The next tutorial will go into greater detail as to why the current record indicator functions in this manner.) In addition to 'ReadText', you can establish a current record with 'ReadPict', 'ReadSound', 'WriteText', 'WritePict', 'WriteSound', and 'SetCurRec'.

The 'ReadText' command returns either an Architectural or Relational record depending on whether the keyset belongs to a Relation or not. If the keyset belongs to a relation, the command returns a valid relational record which can be subsequently used by the Relational commands 'GetRecordField', 'SetRecordField', and 'UpdateRecord'. (You can NOT use 'RewriteText' to "update" the record if it is a Relational record. However, you can "link" keys in keysets which you create to a Relational record. This technique will be shown in our next few tutorials.)

Here's our example that puts all three of the currency indicators to use.

```
1:    on mouseUp
2:       put Jovis("GoToKeysetID","myDB","4")
3:       get Jovis("FindAt","myDB","Jackson")
4:       put Jovis("ReadText","myDB") into card field "Display"
5:    end mouseUp
```

 

Line 2 establishes the current keyset using the keyset ID number 4, which is the "Last_Name" relational index. In line 3, we call the 'FindAt" command in order set the current key indicator to "Jackson". Finally, we are ready to read the record that is "pointed" to by the current key. Our card field "Display" should now contain the "raw" relational record with its header information.

```
"47094252 Customers 2 Mark Jackson TRUE 1453 East View Rd. Apt. 21 Highland IL
60432 3122223513 3513 5/21/95"
```

As we mentioned earlier, the current record is NOT changed automatically each time you change the current key.

Here are the situations when the current record will change:

- When you execute the command 'ReadText', 'ReadPict', or 'ReadSound', the record pointed to by the current key becomes the current record.
- When you execute 'WriteText', 'WritePict', or 'WriteSound', the record passed as a parameter becomes the current record.
- When you execute 'SetCurRec', the record pointed to by the current key becomes the current record without actually accessing the record.

## Summary of Currency and Jovis Architectural Commands

The following table enlarges upon what is happening with the current position indicators prior to and following certain commands.

| | |
|---|---|
| DBClose | The current position indicators can be disregarded when 'DBClose' is executed. When it is done, the database file is closed and the three current position indicators for that file are cleared. |
| DBInit | Clears the currency indicators as well as the data structures, leaving you with a blank database file. |
| DBOpen | When you open a database file using the 'DBOpen' command, the current key indicator is set to the first key in the first keyset of the database. (This keyset is also the current keyset.) The key itself is not returned by 'DBOpen'. If the file is empty (i.e. there are no structures), all three indicators are undefined.<br><br>If the database is empty, the first command you need to perform is |

| | |
|---|---|
| | 'DefineKeyset' to define the characteristics of the first keyset. Upon completion of this command, this new keyset will become the current keyset. |
| CountKeys | Requires a current keyset, because it returns the number of keys for that keyset. |
| DefineKeyset | The current key and current record indicators are undefined. |
| DelKey | Requires a current key, because that is the key being deleted. The next greater key becomes the current key when this command completes. |
| DelKeyset | Requires a current keyset. The current keyset is undefined when this command completes. |
| DelRec | Requires a current key, but not a current record. Deletes whatever record is pointed to by the current key. If you have deleted the last remaining key for the keyset, the current key is undefined. |
| FindAt | Sets the current key indicator to the key equal to or greater than the key value being matched. |
| FindAtCurRec | Requires a current keyset and a current record. When finished, the current key is changed to the key that points to the current record. |
| FindCur | This command is useful for making sure that you know what the current key is.<br><br>In situations where other processing occurred after you positioned on the key, use 'FindCur' before you call a command such as 'DelKey', 'ReadText', 'ReadPict', 'ReadSound', 'WriteText', 'WritePict', 'WriteSound', 'LinkRec', 'DelRec', 'RewriteText', 'RewritePict', or 'RewriteSound'. |
| FindCurRec | Requires a current keyset and a current record. This finds the key which is pointing to the current record. When finished, the current key points to the current record. This command is similar to 'FindAtCurRec'; however, it simply steps through the keyset key by key, from the keyset's first key, trying to match a key's record to the current record indicator. For large keysets, this command is impractical. You should use the 'FindAtCurRec' command as often as possible. |
| FindFirst | Sets the current key indicator to the first key in the current keyset. |
| FindLast | Sets the current key indicator to the last key in the current keyset. |

 

| | |
|---|---|
| FindNext | Requires a current key because it instructs Jovis to set the current key indicator to the key following the current key. |
| FindPrior | Requires a current key because it instructs Jovis to set the current key indicator to the key preceding the current key. |
| FindTop | Sets the current key indicator at the first key in the first keyset in the file. |
| GoToKey | This command sets the current key to a position in the current keyset, such as the fourth key in the keyset, regardless of the key's value. |
| GoToMarker | If the marker was set on a key, that key becomes the current key. If the marker was set only on a keyset, the first key in that keyset becomes the current key. |
| GoToTop | Sets the current key indicator at the first key in the first keyset in the file. (Same as the 'FindTop' command. Used primarily for internal requirements.) |
| KeyStatus | Requires a current key. Returns information about the current key. |
| KeysetStatus | Requires a current keyset. Returns information about the current keyset. |
| LinkRec | Used for linking a current record to a current key. |
| ReadText | Reads the record pointed to by the current key. This command causes the record being read to become the current record. |
| ReadPict | Same as ReadText. |
| ReadSound | Same as ReadText. |
| ReadBlob | Same as ReadText. |
| RewriteKey | Requires a current key, which identifies the key being rewritten. |
| RewriteText | Requires a current key, which identifies the record being rewritten. |
| RewritePict | Same as RewriteText. |
| RewriteSound | Same as RewriteText. |
| RewriteBlob | Same as RewriteText. |

| | |
|---|---|
| SetCurRec | Takes the record pointed to by the current key and makes it the current record. |
| SetMarker | If you are setting a marker on a specific key, the current key must be that key. If you are setting a marker on a keyset only, the current keyset must be that keyset, and the current key is not important. |
| WriteKey | Sets the current key to the key being written. |
| WriteText | Requires a current key because the record being written will be linked to that key. Each command sets the newly-written record as the current record. |
| WritePict | Same as WriteText. |
| WriteSound | Same as WriteText. |
| WriteBlob | Same as WriteText. |

Currency indicators are irrelevant for the following commands. These do not require the indicators to be set, and will not change them in the course of their execution. (The marker commands fall into this group.)

| | |
|---|---|
| AllMarkerStatus | ChangeDBPassword |
| FreeAll | GetFileName |
| FreeMarker | GetVersion |
| GetMarkerID | ReleaseLocks |
| GetMarkerName | MarkerStatus |

## Script Examples

This is an example for "key" data entry which can be used if you want to enter a series of keys into a particular keyset.

```
on mouseUp
   put Jovis("GoToKeysetID","myDB","12")
   repeat
      ask "Enter key value:"
      if the result is "cancel" then exit repeat
      put it into newKey
      get Jovis("WriteKey","myDB",newKey,"False")
   end repeat
   put Jovis("CountKeys","myDB")
end mouseUp
```

 

## Error Messages That You May Encounter

Warnings:
Not Exact Match
Key Truncated
Key Padded

Errors:
Not Enough Params
No Key
Duplicate Key
NullKeys Not Allowed
No Record
Not a Text Record

Keyset Errors:
Keyset Not Found
Internal Keyset
Keyset Not Defined
Keyset Already Exists
End Of Keyset
Keyset Not Empty
Keyset Not Allowed
Keyset Name Empty
KeysetID Invalid
Invalid keyset name
Change Keyset Name
Dup Keyset Name
Not Current Keyset
Rel Keyset Can't Change
Empty Database

## What You Should Have Learned

This tutorial explained how to work with the concept of currency. In particular, we discussed keyset, key, and record indicators which must be set before reading or writing to a database can take place. We provided examples using the 'ListKeysetNames', 'GoToKeysetID', and 'ReadText' commands.

We also explained that you can read relational records using 'ReadText', and subsequently use relational commands with the record. However, you can not alter a relational index or "update" relational records using the Architectural commands.

## What to learn Next

The tutorial that follows will provide much of the remaining information necessary for creating keys and records with the Architectural commands. This information will allow you to store and retrieve not only text records but also multimedia objects.

# Tutorial 3

## Architectural Keys and Records

## Before You Begin

This tutorial provides the "nuts and bolts" on how to work with keys and records. We have been discussing these two topics somewhat superficially during the course of the previous two tutorials. Now it is time to discuss them directly and provide all the information required for working with them.

If you did not study the previous tutorial concerning the concept of "Currency" you should do so now. You can not use the Architectural commands without a firm grasp of this concept.

If you have not already done so, you should open the 'DASCO' database using the 'OpenCollection' command, and you should have already created the "ItemPicture" keyset at this point. We expect the error and warning function handlers to be installed.

## Overview

As we described earlier, keys are units of information, and a keyset contains a series of keys in an organized sorted order. Each key contains information that makes it possible to retrieve a record which is meaningful to that key. It is not a requirement that keys "point" to records; it is perfectly acceptable to have keys without records.

## Getting Started

### Keys

Keys of the "Text" data type are ASCII strings from 1 to 255 characters. All keys within one defined keyset are of the same fixed length and type. The characters you use as keys for the "Text" data type are entirely your choice. They can contain any characters: alphabetic, numeric, or punctuation, even delimiters such as commas or carriage returns. Keys are case-sensitive.

An incorrect search result can occur when the case of the actual key and the intended key do not match. We recommend that you convert text keys to either lower or upper case. (See the utility commands for assistance.)

Keys defined as date or numeric are kept in correct order, without special manipulation.

The three date formats, short, abbreviated, and long can be kept in a date key. (Note that time is not supported in a date key.)

Any number format supported by the shell application can be kept in a number key. You can also use the formatting characters dollar sign, comma, decimal, hyphen, and left and right parenthesis; the number will still be properly recognized:

For Example:

```
$999,999.99
9.9999999999999999
($999,999.99)
-999,999.9999999999
```

Numbers with up to 16 significant decimal places will be recognized by Jovis. You must keep in mind that computations in your shell application may round the result to only 6 significant decimal places.

Keys, although generally used with records, are independent of records. You may, for example, create keys that you intend to assign at some later point to a record.

You can also have a file that contains keys and no records. For example, if you have a file with very short data (such as zip codes and state equivalents), you could store it all in keys and not even have records. Each time you read a key, you have the data without further lookups.

More than one key can point to the same record. There is no limit to the number of keys that can point to a record.

It is often useful to have one record pointed to by keys from different keysets. In a rolodex application, for example, you can index each record by name, serial number, date of birth, and phone. Each of these keys resides in a different keyset, and yet each one points to the same record.

You can also have several keys in the same keyset pointing to one record, as “aliases” to each other. For example, in a 'PICT' database, the keys “dog”, “fido”, and “pet” might all point to the same picture of a dog.

 

You might decide that no duplicates should occur in a keyset. For example, a keyset containing employee ID numbers might require that each key be unique. In such a keyset, any attempt to write a duplicate key would return an error.

On the other hand, if duplicate keys are allowed, several duplicate keys could point to several different records. For instance, several records might be referenced by the same zip code key.

Each key can refer to only one record at a time. A key can be unlinked and relinked to a different record, but it can only be linked to one record at a time.

Deleting a key does not delete the record it points to. It only cuts the link between key and record. Special care should be exercised to prevent a "dangling record" which no longer has any keys pointing to it. This could happen if you delete all the keys that point to a given record.

### Designing the contents of keys

A key may be anything, from a sequence of numbers to a description of the contents of a record, to a portion of some data field, such as a name or a zip code. It is up to you to determine the contents of a key, and how it is related to the record to which it points.

Furthermore, you can include some specific information about the contents of the record in the key itself. This allows you to "screen" the records before actually reading them.

For example, a key written “PizzaPlus,32,1,0,3” might indicate a client named Pizza Plus who is assigned to sales area 32 and is COD (1), no discount (0), and is a restaurant (3).

### Records

An Architectural record can be one of four types: 'Text', 'PICT' resource, 'snd ' resource, or 'Blob'. The 'Blob', or Binary Large OBject, is any resource, or file that you would like it to be.

You cannot mix record types in a single record. For example, 'PICT' and 'Text' information cannot exist in the same record. Data files can have all four record types, or any combination of them.

A record is considered a single entity; you cannot write, read, or rewrite partial records. In the case of sound and PICT records, you must have enough memory to load the entire record into memory. These record types can be quite large, so you should plan ahead when working with them.

There are no fixed record length requirements. All records are treated as variable length. You may rewrite a record no matter what size it has become. Jovis will automatically reuse free space created through deleting or rewriting records.

## "Linking" records with keys

There are two ways to have a key "point" to a record. We will show you the most direct way first, and then demonstrate the other later.

We begin by first creating a new keyset called "Sample". [When we have finished, we will delete this keyset as part of our demonstration.]:

```
get Jovis("DefineKeyset","myDB","8","true","Text","Sample")
```

Notice that we set the key length to 8 characters. If we create a key that is fewer characters, we will get a warning message stating that the key was padded to the full 8 characters. This is perfectly normal and acceptable; the warning message is simply to let you know what is going on. You will also get a warning message when a key has more characters than the defined character length. In this case, the extra characters will be automatically truncated.

Now we are ready to create a record and write it to the database.

```
1: on CreateSaveRecord1
2:    put "A Children's toy for ages 7 to 10." into aRec
3:    get Jovis("GoToKeysetName","myDB","Sample")
4:    get Jovis("WriteText","myDB",aRec)
5:    get Jovis("WriteKey","myDB","WIGGLER","UseCurRec")
6: end CreateSaveRecord1
```

The script above begins by putting some descriptive information into the local variable called 'aRec'. This information will be our database record. Next, we set our newly created keyset, "Sample," as the current keyset. In this situation, this call is not actually necessary, because the most recently created keyset is also the current keyset. However, we include it here to emphasize how important the current keyset indicator is.

Anytime you write a record to a Jovis database, it becomes the current record. In line 6, we use the 'WriteText' command to save the contents of the variable 'aRec' as our record. The next line calls 'WriteKey' with the key "WIGGLER", which is a children's toy. The fourth parameter of this command uses the literal "UseCurRec". This indicates that this key will point to the current record. When we call 'FindAt' for the key "Wiggler" and retrieve it's record using 'ReadText,' it will be returned. Here is the script that does just that:

 

```
on RetrieveRecord
   get Jovis("GoToKeysetName","myDB","Sample")
   get Jovis("FindAt","myDB","WIGGLER")
   put Jovis("ReadText","myDB") into cd fld "Display"
end RetrieveRecord
```

We mentioned above that there is a second way of "linking" a record to a key. Here is an example of this alternate solution:

```
1: on CreateSaveRecord2
2:    put "A Children's toy for ages 7 to 10." into aRec
3:    get Jovis("WriteKey","myDB","WIGGLER")
4:    get Jovis("WriteText","myDB",aRec)
5:    get Jovis("LinkRec","myDB")
6: end CreateSaveRecord2
```

This procedure has two important differences from the first solution. First, you should notice that in line 6 we are using the 'LinkRec' command for "linking" the record to the key. Second, with 'LinkRec' it makes no difference whether you call 'WriteKey' or 'WriteText' first. Both commands set up their currency indicators, and 'LinkRec' does the linking.

### Linking multiple keys to one record

Now we are ready to demonstrate how multiple keys can "point" to the same record. In the following script, we call 'GoToKeysetName' to make sure that we are currently in the correct keyset. In line 4, we create a new record, which also makes it the current record. Each time we call 'WriteKey' (lines 5 through 7) it automatically "links" to the current record because we are using the fourth parameter flag "UseCurRec".

```
1: on CreateSaveRecord3
2:    get Jovis("GoToKeysetName","myDB","Sample")
3:    put "Slider: Children's toy for ages 7 to 10." into aRec
4:    get Jovis("WriteText","myDB",aRec)
5:    get Jovis("WriteKey","myDB","SLIDER","UseCurRec")
6:    get Jovis("WriteKey","myDB","7-10","UseCurRec")
7:    get Jovis("WriteKey","myDB","TOYS","UseCurRec")
8: end CreateSaveRecord3
```

As you would assume, we now have three keys, all "pointing" to the same record. By using 'FindAt' you can make any of these keys the current key, and then retrieve the record using 'ReadText'. Here is an example:

```
on RetrieveRecord
      get Jovis("FindAt","myDB","TOYS")
      put Jovis("ReadText","myDB") into cd fld "Display"
end RetrieveRecord
```

The card field "Display" should now contain the record:

```
"Slider: Children's toy for ages 7 to 10."
```

If you are using multiple keysets, you "go to" a particular keyset and create the new key. The current record will not change, so you can still use the fourth parameter flag "UseCurRec" when you call "WriteKey", or the 'LinkRec' command as an alternative. There is no limit to the number of keys you can create and link to a current record.

## Deleting keys, records, and keysets

The primary purpose for deleting keys and records is to reclaim disk space. Just because keys and records are no longer being used does not mean that you must, or even should, delete them. In general, deletion provides few benefits except for occasionally freeing disk space for reuse.

The order in which you delete keys, records, and keysets is very important. If you want to delete a record, you must use one of the "Find" commands to locate the key that "points" to the record, thereby making it the current key. Next, you need to make the record you want to delete the current record. You can do this by using the 'SetCurRec' command, which simply makes the record pointed by the current key the current record; it will not read the record back to you.

You must be very careful to delete the record before you delete its key. Deleting the last, or only, key that points to a record means the record can NOT be recovered. If there are multiple keys, you can delete the record at any time. However, once you have deleted it, you must NOT access the record later on. You use the 'DelRec' command to delete the record. Once completed, the space previously used by the record becomes available for reuse.

After the record has been deleted, you can use the 'DelKey' command to delete the key that was pointing to the record. This command does not check to see whether a record is linked to it. It simply deletes the key. Once deleted, the next key becomes the current key. If there are no other keys, the error message "end of keyset" is sent.

If all the keys in a keyset have been deleted, you can delete a keyset using the 'DelKeyset' command. This removes the keyset from memory, and sets any disk space used by the keyset as available for reuse.

We have created a keyset called "Sample", in which we created a total of four keys and two records. The following script demonstrates the correct process of deleting the records, keys, and finally the keyset.

 

```
   on DeleteKeysRecords
      --
      get Jovis("GoToKeysetName","myDB","Sample")
      get Jovis("FindAt","myDB","WIGGLER")
      get Jovis("SetCurRec","myDB")
      get Jovis("DelRec","myDB")
      get Jovis("DelKey","myDB")
      --
      get Jovis("FindAt","myDB","TOYS")
      get Jovis("SetCurRec","myDB")
      get Jovis("DelRec","myDB")
      get Jovis("DelKey","myDB")
      --
      get Jovis("FindAt","myDB","7-10")
      get Jovis("DelKey","myDB")
      --
      get Jovis("FindAt","myDB","SLIDER")
      get Jovis("DelKey","myDB")
      --
      get Jovis("DelKeyset","myDB")
      --
   end DeleteKeysRecords
```

## Script Examples

The following examples should be of some assistance in putting the information we have presented into practice.

```
on CreateSmallDB
   CreateKeysets
   CreateRecords
end CreateSmallDB

on FindARec
   GetRecordByName
end FindARec

on CreateKeysets
   global JovisErrorCode
   get Jovis("DefineKeyset","myDB","7","true","Text","Name")
   get Jovis("DefineKeyset","myDB","10","true","Number","ID")
end CreateKeysets
```

```
on CreateRecords
   -- Record, Key: Name, Key: ID
   LoadRecord "John SmithΔPO Box 27ΔHartford, CT  06615", "JOH-SMI", "4432"
   LoadRecord "Alice BrownΔRR 1ΔTimbuktu, OR  97832", "ALI-BRO", "3513"
   LoadRecord "William AndersonΔ11 West Cedar StreetΔAurora, IL  60504", & ¬
                "WIL-AND","8705"
end CreateRecords

on LoadRecord aRec,aName,aID
   global JovisErrorCode
   get Jovis("GoToKeysetName","myDB","Name")
   get Jovis("WriteText","myDB",aRec)
   get Jovis("WriteKey","myDB",aName,"UseCurRec")
   get Jovis("GoToKeysetName","myDB","ID")
   get Jovis("WriteKey","myDB",aID,"UseCurRec")
end LoadRecord

on GetRecordByName
   global JovisErrorCode
   Ask "Enter first name:" with "Must be at least 3 characters"
   if the result = "cancel" then exit GetRecordByName
   put it into fName
   Ask "Enter last name:" with "Must be at 3 least characters"
   if the result is "cancel" then exit GetRecordByName
   put it into lName
   put char 1 to 3 of fName & "-" & char 1 to 3 of lName into aName
   put Jovis("UpperCase",aName) into aName
   get Jovis("SetProperty","myDB","WarningMsg","") --> disable warning msg
   get Jovis("GoToKeysetName","myDB","Name")
   get Jovis("FindAt","myDB",aName)
   put JovisErrorCode into temp
   --> Now enable warning msg
   get Jovis("SetProperty","myDB","WarningMsg","JovisWarningMsg")
   if item 1 of temp is not "0" then
      answer "Name not found."
      exit GetRecordByName
   end if
   put Jovis("ReadText","myDB") into cd fld "Display"
end GetRecordByName

on GetRecordByID
   global JovisErrorCode
   Ask "Enter beginning of a customer's ID number:"
   if the result is "cancel" then exit GetRecordByID
   put it into aID
   get Jovis("SetProperty","myDB","WarningMsg","") -- disable warning msg
   get Jovis("GoToKeysetName","myDB","ID")
   get Jovis("FindAt","myDB",aID)
   put JovisErrorCode into temp
   get Jovis("SetProperty","myDB","WarningMsg","JovisWarningMsg") -- enable
warning msg
   if item 1 of temp is not "0" then
```

 

```
      answer "Name not found."
      exit GetRecordByID
   end if
   put Jovis("ReadText","myDB") into cd fld "Display"
end GetRecordByID

on DeleteSimpleDB
   DeleteRecordsByName
   DeleteKeysByID
   get Jovis("GoToKeysetName","myDB","Name")
   get Jovis("DelKeyset","myDB","Name")
   get Jovis("GoToKeysetName","myDB","ID")
   get Jovis("DelKeyset","myDB","ID")
   put Jovis("ListKeysetNames","myDB","0") into cd fld "Display"
end DeleteSimpleDB

on DeleteRecordsByName
   put "JOH-SMI,ALI-BRO,WIL-AND" into NameList
   get Jovis("GoToKeysetName","myDB","Name")
   repeat with x = 1 to number of items of NameList
      get Jovis("FindAt","myDB",item x of NameList)
      get Jovis("SetCurRec","myDB")
      get Jovis("DelRec","myDB")
      get Jovis("DelKey","myDB")
   end repeat
end DeleteRecordsByName

on DeleteKeysByID
   put "4432,3513,8705" into idList
   get Jovis("GoToKeysetName","myDB","ID")
   repeat with x = 1 to number of items of idList
      get Jovis("FindAt","myDB",item x of idList)
      get Jovis("DelKey","myDB")
   end repeat
end DeleteKeysByID
```

## Error Messages That You May Encounter

Warnings:
Not Exact Match
Key Truncated
Key Padded

Errors:
Not Enough Params
No Key
Duplicate Key

NullKeys Not Allowed
No Record
Not a Text Record

Keyset Errors:
Keyset Not Found
Internal Keyset
Keyset Not Defined
Keyset Already Exists
End Of Keyset
Keyset Not Empty
Keyset Not Allowed
Keyset Name Empty
KeysetID Invalid
Invalid keyset name
Change Keyset Name
Dup Keyset Name
Not Current keyset
Rel keyset cant change
Empty Database

## What You Should Have Learned

We covered a great deal of material in this tutorial. You should feel comfortable with creating keysets, keys, and records and this point. Additionally, we discussed how to delete records and the importance of not deleting the last key pointing to a record until the record had been deleted. We also provided an extensive example using the material presented in this tutorial.

## What to learn Next

In the next tutorial we demonstrate how to use the relational and Architectural commands in the same database. In particular, we will be adding pictures to our "DASCO Database". This will provide an opportunity to explain how to store and retrieve multimedia objects.

# Tutorial 4

## Using Architectural and Relational Commands Together

### Before You Begin

This tutorial assumes that you have a good grasp of everything that has been presented in the previous tutorials. If this tutorial were used in a Jovis database classroom course, it would be part of the advanced topics section. We start by using the "ItemPicture" keyset with the "DASCO Database". We will also demonstrate how multiple keys in an Architectural keyset can point to a "DASCO" relational record. This is referred to as a many-to-one relationship.

### Overview

Until now, all of our tutorials have dealt with either the Relational, or the Architectural commands. In this tutorial, we show the capabilities that become available by using both command sets in conjunction with each other.

In our first example, we will take three pict's from the Macintosh scrapbook and save them as 'PICT' records. We will create keys for each 'PICT' record from information taken from the "Purchase" relation of our "DASCO Database".

In our second example, we show how it is possible for our retail store to access customer records with multiple phone numbers provided by the customer. This will involve creating a new keyset for alternate customer phone numbers. In this way, it is possible to locate a customer's record using any phone number they provided at the time their account was created.

### Getting Started

#### Example One:

Let's start by implementing the "ItemPicture" keyset. This keyset will consist of purchase item keys, such as item "T-253". This is the 'Teddy Bear' purchased by "Judy Ritter", Customer_ID #1959. Each key in our keyset will hold the location of a picture record which is a 'PICT' image

of the purchased item. In this example it is of a "Teddy Bear".

From your Macintosh "Scrapbook", located under the Apple Menu, find the "Teddy Bear" 'PICT' and copy it to the clipboard.

Here is the script to save it to the "DASCO Database," as well as create the key for it.

```
1:    on PictToDB
2:       global thePict
3:       if thePict is not empty then
4:          answer "'Pict' Global is not empty."
5:          exit PictToDB
6:       end if
7:       put "Jovis" into thePict
8:       get Jovis("ClipToPict","thePict")
9:       --
10:      ask "Enter purchase item number:" with "T-253"
12:      if the result = "Cancel" then exit mouseUp
13:      else put char 1 to 5 of it into theKey
14:      put Jovis("GoToKeysetName","myDB","ITEMPICTURE")
15:      get Jovis("WritePict","myDB","thePict")
16:      get Jovis("WriteKey","myDB",theKey,"UseCurRec")
17:      get Jovis("ClearPict","thePict")
18:      put empty into thePict
19:   end PictToDB
```

Jovis uses a global variable for maintaining information about the 'Pict'. You must first "initialize" it by putting the name "Jovis" into it, as in line 7 above. This is the same process used when creating or opening a database file. The next step is to "load" the global variable with the 'Pict' that was saved to the clipboard. The command "ClipToPict" does this in line 8.

Now that we have a valid 'Pict' global, we can save it to the database and create the key for it. This is done in lines 14 through 16. Once everything is saved, we can clear the memory used for the 'Pict' by calling the 'ClearPict' command and then clearing the actual shell application global, as in line 18.

We suggest you repeat this process for customer "Dorthy Quinn", purchase item number "H-934", which is a party hat. There is a "Party Hat" image in the scrapbook that you can use.

 

Now that we have saved our 'Pict' to the database, we need a way to retrieve and display it. Below is the script to do this:

```
1:    on DisplayPicture
2:       global thePict,JovisErrorCode
3:       ask "Enter purchase item number:" with "T-253"
4:       if the result = "Cancel" then exit mouseUp
5:       else put char 1 to 5 of it into aKey
6:       if thePict is not empty then exit DisplayPicture
7:       put "Jovis" into thePict
8:       get Jovis("GoToKeysetName","myDB","ITEMPICTURE")
9:       get Jovis("FindAt","myDB",aKey)
10:      if item 1 of JovisErrorCode is "error" then
11:         put empty into thePict -- reset global
12:         exit DisplayPicture -- nothing found
13:      end if
14:      get Jovis("ReadPict","myDB","thePict")
15:      get Jovis("PictToClip","thePict")
16:      get Jovis("ClearPict","thePict")
17:      put empty into thePict
18:      -- first dispose of any existing picture window
19:      if there is a window "Purchase Item" then close window "Purchase Item"
         -- Now use HyperCard's Picture XCMD
20:      Picture "Purchase Item","clipboard","rect","false","0","true"
21:      if there is a window "Purchase Item" then
22:         set rect of window "Purchase Item" to "325,85,489,279"
23:         show window "Purchase Item"
24:      end if
25:   end DisplayPicture
```

In line 9, we use the 'FindAt' command to locate the purchase item key. Then, in line 14, we load the 'Pict' global with the 'Pict' image using the 'ReadPict' command. Next we save it to the clipboard so that we can us HyperCard's 'Picture' XCMD to display it. (You may have other utilities to display 'PICT's, we found that HyperCard's Picture XCMD is quite good for our needs. If all you need is a modal dialog to display the image, or want to just draw it to the screen, you can use the Jovis 'DisplayPict' command.) Once the picture is saved to the clipboard, we must free the memory used by it. This is done in line 16.

There are also commands for handling 'snd ' and 'Blob's. They are used in the same way as the 'Pict' commands. Each requires a dedicated global to be passed in quotes as the global name in a specified parameter. Here is an example of using the 'Blob' commands:

```
on ReadBlob
   global BlobGlobal
   put "Jovis" into BlobGlobal
   get Jovis("ReadBlob","BlobGlobal")
   get Jovis("BlobToClip","BlobGlobal")
   get Jovis("ClearBlob","BlobGlobal")
end ReadBlob
```

Example Two:

In our customer database, we used the last four digits of the customer's phone number as their customer ID. If a customer were to call our store, a store representative might ask for the last four digits of the customer's phone number. But what happens if the customer doesn't remember which phone number they originally gave? By creating an alternate customer ID keyset, we can save as many customer IDs, i.e. other phone numbers, as they want to supply. For example, customer "John Smith" has customer ID "4432"; his phone number is (860) 881-4432. When his account was created, he gave two phone numbers. The first one is saved as the main account ID, "4432"; the other phone number, (860) 881-4433, is saved as his alternate customer ID, "4433". Here are the scripts required for all of this to work:

```
   on CreateKeyset
      get Jovis("DefineKeyset","myDB","4","False","Number","AltCustomerID")
   end CreateKeyset
```

```
1: on addAltCustomerID
2     global JovisErrorCode
3:    get Jovis("GoToKeysetName","myDB","**CUSTOMERS_CUSTOMER_ID_IDX")
4:    get Jovis("FindAt","myDB","4432")
5:    if item 1 of JovisErrorCode is "error" then
6:       answer "Main Customer ID not found!"
7:       exit addAltCustomerID
8:    end if
9:    get Jovis("SetCurRec","myDB")
10:   get Jovis("GoToKeysetName","myDB","AltCustomerID")
11:   get Jovis("WriteKey","myDB","4433")
12:   end addAltCustomerID
```

In line 3 we go to the relational index "Customer_ID" using the 'GoToKeysetName' command. (Note that Jovis prefixes Relational index names with one or more asterisks, the relation's name, and also appends the name with the characters "_IDX". These two additions are required in order to find the correct relational keyset using the 'GoToKeysetName' command.)

In line 4, we find the main Customer ID "4432" in the index, and in line 9 we make the customer's record the current record using the 'SetCurRec' command. Next, we switch to the "AltCustomerID" keyset and write the new key, which then "binds" it to the current record .

As you can see the process is very simple. Now any "DASCO" customer can have multiple phone numbers as Customer_ID's. Note that the alternate customer numbers should be saved with the relational record as well. If at some future time you need to delete the customer's record, you will then also be able to delete the alternate customer IDs.

 

## Script Examples

Here is an example that deletes a 'Pict' record and the key that points to it:

```
on DeletePicture
   ask "Enter purchase item number:" with "T-253"
   if the result = "Cancel" then exit mouseUp
   else put char 1 to 5 of it into aKey
   get Jovis("GoToKeysetName","myDB","ITEMPICTURE")
   get Jovis("FindAt","myDB",aKey)
   if item 1 of JovisErrorCode is "error" then
      put empty into thePict -- reset global
      exit DisplayPicture -- nothing found
   end if
   get Jovis("SetCurRec","myDB")
   get Jovis("DelRec","myDB")
   get Jovis("DelKey","myDB")
end DeletePicture
```

## Error Messages That You May Encounter

No valid global
Relational keysets cannot be changed
Keyset not found
Null Keys not Allowed
Cannot write or rewrite relational record
Not enough memory

Not PICT File
Not Pict Global
Not a Picture

Not Sound Global
Not a Sound Handle
Not a Sound Record

Not a Blob Record
Blob Not Selected
Not Blob Global
Invalid parameter information

## What You Should Have Learned

This tutorial presented two examples using the Architectural and Relation commands sets in conjunction with each other. At the same time, we demonstrated how to use the "Picture" commands to save and retrieve 'Pict" images. The "Sound" and "Blob" commands are used the in the same manner.

## What to learn Next

If you have not already done so, you should study the scripts used in the Jovis Demo. The demo is an "enhanced" version of our "DASCO Database". There are several scripts worth reusing or modifying for your own needs. The demo also provides examples of how to delete two "Linked" records from two different relations. Finally, it provides a very good basis for developing more complex database requirements.