4th Dimension®

Addendum Version 6.0.2
for Windows and Mac OS

4th Dimension
by
Laurent Ribardière
Adapted by Bernard Gallet

4th Dimension
Addendum 6.0.2 for Windows® and Mac™ OS

IMPORTANT LICENSE INFORMATION

Use of this Software is subject to the ACI Product Line License Agreement, which is provided in electronic form with the Software. Please read the ACI Product Line License Agreement carefully before completely installing or using the Software.

Contents

1 Addendum 4D 6.0.2

Addendum 4D 6.0.2

version 6.0.2

This document describes the changes that have been made to the 4D language since version 6.0

Picture Library

Starting with version 6.0.2, you can access the Picture Library of a database programmatically. The following commands have been added:

- PICTURE LIBRARY LIST (6.0.2)
- GET PICTURE FROM LIBRARY (6.0.2)
- SET PICTURE TO LIBRARY (6.0.2)
- REMOVE PICTURE FROM LIBRARY (6.0.2)

These new commands allow you to get the list of the pictures currently stored in the Picture Library, as well as to change, add and delete pictures.

Note: These new commands are described in this addendum as well as the *4th Dimension Language Reference* manual delivered with version 6.0.2.

Process (Communications)

The command SET PROCESS VARIABLE does NOT write an array as a whole from one process to another. In order to provide this capability, the new command VARIABLE TO VARIABLE has been added to the language. For more information, see the section VARIABLE TO VARIABLE (6.0.2).

Note: This new command is described in this Addendum as well as in the *4th Dimension Language Reference* manual delivered with version 6.0.2. The description of the command SET PROCESS VARIABLE has been updated accordingly.

User and Groups

The command GET USER PROPERTIES no longer returns the encrypted password in the password parameter. For more information, see the section GET USER PROPERTIES (6.0.2).

The command SET USER PROPERITES now accepts the * symbol as value for the password parameter. For more information, see the section SET USER PROPERTIES (6.0.2).

The command Validate password has been added. For more information, see the section Validate password (6.0.2).

Note: Changes and additions to the Users and Groups theme are described in this Addendum, but are not described in the *4th Dimension Language Reference* manual delivered with version 6.0.2.

Web Server

• 4th Dimension sends a text parameter to any 4D method called via a URL. Regarding this text parameter:

- Although you do not use this parameter, you must explicitly declare it with the line C_TEXT($1), otherwise runtime errors will occur when using the Web to access a database that runs in compiled mode.
- This parameter returns the extra data placed at the end of the URL, and can be used as a placeholder for passing values from the HTML environment to the 4D environment.

For more information, see the section The Text Parameter Passed to 4D Methods Called via URLs (6.0.2).

Note: This text parameter is described in this Addendum as well as in the *4th Dimension Language Reference* manual delivered with version 6.0.2.

• The command CHANGE WEB LICENSE has been added. This command displays the Web License dialog box in the User or Custom menus environment. For more information, see the section CHANGE WEB LICENSE (6.0.2).

Note: The command CHANGE WEB LICENSE is described in this Addendum, but is not described in the *4th Dimension Language Reference* manual delivered with version 6.0.2.

2 Picture Library (6.0.2)

PICTURE LIBRARY LIST — Pictures

version 6.0.2

PICTURE LIBRARY LIST (picRefs; picNames)

Parameter	Type		Description
picRefs	Numeric Array	←	Reference numbers of the Picture Library graphics
picNames	String Array	←	Names of the Picture Library graphics

Description

The command PICTURE LIBRARY LIST returns the reference numbers and names of the pictures currently stored in the Picture Library of the database.

After the call, you retrieve the reference numbers in the array picRefs and the names in the array picNames. The two arrays are synchronized: the nth element of picRefs is the reference number of the Picture Library graphic whose name is returned in the nth element of picNames.

The array picRefs can be a Real, Long Integer or Integer array. In interpreted mode, if the array is not declared prior to the call to PICTURE LIBRARY LIST, a Long Integer array is created by default.

The array picNames can be a String or Text array. In interpreted mode, if the array is not declared prior to the call PICTURE LIBRARY LIST, a Text array is created by default.

The maximum length of a Picture Library graphic name is 31 characters. If you use a String array as picNames, declare it with a large enough fixed length to avoid having a truncated name returned.

If there are no pictures in the Picture Library, both arrays are returned empty.

To obtain the number of pictures currently stored in the Picture Library, use the Size of array command to get the size of one of the two arrays.

Examples

1. The following code returns the catalog of the Picture Library in the arrays alPicRef and asPicName:

⇒ **PICTURE LIBRARY LIST**(alPicRef;asPicName)

2. The following example tests whether or not the Picture Library is empty:

```
PICTURE LIBRARY LIST(alPicRef;asPicName)
If (Size of array(alPicRef)=0)
   ALERT("The Picture Library is empty.")
Else
   ALERT("The Picture Library contains "+String(Size of array(alPicRef))+" pictures.")
End if
```

3. The following example exports the Picture Library to a document on disk:

```
⇒  PICTURE LIBRARY LIST($alPicRef;$asPicName)
   $vlNbPictures:=Size of array($alPicRef)
   If ($vlNbPictures>0)
      SET CHANNEL(12;"")
      If (OK=1)
         $vsTag:="4DV6PICTURELIBRARYEXPORT"
         SEND VARIABLE($vsTag)
         SEND VARIABLE($vlNbPictures)
         gError:=0
         For($vlPicture;1;$vlNbPictures)
            $vlPicRef:=$alPicRef{$vlPicture}
            $vsPicName:=$asPicName{$vlPicture}
⇒           GET PICTURE FROM LIBRARY(alPicRef{$vlPicture};$vgPicture)
            If (OK=1)
               SEND VARIABLE($vlPicRef)
               SEND VARIABLE($vsPicName)
               SEND VARIABLE($vgPicture)
            Else
               $vlPicture:=$vlNbPictures+1
               gError:=-108
            End if
         End for
         SET CHANNEL(11)
         If (gError#0)
            ALERT("The Picture Library could not be exported, retry with more memory.")
            DELETE DOCUMENT (Document)
         End if
      End if
   Else
      ALERT("The Picture Library is empty.")
   End if
```

See Also

GET PICTURE FROM LIBRARY, REMOVE PICTURE FROM LIBRARY, SET PICTURE TO LIBRARY.

GET PICTURE FROM LIBRARY

Pictures

version 6.0.2

GET PICTURE FROM LIBRARY (picRef; picture)

Parameter	Type		Description
picRef	Number	→	Reference number of Picture Library graphic
picture	Picture Variable	←	Picture from the Picture Library

Description

The GET PICTURE FROM LIBRARY command returns in the picture parameter the Picture Library graphic whose reference number is passed in picRef.

If there is no picture with that reference number, GET PICTURE FROM LIBRARY leaves picture unchanged.

Examples

1. The following example returns in vgMyPicture the picture whose reference number is stored in the local variable $vlPicRef:

⇒ **GET PICTURE FROM LIBRARY**($vlPicRef;vgMyPicture)

2. See the third example for the command PICTURE LIBRARY LIST.

See Also

PICTURE LIBRARY LIST, REMOVE PICTURE FROM LIBRARY, SET PICTURE TO LIBRARY.

System Variables and Sets

If the Picture Library exists, the OK variable is set to 1. Otherwise, OK is set to zero.

Error Handling

If there is not enough memory to return the picture, an error -108 is generated. You can catch this error using an error-handling method.

SET PICTURE TO LIBRARY

Pictures

version 6.0.2

SET PICTURE TO LIBRARY (picture; picRef; picName)

Parameter	Type		Description
picture	Picture	→	New picture
picRef	Number	→	Reference number of Picture Library graphic
picName	String	→	New name of the picture

Description

The command SET PICTURE TO LIBRARY creates a new picture or replaces a picture in the Picture Library.

Before the call, you pass:
• the picture reference number in picRef (range 1...32767)
• the picture itself in picture.
• the name of the picture in picName (maximum length: 31 characters).

If there is an existing Picture Library graphic with the same reference number, the picture contents are replaced and the picture is renamed according to the values passed in picture and picName.

If there is no Picture Library graphic with the reference number passed in picRef, a new picture is added to the Picture Library.

4D Server: SET PICTURE TO LIBRARY cannot be used from within a method executed on the server machine (stored procedure or trigger). If you call SET PICTURE TO LIBRARY on a server machine, nothing happens—the call is ignored.

Warning: Design objects (hierarchical list items, menu items, etc.) may refer to Picture Library graphics. Use caution when modifying a Picture Library graphic programmatically.

Note: If you pass an empty picture in picture or a negative or null value in picRef, the command does nothing.

Examples

1. No matter what the current contents of the Picture Library, the following example adds a new picture to the Picture Library by first looking for a unique picture reference number:

```
⇒   PICTURE LIBRARY LIST($alPicRef;$asPicNames)
    Repeat
       $vlPicRef:=1+Abs(Random)
    Until (Find in array($alPicRef;$vlPicRef)<0)
⇒   SET PICTURE TO LIBRARY(vgPicture;$vlPicRef;"New Picture")
```

2. The following example imports into the Picture Library the pictures (stored in a document on disk) created by the third example for the command PICTURE LIBRARY LIST:

```
    SET CHANNEL(10;"")
    If (OK=1)
       RECEIVE VARIABLE($vsTag)
       If ($vsTag="4DV6PICTURELIBRARYEXPORT")
          RECEIVE VARIABLE($vlNbPictures)
          If ($vlNbPictures)
             For($vlPicture;1;$vlNbPictures)
                RECEIVE VARIABLE($vlPicRef)
                If (OK=1)
                   RECEIVE VARIABLE($vlPicName)
                End if
                If (OK=1)
                   RECEIVE VARIABLE ($vgPicture)
                End if
                If (OK=1)
⇒                  SET PICTURE TO LIBRARY($vgPicture;$vlPicRef;$vlPicName)
                Else
                   $vlPicture:=$vlNbPictures+1
                   ALERT("This file looks like being damaged.")
                End if
             End for
          Else
             ALERT("This file looks like being damaged.")
          End if
       Else
          ALERT("The file ""+Document+"" is not a Picture Library export file.")
       End if
       SET CHANNEL(11)
    End
```

See Also

GET PICTURE FROM LIBRARY, PICTURE LIBRARY LIST, REMOVE PICTURE FROM LIBRARY.

System Variables and Sets

None is affected.

Error Handling

If there is not enough memory to add the picture to the Picture Library, an error -108 is generated. Note that I/O errors may also be returned (i.e., the structure file is locked). You can catch these errors using an error-handling method.

REMOVE PICTURE FROM LIBRARY

Pictures

version 6.0.2

REMOVE PICTURE FROM LIBRARY (picRef)

Parameter	Type		Description
npicRef	Number	→	Reference number of Picture Library graphic

Description

The command REMOVE PICTURE FROM LIBRARY removes from the Picture Library the picture whose reference number is passed in picRef.

If there is no picture with that reference number, the command does nothing.

4D Server: REMOVE PICTURE FROM LIBRARY cannot be used from within a method executed on the server machine (stored procedure or trigger). If you call REMOVE PICTURE FROM LIBRARY on a server machine, nothing happens—the call is ignored.

Warning: Design objects (hierarchical list items, menu items, etc.) may refer to Picture Library graphics. Use caution when deleting a Picture Library graphic programmatically.

Examples

1. The following example deletes the picture #4444 from the Picture Library.

```
⇒   REMOVE PICTURE FROM LIBRARY(4444)
```

2. The following example deletes from the Picture Library any pictures whose names begin with a dollar sign ($):

```
   PICTURE LIBRARY LIST($alPicRef;$asPicName)
   For($vlPicture;1;Size of array($alPicRef))
      If ($asPicName{$vlPicture}="$@")
⇒        REMOVE PICTURE FROM LIBRARY($alPicRef{$vlPicture})
      End if
   End for
```

See Also

GET PICTURE FROM LIBRARY, PICTURE LIBRARY LIST, SET PICTURE TO LIBRARY.

3 Process Communications (6.0.2)

VARIABLE TO VARIABLE — Process (Communications)

version 6.0.2

VARIABLE TO VARIABLE (process; dstVar; srcVar{; dstVar2; srcVar2; ...; dstVarN; srcVarN})

Parameter	Type		Description
process	Number	→	Destination process number
dstVar	Variable	→	Destination variable
srcVar	Variable	→	Source variable

Description

The command VARIABLE TO VARIABLE writes the dstVar process variables (dstVar2, etc.) of the destination process whose number is passed in process using the values of the variables srcVar1 srcVar2, etc.

VARIABLE TO VARIABLE has the same action as SET PROCESS VARIABLE, with the following differences:

• You pass source expressions to SET PROCESS VARIABLE, and therefore cannot pass an array as a whole. You must exclusively pass source variables to VARIABLE TO VARIABLE, and therefore can pass an array as a whole.

• Each destination variable of SET PROCESS VARIABLE can be a variable or an array element, but cannot be an array as a whole. Each destination variable of VARIABLE TO VARIABLE can be a variable or an array or an array element.

For each couple of dstVar;expr variables, the source variable must be of a type compatible with the destination variable, otherwise you may end up with a meaningless value in the variable. In interpreted mode, if a destination variable does not exist, it is created and assigned with the type and value of the source variable.

The current process “pokes” the variables of the destination process—the destination process is not warned in any way that another process is writing the instance of its variables.

Restrictions

VARIABLE TO VARIABLE does not accept local variables as destination variables.

VARIABLE TO VARIABLE accepts any type of destination process or interprocess variables except:
- Pointers
- Array of pointers
- Two-dimensional arrays

The destination process must be a user process; it cannot be a kernel process. If the destination process does not exist, an error is generated. You can catch this error using an error-handling method installed with ON ERR CALL.

Example

The following example reads a process array from the process indicated by $vlProcess, sequentially sets the elements to uppercase and then writes back the array as a whole:

```
    GET PROCESS VARIABLE($vlProcess;at_IPCom_Array;$anArray)
    For($vlElem;1;Size of array($anArray))
       $anArray{$vlElem}:=Uppercase($anArray{$vlElem})
    End for
⇒   VARIABLE TO VARIABLE($vlProcess;at_IPCom_Array;$anArray)
```

See Also

GET PROCESS VARIABLE, Processes, SET PROCESS VARIABLE.

4 Users and Groups (6.0.2)

GET USER PROPERTIES (6.0.2)

Users and Groups (6.0.2)

version 6.0.2

GET USER PROPERTIES (userID; name; startup; password; nbLogin; lastLogin{; memberships})

Parameter	Type		Description
userID	Number	→	Unique user ID number
name	String	←	Name of the user
startup	String	←	Startup method name
password	String	←	Always an empty string
nbLogin	Number	←	Number of logins to the database
lastLogin	Date	←	Date of last login to the database
memberships	Numeric Array	←	ID numbers of groups to which the user belongs

The command GET USER PROPERTIES no longer returns the encrypted password in the password parameter. Starting with version 6.0.2, an empty string is always returned in this parameter.

Note: The *4th Dimension Language Reference* manual delivered with version 6.0.2 does not describe this change.

See Also

GET USER PROPERTIES, SET USER PROPERTIES, SET USER PROPERTIES (6.0.2), Validate password (6.0.2).

SET USER PROPERTIES (6.0.2)

Users and Groups (6.0.2)

version 6.0.2

SET USER PROPERTIES (userID; name; startup; password; nbLogin; lastLogin{; memberships})

Parameter	Type		Description
userID	Number	→	Unique ID number of user account, or -1 for adding a user affiliated with the Designer, or -2 for adding a user affiliated with the Administrator
		←	Unique ID number of new user
name	String	→	New user name
startup	String	→	Name of new user startup method
password	String	→	New (unencrypted) password, or * to leave the password unchanged
nbLogin	Number	→	New number of logins to the database
lastLogin	Number	→	New date of last login to the database
memberships belongs	Numeric Array	→	ID numbers of groups to which the user

The command SET USER PROPERITES now accepts the * symbol as a value for the password parameter. This allows you to change the other properties of the user account without changing the password for this account.

Note: The *4th Dimension Language Reference* manual delivered with version 6.0.2 does not describe this change.

See Also

GET USER PROPERTIES, GET USER PROPERTIES (6.0.2), SET USER PROPERTIES, Validate password (6.0.2).

Validate password (6.0.2)

Validate password (userID; password)

Parameter	Type		Description
userID	Number	→	Unique user ID number
password	String	→	Unencrypted password

Description

The command Validate password returns True if the string passed in password is the password for the user account whose ID number is passed in userID.

Example

The following example checks whether the password of the user “Hardy” is “Laurel”:

```
      GET USER LIST(atUserName;alUserID)
      $vlElem:=Find in array(atUserName;"Hardy")
      If ($vlElem>0)
⇒        If (Validate password(alUserID{$vlElem};"Laurel")>0)
            ALERT("Yep!")
         Else
            ALERT("Too bad!")
         End if
      Else
         ALERT("Unknown user name")
      End if
```

See Also

GET USER PROPERTIES (6.0.2), SET USER PROPERTIES (6.0.2).

5 Web Server (6.0.2)

The Text Parameter Passed to 4D Methods Called via URLs Web Server

version 6.0.2

4th Dimension sends a text parameter to any 4D method called via a URL. Regarding this text parameter:

• Although you do not use this parameter, you must explicitly declare it with the line C_TEXT($1), otherwise runtime errors will occur while using the Web to access a database that runs in compiled mode.
• This parameter returns the extra data placed at the end of the URL, and can be used as a placeholder for passing values from the HTML environment to the 4D environment.

Runtime Errors in Compiled Mode

Let's consider the following example. You execute a method bound to an HTML object using a link and obtain the following screen on your Web browser:

This runtime error is related to the missing declaration of the text $1 parameter in the 4D method that is called when you click on the HTML link referring to that method. As the context of the execution is the current HTML page, the error refers to the "line 0" of the method that has actually sent the page to the Web browser.

Following the example from the section Web Services, Your First Time (Part I), you eliminate the problem by explicitly declaring the text $1 parameter within the M_ADD_RECORDS and M_LIST_RECORDS methods:

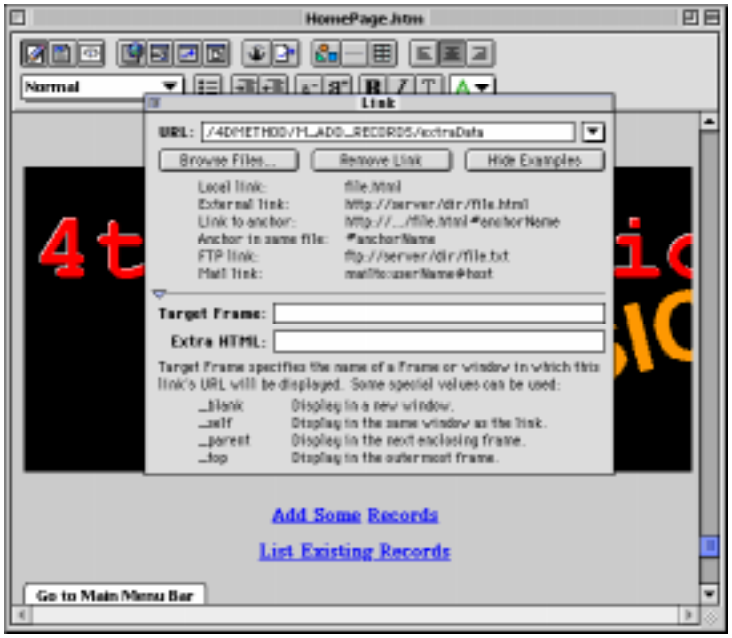
```
      ` M_ADD_RECORDS project method
⇒     C_TEXT($1) ` This parameter MUST be declared explicitely
      Repeat
         ADD RECORD([Customers])
      Until(OK=0)

      ` M_LIST_RECORDS project method
⇒     C_TEXT($1) ` This parameter MUST be declared explicitely
      ALL RECORDS([Customers])
      MODIFY SELECTION([Customers])
```

After these changes have been made, the compiled runtime errors no longer occur.

Working with the URL Extra Data

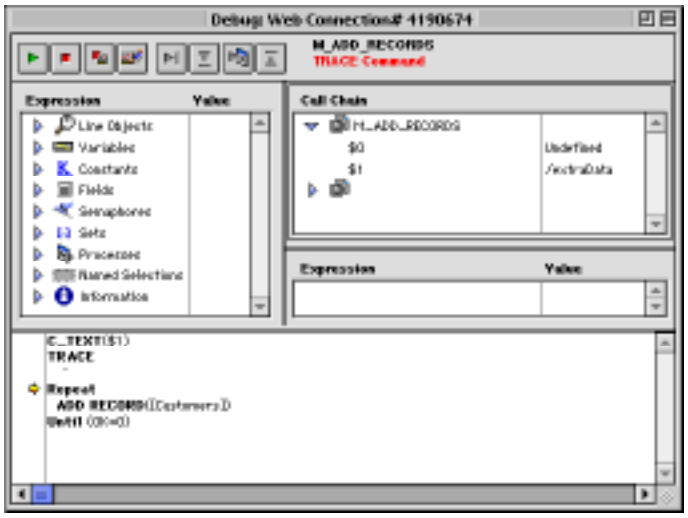
The text $1 parameter passed to the 4D method returns the extra data appended to the URL.

Again following the example in the *4th Dimension Language Reference* manual, the change (shown in the following figure) is made to the URL of the link that refers to the M_ADD_RECORDS method:



Note: The figure depicts the change as made using Claris Home Page on MacOS.

The data added to the URL is therefore the string “/extraData”. After this change has been made, you can use the Debugger window, on the 4D side, to quickly check that the $1 parameter actually returns the string “/extraData”:



By using conventions and algorithms similar to those described in the section On Web Connection Database Method of the *4th Dimension Language Reference* manual, you therefore have the means to exchange additional data between the HTML and the 4D environments when a 4D method is called by an HTML link.

How to Dynamically Set the URL Extra Data

If you create and write your own HTML files “on the fly” (using, for example, Create document and SEND PACKET), you simply write the URLs accordingly to your needs.

If you work with existing HTML files, you can use JavaScript to dynamically set the link properties of your objects.

See Also

Web Services, Your First Time (Part II).

CHANGE WEB LICENSE

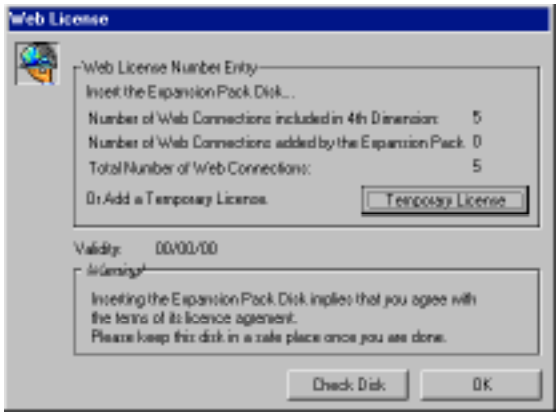
Parameter	Type	Description

This command does not require any parameters

Description

The command CHANGE WEB LICENSE displays the Web License dialog box, which enables the user to add and remove licenses to and from the built-in 4th Dimension Web Server.

Web License dialog box on Windows:



Web License dialog box on Macintosh:

This dialog box was originally available only in the Design environment. Using the CHANGE WEB LICENSE command, you can display the Web License dialog box in the User and Custom menus environment.

Note: In the Design environment, you display this dialog box by clicking on the Licenses button in the Database Properties dialog box.
Tip: CHANGE WEB LICENSE is a convenient way to allow Web licensing expansion in a compiled and merged 4D application distributed to customers. 4D developers or IS managers can use this command to distribute a 4D application and let users expand their Web License without sending an update of the application each time.

Example

In a custom configuration or preferences dialog box, you include a button whose method is:

```
` bWebLicense button object method
⇒ CHANGE WEB LICENSE
```

In doing so, you enable the user to increase the number of users who can connect simultaneously to the 4D Web Server, without modifying the database itself.