ISIS System 7 Pack™

Version 3.3

Written by Mike Cohen

ISIS International
14270 Dickens St., Suite 6
Sherman Oaks, CA 91423-4196

(818) 788-4747

Introduction

System 7 Pack™ allows 4th DIMENSION® to take advantage of one of System 7's most powerful features: AppleEvents. In addition, you can open or print documents and launch or terminate any application. System 7 Pack™ also provides the ability to execute Frontier™ scripts from within 4th DIMENSION®.

Using AppleEvents, you can request any running application to open or print a document. Certain applications which support scripting (most notably Frontier™, HyperCard®, and SuperCard®) can also be requested to execute almost any command via AppleEvents. System 7 Pack™ will also allow you to install a 4D procedure to be executed in response to an AppleEvent. You can also send AppleEvents to compatible applications running on a remote networked Macintosh. AppleEvents will also allow certain applications such as Frontier™ to extract information from 4D or request 4D to execute a command.

Not all applications currently support AppleEvents. However, the Launch/Open Document and Quit functions will work for all applications whether or not they are AppleEvent aware.

System 7 Pack requires 4th DIMENSION® version 2.2 or later and any Macintosh® that is capable of running System 7. Earlier versions of 4D® will allow you to launch applications and send AppleEvents, but won't allow AppleEvents to be received. Version 3.3 of System 7 Pack is fully compatible with 4D® v3.0.

Installing System 7 Pack

The easiest way to install System 7 Pack™ is to simply drop the enclosed **Proc.Ext** file into your database project folder. You can also use 4D®'s **External Mover** application to copy the package into an existing Proc.Ext file. If you have **4D Mover™** from ACIUS, you can also use it to copy the single *Package* from the demo database to your own database. In either **External Mover** or **4D Mover™**, System 7 Pack™ will appear as a single *Package*, which actually consists of many individual commands. The individual commands **cannot** be moved separately.

You can also place the Proc.Ext file in your System folder or 4D®'s folder to make it accessible to all databases. If you do it this way, make sure you copy the Proc.Ext file if you wish to use your database on a different system.

You may also wish to place the **S7P Help** desk accessory in your **Apple Menu Items** folder so you can access an online command reference.

What is an AppleEvent?

An AppleEvent is a message sent between two programs running under System 7 (a program can also send AppleEvents to itself). AppleEvents are identified by a 4-letter event class and a 4-letter event ID. An AppleEvent has certain *attributes* (which are added automatically when the event is created), such as the address of the program that sent it, and any number of *parameters*, which provide data or give more information about the event. AppleEvents can either request information from a program or tell a program to take some action.

AppleEvent parameters and keywords are identified by 4-character keywords. Many AppleEvents will only have a single *direct object*, which is identified by the key "----".

AppleEvents are limited to a total size of 64K, which includes all of the data, parameters, and attributes.

Using System 7 Pack™'s simple high-level interface, you can send events with a single parameter. In order to create more complex events, you must use the low-level interface. See the section "Sending AppleEvents" for more information.

Built-in AppleEvent Handlers

System 7 Pack™ installs several AppleEvent handlers which allow certain applications, such as Frontier™, to request information from 4D®. In addition, it installs a handler for the 'do script' AppleEvent which allows another application to execute most 4D® commands. **This event handling is OFF by default. Call *AllowAccess(1)* to allow these events to be processed.** When these events are disabled, the sending application will get back an error message "AppleEvents not allowed at this time".

If you're using a pre-3.0 version of 4D®, use extreme caution when sending any command to 4D® which could change the current record or selection. In particular, moving to a different record while a record is being modified in an input layout could cause changes to be lost. For this reason, I strongly suggest that ***AllowAccess(0)*** should be called before doing a **Modify Selection** or any other command which modifies or creates records. In version 3.0 or later of 4D, All AppleEvent handlers run in a separate process called "AEHandler", so AppleEvents can change the selection or current record without affecting any other process. Since one process is used for all AppleEvent processing, A series of AppleEvents can act on a current record or selection.

For a more complete description of the AppleEvents handled by System 7 Pack™, see Appendix A.

Sending AppleEvents

System 7 Pack™ includes several high-level commands which make it easy to send AppleEvents.

Before you can send an AppleEvent, you must create a target address. The address can be selected from a list of programs on the network using the **SelectAddress** command, or created with the commands **MakeAddress** (if the program resides on your machine) or **StringToAddr**.

Once you have a target address, the easiest way to send an AppleEvent is by using the high-level command **AESend**, which lets you specify the class and ID as well as a message which can consist of up to 32K of text. You can also send a picture instead of text by using **AESendPict**.

If the receiving application will return any meaningful information in response to the AppleEvent, you can use the command **SendWithReply** instead. NOTE: SendWithReply isn't intended to be used when sending a user-defined event to another 4D database, although it can be used to send one of the pre-defined events which return information.

If you need to send an event with more than one parameter or additional data types such as file aliases, object specifiers, or arrays you **must** use the low-level interface.

Sending an AppleEvent with the low-level functions requires three steps: First, you must create the event using **CreateAEVT**. Next, you must add any needed parameters using commands such as **PutObject**, **PutAliasParam**, **PutTextParam**, or **PutList**. Finally, you must use **SendAppleEvent** to actually send the event. When you're finished, you must dispose the event and the reply using **DisposeDesc**. For more information, see the section "Low Level Interface"

Example 1: Sending a "do script" using high-level commands (the DoScript & Frontier commands can simplify this even more)

```
$Err:=MakeAddress("LAND";$Addr)
if ($Err=0)
  $Err:=AESend($Addr;"misc";"dosc";"message("hello")")
end if
```

Example 2: Using the low-level interface to send a "Get Data" event and access the reply

```
$Err:=MakeAddress("DFB0";$Addr)
if ($Err=0)
  $Err:=CreateAEVT("core";"getd";$Addr;$myAEVT)
  if ($Err=0)
    $Err:=PutObject($myAEVT;ObjNamed("cMPV";0;"MPVariable"))
    if ($Err=0)
      $Err:=SendAppleEvent($myAEVT;$myReply;kAEWaitReply;-1)
      if ($Err=0)
        $Err:=GetTextParam($myReply;"----";Result)
        $Err:=DisposeDesc($myReply)
      end if
    end if
    $Err:=DisposeDesc($myAEVT)
  end if
  $Err:=DisposeDesc($Addr)
end if
```

Receiving AppleEvents

In order to receive an AppleEvent, you must install a handler using the function **HandleAEVT**. A handler is a normal 4D procedure which will be executed when an AppleEvent of the specified class and ID is received.

System 7 Pack™ provides several commands which allow an AppleEvent handler to extract information from the received event. To obtain the address of the sending application, use **GetReturnAddr**. The resulting target address can be used to send an event back to the same application. To determine the data type of the direct object, use **GetAEType**.

The easiest way to extract the direct object if it is text or a numeric value is by using **GetAEMessage**. To obtain a picture that was sent using **AESendPict**, use **GetAEPict**. NOTE: GetAEPict should only be used to extract a picture that was sent via **AESendPict**. In all other cases, you should use the low-level function **GetPicParam**.

If the received event contains additional parameters or uses other data types, you must use the low-level functions, such as **GetTextParam**, **GetAliasParam**, or **GetList**.

Note for 4D v3.0: All AppleEvent handlers run in a single process called AEHandler. This process has no window associated with it unless a procedure explicitly creates a window. **You must never call MESSAGE from such a procedure without first creating a window.**

Example 1: Using the high-level functions to receive text & send acknowledgement

```
GetAEMessage($theMessage)
GetReturnAddr($who)
Alert($theMessage)
$Err:=SendAEVT($who;"TEST";"DEMO";"OK, I got the message!")
$Err:=DisposeAddress($who)
```

Example 2: Using the low-level functions to receive an array and create new records

```
$Err:=GetList(0;"----";anArray)
if ($Err=0)
  ArrayToSelection(anArray;[File1]Field1)
End if
```

Example 3: Waiting for an event to be received

```
HandleAEVT("aevt";"xxxx";"SetFlag")
EventReceived:=False ` the event handler will change this
while (Not(EventReceived))
  idle
  ProcessAEVT
end while

` Procedure SetFlag -- executed in response to AppleEvent
EventReceived:=True
```

Command Reference

The following commands are added to 4th DIMENSION®'s programming language when you install System 7 Pack™. They will be separated into logical groupings in the procedure editor window.

Launching & Quitting

Launch

```
Err:=Launch(Signature;Document Name)
```

This will launch the application associated with the signature (example: to launch Microsoft Word®, give it "MSWD"). If the document name is not an empty string, it will be opened when the application starts up. You can also use this to tell an application that's already running to open a document. This will work correctly for any application whether or not it supports AppleEvents.

IMPORTANT NOTE: An application launch doesn't complete until the frontmost application calls GetNextEvent. Unfortunately, 4D doesn't do so while a procedure is running. If you need to call launch from a procedure which must continue running, you must display a dialog or open a window and call **ProcesAEVT** and **IDLE** to allow the launch to complete by forcing 4D to call GetNextEvent.

LaunchBehind

```
Err:=LaunchBehind(Creator;Document)
```

Works exactly the same as Launch except the application won't be brought to the front.

PrintDoc

```
Err:=PrintDoc(Signature;Document Name)
```

This is similar to Launch, except that the document will be printed. Unlike Launch, you must give it a non-blank document name.

QuitApp

```
Err:=QuitApp(Signature)
```

This will tell the specified application to quit. In many cases the application will ask whether any files should be saved. Some applications need to be brought to the front before quitting so they can display any alerts necessary.

Misc. Utilities

BringToFront

```
Err:=BringToFront(Signature)
```

Brings the application specified by a 4-character signature to the front. The application must already be running. This only requests that the application be brought to the front. It won't actually happen until 4D gives up CPU time by calling WaitNextEvent. In pre-3.0 versions of 4D you may have to call **ProcessAEVT** to force it to happen.

FindAppName

```
Err:=FindAppName(Signature;Name)
```

Given a 4-character signature, this will return the name of the application associated with that ID. It will return the name without a full path.

FindCreator

```
Err:=FindCreator(File Name;Creator ID)
```

Given a full or partial pathname of a file, will return the creator ID. This is useful to figure out which application to launch when any document is selected.

Example:

```
Set Channel(10;"")
if (OK = 1)
      Set Channel(11)
      $err := FindCreator(Document;$creator)
      if ($err = 0)
            $err := Launch($creator;Document)
      end if
end if
```

IsRunning

```
Result:=IsRunning(Creator ID)
```

Returns 1 if the application is running, 0 otherwise.

Long

```
Result:=Long(TypeString)
```

Converts a 4-character type string to a long integer value. This is useful when a type or application signature needs to be passed to one of the low-level functions.

ProcessList

```
Err:=ProcessList(Visible;StringArray)
```

Builds an array of the signatures of all running processes. If 'visible' is non-zero, only the visible applications will be listed. Otherwise, the list will also include invisible processes, such as background-only applications and File Sharing Extension. The array will be allocated if necessary and will contain a 4-character signature string for each running (or visible) application. In a compiled database, you must pre-define the array as fixed 4 character strings, although the number of elements will change dynamically.

Example:

```
If (Before)
  $err:=ProcessList (0;ProcList)
  If ($err=0)
    ARRAY STRING(64;ProcNames;Size of array(ProcList))
    For ($i;1;Size of array(ProcList))
      $err:=FindAppName (ProcList{$i};$pn)
      ProcNames{$i}:=$pn
    End for
  End if
End if
```

S7Version

```
Version:=S7Version(Serialization String)
```

Returns the serialization string and version # of System 7 Pack™ as a long integer value, which will be negative for a demo version. The current version is 3.26, which will be returned as 326 (or -326 for the demo version).

SetTimeout

```
SetTimeout(Ticks)
```

Sets the time-out value in ticks (60ths of a second) to use when sending an AppleEvent. A good value to use is "600", which is 10 seconds. To specify the default value, which is about a minute, pass -1. If the value is too short, you may get an error code of -1712 (time out) when sending an AppleEvent. If the value is too long, the system will appear to "hang" while waiting for a response if none is returned.

System7

```
Result:=System7
```

Returns 1 if you're running a compatible system which supports AppleEvents. This should be one of the first calls in your program, so you can provide a graceful exit if necessary. Other System 7 Pack™ functions will not crash if you're not running system 7 but will simply return an error code of -2.

AE Process ID

```
Result:=AE Process ID
```

Returns the Process ID of the AppleEvent handler in 4D® v3.0 or later. This will always return 0 in a pre-3.0 version of 4D, so it can be used to test whether multiple processes are supported in the running version of 4D.

Target Addresses

AddrToString

```
Err:=AddrToString(Target;ATLocation;Name;Port)
```

Converts a target address returned by MakeAddress, SelectAddress, or GetReturnAddr into strings which may be saved between sessions. ATLocation describes the machine's network identity and will be a string something like 'Mike:PPCToolBox@*'. It will be an empty string if the program is running on your local machine. 'Name' is the actual name of the program. Port is a PPC Toolbox port ID (usually the creator ID followed by "ep01").

DisposeAddress

```
Err:=DisposeAddress(Target)
```

Disposes the target address handle that was previously allocated with MakeAddress, SelectAddress, or GetReturnAddr. This is assumed to be a handle to an address descriptor. Since this is passed to DisposHandle, the system may crash if this isn't a valid handle. Note that DisposeAddress is the same as the new low-level function DisposeDesc, but has been kept for compatibility with older versions.

MakeAddress

```
Err:=MakeAddress(Signature;Target)
```

Creates a target address from an application signature. The application must be running on the local system. You must not modify this value in any way and should call DisposeAddress when you're finished using this handle.

SelectAddress

```
Err:=SelectAddress(Prompt;Signature;Target)
```

Opens the PPC Browser window, which allows you to select a program on your system or on a remote system to send AppleEvents to. If prompt isn't an empty string, it will be used instead of the default "select a program to link to" message. If the signature is an empty string, all programs will be listed, otherwise only programs matching that signature will be listed. If this function returns 0, a handle to an address descriptor will be allocated and placed into 'target'. You must not modify this value in any way and should call DisposeAddress when you're finished using this handle.

Example:

```
if (SelectAddress("Select a copy of Hypercard:";"WILD";$Target)=0)
      $err:=DoScript($Target;"Go To Stack 'my stack'")
      $err:=DoScript($Target;"Go To Next Card")
      $err:=DoScript($Target;"Print Card")
      $err:=DisposeAddress($Target)
end if
```

StringToAddr

```
Err:=StringToAddr(ATLocation;Name;Port;Target)
```

Creates a target ID from a set of strings describing it. This is the complementary function to AddrToString. ATLocation describes the machine's network identity (should be an empty string if the program is running on your local machine). Otherwise, it will be something like 'name:PPCToolBox@zone'. 'Name' is the actual name of the program. Port is a PPC Toolbox port ID, which is the 4-letter creator ID (it's actually the creator ID followed by "ep01", but the creator ID alone is allowed).

Scripting

DoScript

```
Err:=DoScript(Target;Command)
```

Sends a "do script" AppleEvent to an application which must already be running. 'Command' is a text variable, field, or literal and will be in a format specific to the receiving application. For Hypercard, this should be any HyperTalk command. For Frontier, it should be a UserTalk command. Other applications will have different command formats. This can also be used with Excel 4.0 to execute any function or macro. **NOTE**: DoScript won't wait for the commands to complete, so if you need to synchronize operations in another application, you should use **SendWithReply** instead.

Example:

```
if (IsRunning("WILD")=0)
      $err := Launch("WILD";"")
end if
if ($err = 0)
      $err := MakeAddress("WILD";$ad)
      if ($err = 0)
            $err:=DoScript($ad;"Go To Stack 'my stack'")
            $err:=DoScript($ad;"Go To Next Card")
            $err:=DoScript($ad;"Print Card")
            $err := DisposeAddress($ad)
      end if
end if
```

Evaluate

```
Err:=Evaluate(Target;Command;Result)
```

Sends an "evaluate" message to an application and waits for the result, which will be of type TEXT. This is most often used with HyperCard® and SuperCard®.

Example:

```
$err:=MakeAddress("WILD";$hc)
if ($err = 0)
  $err:=Evaluate($hc;"the long name of the target";$result)
  alert($result)
  $err:=DisposeAddress($hc)
end if
```

Frontier

```
Err:=Frontier(Command;Result)
```

Sends a command to UserLand Frontier™ (which must already be running on your machine) and returns the result or an error message. If you pass the name of a Frontier™ object (such as User.Name) rather than an executable command, the contents of that object will be returned.

Example:

```
$err:=Frontier("User.Name";$myName)
If ($err=0)
  $q:=Char(34)
  $err:=Frontier("msg("+$q+"Hello, "+$myname+$q+")";$rep)
  $err:=Frontier("Frontier.BringToFront()";$rep)
  $err:=Frontier("edit(readme)";$rep)
End if
```

QuicKeys

```
Err:=QuicKeys(Macro Name)
```

Executes a QuicKeys™ macro. You must be running QuicKeys2 v2.1 or later with CEIAC. The macro must be installed in the Universal keyset or 4th DIMENSION®'s keyset.

Sending AppleEvents

AESend

```
Err:=AESend(Target;Class;ID;Message)
```

Sends any AppleEvent with a single direct object of type TEXT. The event is sent asynchronously and 4D doesn't wait for a reply. If you need to wait for a reply, use SendWithReply instead. **Note that the AppleEvent manager can coerce a string which represents a number into any numeric format at the receiving end.**

Example:

```
$err:=MakeAddress("WILD";$hc)
if ($err = 0)
      $err:=AESend($hc;"misc";"dosc";"go to next card")
      $err:=DisposeAddress($hc)
end if
` this is really the same as 'DoScript'
```

AESendPict

```
Err:=AESendPict(target;class;id;aPicture)
```

Sends an AppleEvent with a picture as the direct object. This is most useful for sending pictures between 4D® databases, as pictures will be sent in 4D's internal format. If you need to send a picture to an application other than 4D, use the low level function PutPicParam. NOTE: The maximum size of all data in an Apple-Event is 64K, so extremely large pictures can't be sent this way.

Example:

```
$err:=AESendPict(remoteDB;"DEMO";"PICT";aPicture)
if ($err#0)
      Alert("Error sending picture:"+string($err))
end if
```

SendWithReply

```
Err:=SendWithReply(Target;Class;ID;message;reply)
```

Sends any AppleEvent with a single direct object of type TEXT and returns the direct object of the reply as text. If an error occurs, it will return the 'errs' (error string) parameter instead. If no reply or error message was available, the reply will be an empty string. **This is not intended to be used for sending custom events to other 4D databases, since System 7 Pack handles events asynchronously and doesn't return a reply with any meaningful information.** This can be used, however, to send one of System 7 Pack's pre-defined events which extract information from 4D.

This command can also be used to force 4D to wait until the other application finishes processing an AppleEvent before it proceeds. Instead of using **DoScript** you can use **SendWithReply(target;"misc";"dosc";"some commands";reply)** and simply ignore the reply.

AppleEvent Handling

AllowAccess

```
AllowAccess(Flag)
```

Turns on or off System 7 Pack™'s standard AppleEvent handling (built in handlers for DoScript and data & structure access events). The event handling is initially turned off. Calling this function with a non-zero Flag will enable standard events. Calling this with 'Flag' set to zero will disable standard event handling. If an application tries to send an AppleEvent when they're disabled, it will get back an error message 'AppleEvents not allowed at this time'. This doesn't affect handlers installed with HandleAEVT.

GetAEMessage

```
GetAEMessage(Msg)
```

Gets the text-based direct object received with the last AppleEvent. This will usually be called in a procedure installed via HandleAEVT. Note: versions 2.2 and later allows pictures to be sent as well as text. To make sure that what you received is text, you should call GetAEType and make sure the type isn't PICT (any numeric type can be converted to text and will be returned as a string by GetAEMessage, but pictures will result in a null string). In version 3.0, you can also use the new low-level interface to extract multiple pieces of data with finer control.

GetAEPict

```
Err:=GetAEPict(aPicture)
```

This should be called in a procedure installed via HandleAEVT to retrieve picture data from the AppleEvent. Use this function to handle an event that was sent with AESendPict.

Example:

```
HandleAEVT("DEMO";"PICT";"Receive Picture")

-- Receive Picture --
GetAEType($type)
If ($type # "PICT")
      Alert("Wrong data type received:"+$type)
Else
      $err:=GetAEPicture(picVariable)
      if ($err#0)
         Alert("Error receiving picture:"+string($err))
      end if
      Create Record([Pictures])
      [Pictures]thePicture:=picVariable
      Save Record([Pictures])
End if
```

GetAEType

```
GetAEType(theType)
```

Returns the data type of the direct object of the last AppleEvent. This should be called in a procedure installed via HandleAEVT to ensure that the received data is in the expected format. A procedure that expects PICT will be able to receive only PICT by calling GetAEPict. A procedure that expects text can receive any text or numeric type by calling GetAEMessage.

GetReturnAddr

```
GetReturnAddr(Target)
```

Gets the return address of the last AppleEvent received. The target is identical to one returned by SelectAddress and may be used in any of the AppleEvent functions which take a target address. It should be disposed of when you finish with it. This is most useful in a procedure installed via HandleAEVT.

HandleAEVT

```
Err:=HandleAEVT(Class;ID;Command)
```

Installs a 4D® command line or procedure to be executed in response to a particular AppleEvent. Don't try to replace any of the 4 required events (class 'aevt' & ID 'odoc', 'oapp', 'pdoc', or 'quit') or one of the standard events handled by System 7 Pack™. **Note**: versions 3.1 and later of S7P allow you to remove the built-in ‘aevt’,’quit’ handler, after which you can add your own quit procedure which can do any necessary clean up and then call **QUIT 4D**.

4D 3.0 Note: All AppleEvent handlers execute in a process called AEHandler. **This process has no window associated with it. Therefore, you must never call MESSAGE without first creating a window.**

Example:

```
err:=HandleAEVT("DEMO";"TEST";"DemoHandler")

-- DemoHandler --
GetAEMessage($txt)
GetReturnAddr($sender)
$err:=AESend($sender;"DEMO";"ACK!";$txt)
Alert($txt)
$err:=DisposeAddress($sender)
```

IgnoreAEVT

```
Err:=IgnoreAEVT(Class;ID)
```

Removes a previously installed AppleEvent handler. **Note:** version 3.1 will now allow you to remove the built-in handler for ‘aevt’,’quit’. Don't use this to remove any of 4D's standard event handlers other than ‘quit’.

ProcessAEVT

```
ProcessAEVT
```

Allows AppleEvents to be received and processed while 4D is in a tight processing loop. You only need to use this function if a loop is waiting for an AppleEvent to set some flag before it exits, since 4D normally doesn’t poll for events during such loops. **THIS COMMAND SHOULD NEVER BE NECESSARY IN 4D V3.0**.

Example:

```
MyEventFlag := False            ` will be set in AE handler
while (MyEventFlag = False)
      ProcessAEVT
end while
```

Low Level Interface

CreateAEVT

```
Err:=CreateAEVT(Class;ID;Addr;theEvent)
```

Creates an AppleEvent of the specified class and ID with the specified target address. This doesn't add any parameters to the event and doesn't actually send it.

Example:

```
$err:=CreateAEVT("misc";"dosc";theTarget;$myEvent)
if ($err=0)
 $err:=PutTextParam($myEvent;"----";"message(“Hi there!”)")
 if ($err=0)
  $err:=SendAppleEvent($myEvent;$myReply; kAENoReply+CanInteract ;-1)
  $err:=DisposeDesc($myReply)
 end if
 $err:=DisposeDesc($myEvent)
end if
```

DisposeDesc

```
Err:=DisposeDesc(desc)
```

Disposes an AppleEvent or Target Address descriptor. This should always be called for an AppleEvent created with CreateAEVT and the reply created when the event is sent as well as a target address that is no longer needed.

GetAliasParam

```
GetAliasParam(theEvent;theKey;aFileName)
```

Extracts a file alias parameter from an AppleEvent. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle. The alias will be converted to a string representing the full pathname of the file.

GetList

```
Err:=GetList(anEvent;theKey;anArray)
```

Extracts a descriptor list from an AppleEvent into a 4D array. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle. All items in the list must be of the same type. The array will be created automatically of the correct type to hold the contents of the list. A list of lists or AERecords will be returned as an array of long integers, with each element containing a handle to a descriptor. **The type of an existing array won’t be changed, and if you’re using an existing array, a list of strings requires a TEXT array. In a compiled database, the array must be previously defined and you must know the data type in advance.**

GetLongParam

```
Err:=GetLongParam(anEvent;keyWord;actualType;Value)
```

Extracts a long integer parameter from an AppleEvent. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle.

GetPicParam

```
Err:=GetPicParam(anEvent;theKey;convert;aPicture)
```

Extracts a picture parameter from an AppleEvent. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle. If the picture came from another 4D application, 'convert' should be 0, otherwise it should be non-zero and will cause the picture to be converted from standard Macintosh format to 4D's internal picture format.

GetRealParam

```
Err:=GetRealParam(anEvent;theKey;Value)
```

Extracts a floating point number parameter from an AppleEvent. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle. The value can be any numeric type and will be coerced into 4D's 10-byte extended format.

GetShortParam

```
Err:=GetShortParam(anEvent;keyWord;actualType;Value)
```

Extracts a short integer parameter from an AppleEvent. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle.

GetTextParam

```
Err:=GetTextParam(anEvent;theKey;Value)
```

Extracts a text parameter from an AppleEvent. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle.

Obj

```
ospec:=Obj(class;container;value)
```

Creates an object specifier identified by a numeric value. Class should be the 4-letter class ID. Container can be either another object specifier or 0 to specify a null container. The container will automatically be disposed of to simplify creating nested containers. Value should be numeric. For example, to create the specifer 'Word 1 of Paragraph 2 of Document "Foo"' you would use:

Obj("cwor";Obj("cpar";ObjNamed("docu";0;"Foo");2);1)

ObjNamed

```
ospec:=ObjNamed(Class;Container;Name)
```

Creates an object specifier identified by a name. Class should be the 4-letter class ID. Container can be either another object specifier or 0 to specify a null container. The container will automatically be disposed of to simplify creating nested containers. Name should be a string. For example, to create the specifer 'Word 1 of Paragraph 2 of Document "Foo"' you would use:

Obj("cwor";Obj("cpar";ObjNamed("docu";0;"Foo");2);1)

Property

```
ospec:=Property(Name;Container)
```

Creates an object specifier for a property. Container should be the object specifier the property applies to. The container will automatically be disposed of to simplify creating nested containers. Name should be the 4-letter property ID. For example, to create the specifer "font of word 1" you would use:

Property("font";Obj("cwor";0;1))

PutAliasParam

```
Err:=PutAliasParam(anEvent;theKey;fileName)
```

Adds a file alias to an AppleEvent. The parameter is identified by a keyword. The value should be a string representing a file name which will be converted to an alias record.

Example:

```
$err:=CreateAEVT("misc";"dosc";theTarget;$myEvent)
if ($err=0)
 $err:=PutAliasParam($myEvent;"----";"HD80:Scripts:Do this")
 if ($err=0)
  $err:=SendAppleEvent($myEvent;$myReply; kAENoReply ;-1)
  $err:=DisposeDesc($myReply)
 end if
 $err:=DisposeDesc($myEvent)
end if
```

PutList

```
Err:=PutList(anEvent;key;SpecialType;anArray)
```

Adds a descriptor list built from a 4D array to an AppleEvent. SpecialType can specify a data type other than the default implied by the type of the list (the only conversions are from text representing file names to a list of aliases and from long integer to AERecord or object specifiers, in all other cases the data format will remain unchanged). To specify text to alias conversion, SpecialType should be "alis". To send a list of AERecords or object specifiers, SpecialType should be “reco” and the long integer array should contain descriptors created with Obj, ObjNamed, or CreateAERec. In this case, each element of the array will automatically be disposed when you call PutList.

Example:

```
Selection To Array([file1]field1;anArray)
$err:=CreateAEVT("send";"data";theTarget;$myEvent)
if ($err=0)
 $err:=PutList($myEvent;"----";"";anArray)
 if ($err=0)
  $err:=SendAppleEvent($myEvent;$myReply;kAEWaitReply;-1)
  $err:=DisposeDesc($myReply)
 end if
 $err:=DisposeDesc($myEvent)
end if
```

PutLongParam

```
Err:=PutLongParam(anEvent;key;actualType;value)
```

Adds a long integer parameter to an AppleEvent. The parameter is identified by a keyword and can be given a special type (it defaults to typeLongInteger if 'actualType' is an empty string).

PutObject

```
Err:=PutObject(anEvent;key;anObject)
```

Puts an object specifier in an AppleEvent. THIS COMMAND WILL DISPOSE THE OBJECT SPECIFIER. This command should be used in conjunction with Obj and/or ObjNamed to create the specifier.

Example:

```
$err:=MakeAddress("DFB0";$addr)
if ($err=0)
 $err:=CreateAEVT("core";"getd";$addr;$myEvent)
 if ($err=0)
  $err:=PutObj($myEvent;"----";objNamed("cMPV";0;"theDate"))
  if ($err=0)
   $err:=SendAppleEvent($myEvent;$myReply;kAEWaitReply;-1)
   $err:=GetTextParam($myReply;"----";theResult)
   $err:=DisposeDesc($myReply)
  end if
 $err:=DisposeDesc($myEvent)
 end if
 $err:=DisposeDesc($addr)
end if
```

PutPicParam

```
Err:=PutPicParam(anEvent;theKey;convert;aPicture)
```

Adds a picture parameter to an AppleEvent. It will be identified by a keyword and will always be of type 'PICT'. If ‘convert’ is non-zero, the picture will be converted from 4D’s internal format to a standard Macintosh picture.

PutRealParam

```
Err:=PutRealParam(anEvent;theKey;Value)
```

Adds a floating point number to an AppleEvent. The number will be in 4D’s internal 10 byte floating point format and the descriptor type will be ‘exte’. Most applications will be able to convert it to their desired floating point type.

PutShortParam

```
Err:=PutShortParam(anEvent;key;actualType;value)
```

Adds a short integer parameter to an AppleEvent. The parameter is identified by a keyword and can be given a special type (it defaults to typeShortInteger if 'actualType' is an empty string).

PutTextParam

```
Err:=PutTextParam(anEvent;key;value)
```

Adds a text parameter to an AppleEvent. The parameter is identified by a keyword and will always be of type 'typeText'.

SendAppleEvent

```
Err:=SendAppleEvent(theEvent;theReply;send mode;time out)
```

Sends an AppleEvent previously created by CreateAEVT. SendMode can be any of the constants kAEWaitReply, kAENoReply, AlwaysInteract, CanInteract, and/or NeverInteract. Time Out specifies the time (in 1/60ths of a second) to wait for a reply.

CreateAERec

```
Err:=CreateAERec(theRecord)
```

Creates an AE Record, which is a list of keyword specified parameters within an AppleEvent. All commands which add and extract parameters from an AppleEvent can be applied the same way to an AE Record. The AE Record must be disposed when you finish using it (adding it to an AppleEvent or another AE Record with PutAERecord will automatically dispose it).

Example:

```
` this is how to send a request to Claris® Resolve™
` create the request range table
$Err:=CreateAERec($tabl)
$Err:=PutLongParam($tabl;"TLPT";"";100)
$Err:=PutLongParam($tabl;"BRPT";"";200)
$Err:=PutKeyword($tabl;"RTRN";"TEXT")
` create the record which holds the table
$Err:=CreateAERec($reqList)
$Err:=PutAERecord($reqlist;"TABL";$tabl)
` create the actual AppleEvent and add the record
$Err:=CreateAppleEvent("CLRS";"GVAL";$Resolve;$GetValue)
$Err:=PutAERecord($GetValue;"----";$reqList)
$Err:=SendAppleEvent($GetValue;$Reply;kAEWaitReply;-1)
` extract the data from the reply record
$Err:=GetAERecord($Reply;"----";$reqList)
$Err:=GetAERecord($reqList;"TABL";$tabl)
$Err:=GetTextParam($tabl;"VAL ";$resultString)
` clean up everything we allocated
$Err:=DisposeDesc($tabl)
$Err:=DisposeDesc($reqList)
$Err:=DisposeDesc($Reply)
$Err:=DisposeDesc($GetValue)
```

GetAERecord

```
Err:=GetAERecord(anEvent;Key;theRecord)
```

Extracts an AE Record from an AppleEvent or from another AE Record. Any commands which apply to AppleEvents for adding and/or extracting data can also be used on AE Records. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle.

PutAERecord

```
Err:=PutAERecord(anEvent;Key;theRecord)
```

Adds an AE Record to an AppleEvent or to another AE Record. The AE Record will automatically be disposed after this call. Note that function this is really the same as PutObject, since an object specifier is simply a special kind of AE Record.

PutKeyword

```
Err:=PutKeyword(anEvent;key;value)
```

Adds a 4-character keyword (enumerated type) to an AppleEvent or AE Record. If you need to pass a **type** rather than an enumeration, use **PutLongParam(aevt; key; ”type”; Long(value))** instead.

GetKeyword

```
Err:=GetKeyword(anEvent;key;value)
```

Extracts a 4-character enumerated type keyword from an AppleEvent or AE Record. To reference the last event received in a procedure installed via HandleAEVT, pass 0 for the event handle.

Constants

AlwaysInteract
CanInteract
NeverInteract

Use these constants in SendAppleEvent to specify the user interaction option of the send mode. These determine whether the receiving program will be allowed to interact with the user when processing the AppleEvent. In general you should specify “NeverInteract” when sending to a remote system. When sending to an application on the local machine, you should probably specify “AlwaysInteract” or “CanInteract” to allow it to bring up any dialogs or alerts necessary when handling the event.

kAENoReply
kAEWaitReply

Use these constants in SendAppleEvent to specify the reply option of the send mode (you can also use 3 for QueueReply, but in that case you’ll have to add a handler for reply events, which will be class “aevt” and ID “ansr”.)

Advanced Commands

CreateXAEVT

```
Err:=CreateXAEVT(Class;id;addr;returnID;TransID;aevt)
```

This works exactly like CreateAEVT except it allows you to supply a return ID and/or transaction ID. The return ID will allow you to uniquely identify the reply to this AppleEvent if you allow the reply to be queued and install a handler for 'aevt','ansr'. You can allow the return ID to be generated automatically by passing -1 for the return ID. The transaction ID, if non-zero, can be used to specify that a series of AppleEvents are part of a single operation (**not all applications support this feature**).

CopyDesc

```
Result:=CopyDesc(aevt)
```

Creates a duplicate copy of an AppleEvent, AERecord, or target ID. You can use CopyDesc(0) in an AppleEvent handler to obtain a handle to the actual event being processed. This is useful if you wish to write extensions to System 7 Pack™ which need to access an AppleEvent. When you finish using the result, you must dispose of it by calling DisposeDesc.

GetTransactionID
GetReturnID

```
Result:=GetTransactionID(aevt)
Result:=GetReturnID(aevt)
```

Obtains the transaction ID and return ID of an AppleEvent. You can pass 0 to use the AppleEvent currently being processed in an AppleEvent handler procedure. GetReturnID can be used in a handler for queued reply events (class 'aevt',id 'ansr') to determine which AppleEvent is being replied to.

GetAEInfo

```
Err:=GetAEInfo(Descriptor;Size;Keys;Types;Lengths)
```

Returns arrays of keywords, types, and lengths of each descriptor in an AppleEvent, AERecord, or List. 'Size', a LongInt variable, will be set to the number of descriptors. 'Keys' and 'Types' should be arrays of String(4) and 'Lengths' should be an array of long integers.

GetNthDesc

```
Err:=GetNthDesc(aevt,index,key,type,result)
```

Extracts a descriptor from an AppleEvent, AERecord, or List by position rather than by keyword. ‘index’ tells which descriptor to extract (the first one is 1). ‘key’ and ‘type’ will be set to the keyword and actual datatype of that descriptor. ‘result’ will be a handle to the descriptor. When you finish using it, you must dispose of the result by calling DisposeDesc.

GetNthItem

```
Err:=GetNthItem(aevt,index,key,type,textVar)
```

Extracts a value from an AppleEvent, AERecord, or List by position rather than by keyword and coerces the result to text. ‘index’ tells which descriptor to extract (the first one is 1). ‘key’ and ‘type’ will be set to the keyword and actual datatype of that descriptor. This will work for all numeric types and any other data which can be coerced to text. This command, along with GetAEInfo and GetNthDesc can be used to examine an entire AppleEvent and extract the contents of all nested lists.

Example:

```
DumpList($aevt;0)

--- DumpList
$err:=GetAEInfo ($1;$Size;Keys;Types;Lengths)
For ($i;1;$size)
  MESSAGE($2*"> ")
  $err:=GetNthDesc ($1;$i;$aKey;$aType;$aDesc)
  If (($aType="list") | ($aType="reco"))
    MESSAGE($aKey+"   "+$aType+Char(13))
    DumpList ($aDesc;$2+1)
  Else
    $err:=GetNthItem ($1;$i;$aKey;$aType;$value)
    MESSAGE($aKey+"   "+$aType+"   "+$value+Char(13))
  End if
  $err:=DisposeDesc ($aDesc)
End for
```

Using System 7 Pack with UserLand Frontier™

We provide a glue table for Frontier™ users to simplify access to Sytstem 7 Pack™'s AppleEvents. To install these commands into Frontier™, simply double-click on the file called "FourthDimension.Frontier". If Fontier™ isn't already running, it will start up. Frontier™ will ask you for the name of the table to import. Accept the default name of 'System.verbs.apps.FourthDimension'. The following new verbs will be added to Frontier™:

FourthDimension.countFiles()

Returns the number of files in your database.

FourthDimension.countFields(fileno)

Returns the number of fields in the specified file.

FourthDimension.fileName(fileno)

Returns the name of the specified file.

FourthDimension.fieldName(fileno,fieldno)

Returns the name of the specified field.

FourthDimension.currentRecord(fileno)

Returns the current record number in the specified file.

FourthDimension.recordsInSelection(fileno)

Returns the number of records currently selected in the specified file.

FourthDimension.fieldInfo(fileno,fieldno,@table)

Inserts information about the specified field in a table. The following entries will be created:

----	Name of the field
FTYP	a numeric code indicating the field type (See 4D®'s Language

FourthDimension.getField(fileno,fieldno)

Returns the contents of the specified field in the current record.

FourthDimension.doScript(command)

Executes almost any 4D® command. **In pre-3.0 versions of 4D, *Don't* execute a command which changes the current record or selection while a record is being displayed in an imput layout or data may be lost.**

FourthDimension.buildOutline(@outline)

Builds an outline describing the database structure.

FourthDimension.extractData(fileno,@outline)

Builds an outline containing all data in the specified file.

FourthDimension.searchForData(fileno,expr,@outline)

Does a simple search of a file and builds an outline containing only the records returned by the search.

Interfacing with Claris® Resolve™

In addition to the required AppleEvents and DoScript, Claris® Resolve™ Supports the following custom events:

CLRS	GVAL	Extracts data from the current worksheet.
CLRS	PVAL	Puts data into the current worksheet.

Both of these events require a special record which must be created with the low-level functions. This record consists of top left and bottom right points, and a return type (for GVAL) or return value (for GVAL reply or PVAL) and is contained in another record.

TABL	Record with the following fields:		
	TLPT	typeInteger	Top left cell in the range
	BRPT	typeInteger	Bottom Right cell of the range
	VAL	typeEnum	Return type (should be “TEXT”)

The code to create and send such a record would look something like this:

```
` this is how to send a request to Claris® Resolve™
` create the request range table
$Err:=CreateAERec($tabl)
$Err:=PutLongParam($tabl;"TLPT";"";100)
$Err:=PutLongParam($tabl;"BRPT";"";200)
$Err:=PutKeyword($tabl;"RTRN";"TEXT")
` create the record which holds the table
$Err:=CreateAERec($reqList)
$Err:=PutAERecord($reqlist;"TABL";$tabl)
` create the actual AppleEvent and add the record
$Err:=CreateAppleEvent("CLRS";"GVAL";$Resolve;$GetValue)
$Err:=PutAERecord($GetValue;"----";$reqList)
$Err:=SendAppleEvent($GetValue;$Reply;kAEWaitReply;-1)
` extract the data from the reply record
$Err:=GetAERecord($Reply;"----";$reqList)
$Err:=GetAERecord($reqList;"TABL";$tabl)
$Err:=GetTextParam($tabl;"VAL ";$resultString)
` clean up everything we allocated
$Err:=DisposeDesc($tabl)
$Err:=DisposeDesc($reqList)
$Err:=DisposeDesc($Reply)
$Err:=DisposeDesc($GetValue)
```

Interfacing with Microsoft® Excel 4.0

Excel 4.0 supports most of the AppleEvent Registry (See Appendix D) and also allows any command or macro to be executed with the DoScript command.The most useful events are Set Data, Get Data, and Create Element. You can use these events to create new worksheets or charts, fill in a range of cells, extract the contents of a range of cells, produce various kinds of charts, and copy a chart into 4D as a picture field.

Example 1: Filling a range of cells from a 4D array[1]

```
$err:=CreateAEVT("core";"setd";Excel;$aevt)
if ($err=0)
      $err:=PutObject($aevt;"----"ObjNamed("crng";0;"R1C1:R10C1"))
      $err:=PutList($aevt;"data";ArrayOfValues)
      $err:=SendAppleEvent($aevt;$reply;kAEWaitReply+CanInteract;-1)
      $err:=DisposeDesc($reply)
      $err:=DisposeDesc($aevt)
end if
```

Example 2: Creating a chart

```
$err:=DoScript(Excel;"Select("+char(34)+"R1C1:R10C2"+char(34)+")")
$err:=CreateAEVT("core";"crel";Excel;$aevt)
if ($err=0)
      $err:=PutLongParam($aevt;"kocl";"type";Long("chrt"))
      $err:=SendAppleEvent($aevt;$reply;kAEWaitReply+CanInteract;-1)
      $err:=DisposeDesc($reply)
      $err:=DisposeDesc($aevt)
end if
```

Example 3: Copying chart to 4D

```
$err:=CreateAEVT("core";"getd";Excel;$aevt)
if ($err=0)
      $err:=PutObject($aevt;"----";Obj("chrt";0;1))
      $err:=PutLongParam($aevt;"rtyp";"type";Long("SPIC"))
      $err:=SendAppleEvent($aevt;$reply;kAEWaitReply+CanInteract;-1)
      if ($err=0)
            $err:=GetPicParam($reply;"----";1;aPicture)
      End if
      $err:=DisposeDesc($reply)
      $err:=DisposeDesc($aevt)
End if
```

[1]These examples assume the variable 'Excel' was previously set to the target ID for Excel 4.0. Error handling not shown.

Using the Low-Level Interface

DON'T READ THIS SECTION IF *INSIDE MACINTOSH* FRIGHTENS YOU!!!!!!!

System 7 Pack's low-level interface gives you total control for sending and receiving more complex AppleEvents. These commands are very similar to the functions documented in the AppleEvent Manager chapter of *Inside Macintosh*™ Volume 6.

Sending an AppleEvent using the low-level interface involves creating the event, adding parameters to it, sending it, and finally disposing of the event and the reply.

To create an AppleEvent, call **CreateAEVT** specifying the target address, a 4-character event class, and a 4-character event ID. To add parameters to the AppleEvent, call **PutShortParam**, **PutLongParam**, **PutTextParam**, **PutAliasParam**, **PutObject**, **PutPicParam**, or **PutList**, depending on the data type to be used.

The PutObject command allows you to include an *object specifier* in your AppleEvent, which allows you to tell the receiving application to act on one specific piece of data such as a particular window, a word or paragraph in a particular document, or an object in a drawing. Object specifiers may be nested to represented objects contained in another object (called a container). For example, an object specifier might be something like 'the first character of the second word of the fifth paragraph in the document named "gone with the wind"'.

Objects are always identified by a 4-letter class ID and either a numeric value or a name and an optional container, which will be 0 (null) for the top-level object. For example, a paragraph may be contained in a document or window, but a window or document has no container other than the application itself, so their container would be 0. An object also has certain **properties**, also identified by 4-letter words. Typical properties would be the font, font size, style, and color. See Appendix D for a list of common object classes and property IDs.

Receiving an AppleEvent using the low-level commands is much easier, since most of the work is already done for you by the AppleEvent manager. When an AppleEvent is received, the procedure that was installed with **HandleAEVT** will be called. It can use a combination of low-level and high-level functions to obtain data from the AppleEvent.

When receiving anything more complex than a single text or numeric direct object, you should use the commands **GetShortParam**, **GetLongParam**, **GetTextParam**, **GetAliasParam**, **GetPicParam**, or **GetList** to extract data from the event. Note that there is no way to receive an object specifier, since 4D doesn't provide a way to resolve them into references to actual data.

Glossary

AERecord: A single descriptor within an AppleEvent which contains a list of descriptors specified by keywords. Can contain nested AERecords.

Alias: A reference to a file sent as part of an AppleEvent.

AppleEvent: 1. A message sent between two applications running under System 7 or from an application to itself either requesting information or telling it to do something. 2. The data structure used to represent an AppleEvent message, which consists of a list of data items (“descriptors”).

Container: An object which contains other objects; part of an object specifier. Example: in ‘word 1 of document named “Sample”’, the container is ‘document named “sample”’. Containers can be nested many levels deep.

Descriptor: A single piece of data within an AppleEvent or AERecord, identified by a 4-character key.

Descriptor List: A single descriptor within an AppleEvent or AERecord which consists of several items of the same type, refered to by position rather than by keyword.

Direct Object: The AppleEvent parameter identified by the 4-character key “----”. In many AppleEvents, this will be the only parameter.

IAC: Apple Computer's Interapplication Communication, a method which allows different applications on the same machine or across the network to communicate with each other.

Keyword: A 4-character identifier used to specify a particular descriptor in an AppleEvent or AERecord.

Object Specifier: An AppleEvent parameter which tells the application which piece of data the command applies to, such as ‘the first word of the second paragraph of the document named “gone with the wind”’.

Parameter: A single piece of data contained in an AppleEvent.

PPC Browser: Program-to-Program Communication (part of the Macintosh® Operating System) Browser. This is the standard user interface for selecting a program to link to or receive AppleEvents.

Script: A series of executable commands which take advantage of the host application's environment.

Signature: A unique four-character code which identifies a particular application.

Tick: One sixtieth (1/60th) of one second.

Time-Out: (n.) The period of time within which some expected action, event, or result must be initiated, finished, or occur. (v.) To cause a wait-period limit error to occur.

Appendix A: Built-in Event Handling

System 7 Pack™ has built-in handlers for the following AppleEvents:

'misc','dosc'	This is the standard 'do script' command. It will execute any 4D command sent with it. An optional parameter 'ACK0', which should be 1 or 0 specifies whether an acknowledgment should be sent after the command executes.
'ISIS','CFIL'	Count Files - returns the number of files.
'ISIS','CFLD'	Count Fields - returns the number of fields in the file specified by the direct object which must be numeric (either long or short integer).
'ISIS','FNAM'	File Name - returns the name of the file specified by the direct object which must be numeric.
'ISIS','FINF'	Field Info - returns the name and type of a field. The file number is specified by the direct object and the field number is given by the key 'FNUM'. Both of these must be numeric. The field name is returned as the direct object of the reply and the field type (an integer as returned by 4D's "type" function) is returned with the key 'FTYP'.
'ISIS','RECS'	Returns the number of records in the selection of the file specified by the direct object.
'ISIS','RECN'	Returns the current record number of the file specified by the direct object.
'ISIS','GETF'	Returns the contents of a field in the current record. The direct object specifies the file number and the field number is given by the key 'FNUM'.
'ISIS', 'ACK0'	This is optionally sent by 4D® as an acknowledgment after a 'do script' command is completed. The direct object is a numeric result code with any 4D® error that occurred when executing the command.

All of these events will return a descriptive error message[2*] in the 'errs' parameter if any errors occur. The reply will be one of the following: "Bad File Number", "Bad Field Number", "Bad Record Number", or "No Current Record". In order to accept these AppleEvents, you must execute the command **AllowAccess(1)**.

[2*] These messages are in the 'STR#' resource named "Errors" and may be translated.

Appendix B: Sample Programs

Your System 7 Pack™ disk includes several demo programs which illustrate how to use most of the features.

Reference is System 7 Pack's on-line reference database. It includes a complete listing of all commands in System 7 Pack in a HyperCard-like format. This also includes demos of most of System 7 Pack's features. There are also several useful 4D procedures which you can copy to your own databases. See Appendix D for a description of several functions which create and send some of the core AppleEvents. Here's a brief description of the menu commands:

- **List Commands**: Displays a list of System 7 Pack's commands. Double-clicking on a command name will open up a HyperCard-like information window.
- **Export to Word**: Creates a list of System 7 Pack's commands as a formatted MS Word file and then uses System 7 Pack to open the document in Word.
- **Frontier Demo**: Sends a few simple commands to UserLand Frontier™, which must be already running.
- **Finder Demo**: Demonstrates how to send AppleEvents to the Finder.
- **Microphone 4.0**: Illustrates how to use the low-level commands and create object specifiers by launching Microphone 4.0, setting some variables using DoScript, and sending Get Data events to examine the contents of Microphone's variables.
- **Excel 4.0:** Demonstrates how to send data to Excel 4.0, request different types of charts in Excel, and convert Excel charts to a picture in 4D.

- **Send & Recv List**: Demonstrates how to send and receive arrays using the low-level functions. For the receiver, you must select either your own copy of 4D or another one on the network running this same demo.
- **Net Chat**: Demonstrates how to use the high-level functions to send a simple text message. For the receiver, you must select either your own copy of 4D or another one on the network running this same demo.

Address Server and Address Client demonstrates one way 4D® can provide data to a client application. The 4D application provides a minimal address file which responds to AppleEvents (class "ISIS" id "FIND") that request a name lookup. Any text passed via the lookup request will trigger a Search on the name field of the address file. The full address will be returned in an event of class "ISIS" and id "REPL". Address Client is a tiny (28k) stand-alone application which simply sends lookup requests to the 4D® application and displays the results. Address Client Stack is a Hypercard 2.1 stack which works exactly the same as the application. FM Pro Client is a sample client written in FileMaker Pro 2.0. This one sends a different request to 4D and gets back a series of Create Element events to create records containing the results.

Appendix C: Error Codes

All System 7 Pack™ functions will return 0 if the operation is completed successfully. Some applications may return codes not listed here if they're unable to handle an AppleEvent sent to them. Other possible results are:

-1.....An invalid signature or target address handle was given

-2.....You're not running System 7 or your system doesn't support AppleEvents

-3.....The specified application isn't running.

-4.....Incompatible array type (PutList or GetList).

-10.....Handler for that event is already present (InstallAEVT).

-20.....Attempted to replace standard event handler (InstallAEVT).

-30.....Attempted to remove event handler we didn't install (IgnoreAEVT).

-35.....Couldn't find application to launch on any volume.

-43.....Couldn't find document to open or print.

-100.....Unable to install AppleEvent handler proc (InstallAEVT).

-108.....Not enough memory to launch an application.

-606.....Attempt to bring background-only application to front. (BringToFront).

-906.....Attempt to send AppleEvent to non-aware application. (usually PrintDoc).

-1700.....Incompatible data type in an AppleEvent parameter.

-1701.....Parameter not found in the AppleEvent.

-1702-1707 Not a valid AppleEvent or invalid parameter.

-1708.....Receiving application couldn’t handle that AppleEvent.

-1709.....Reply wasn’t valid.

-1710.....Unknown send mode.

-1711.....Wait for reply cancelled by user.

-1712.....Timed out waiting for reply.

-1713.....Required user interaction but none was allowed.

-1715.....Required parameter wasn’t accessed.

-1716.....Invalid target address.

-1718.....Attempted to access reply which hasn’t arrived yet.

Appendix D: AppleEvent Registry

Fully describing all of the standard AppleEvents is beyond the scope of this manual, but here are some of the more common events defined by Apple in the required, core, and miscellaneous standard suites of the AppleEvent registry:

Name	Class	ID	Description
Open Application	aevt	oapp	Sent by the finder when an application is opened with no documents. You shouldn't send this event.
Open Document	aevt	odoc	Tells an application to open a list of documents.
Print Document	aevt	pdoc	Tells the application to open and print one or more documents.
Quit	aevt	quit	Tells the application to quit.
Close	core	clos	Closes the specified objects.
Delete	core	delo	Deletes the specified objects.
Do Objects Exist	core	doex	Determines if the specified objects exist.
Get Class Info	core	qobj	Get information about a particular object class.
Get Data	core	getd	Get data from the specified objects.
Get Data Size	core	dsiz	Get the size of specified objects.
Get Event Info	core	gtei	Get information about a particular AppleEvent.
Save	core	save	Save the specified objects.
Set Data	core	setd	Change the specified objects.
Do Script	misc	dosc	Executes commands in the applications specific language.
Evaluate	misc	eval	Evaluates an expression and returns the results.

Here some of the more common class names and property IDs:

Cell.....ccel
Column.....ccol
Document.....docu
File.....file
Graphic object.....cgob
Menu.....cmnu
Paragraph.....cpar
Row.....crow
Selection.....csel
Table.....ctbl

Window.....cwin
Word.....cwor
Best type (property) pbst
Bounds (property). .pbnd
Class (property).....pbnd
Color (property).....colr
Default type.....deft
Font (property).....font
Name (property).....pnam
Point Size.....ptsz
Version.....vers

The sample database, S7P Reference, includes the following functions which create and send some of the more common core events:

Err:=Create Element(Target;Class;Container;Position)

Creates a new element of the specified class. Specify 0 for container and "" for position if no value needs to be given.

Result:=Do ObjectsExist(Target;Object)

Returns TRUE if the specified objects exist.

Err:=Get Text(Target;Object;»data) Err:=Get Pict(Target;Object;type;»data) Err:=Get Array(Target;Object;type;»data)

Returns the value of an object or object property. Type should be a 4-letter string specifying the data type to be returned. It should be one of "TEXT", "LIST", "PICT", or "SPIC". "SPIC" is only used with Excel 4.0 and is the same as PICT except it will return a color screen picture rather than a dithered black & white print image picture. Data should be a pointer to a variable of the appropriate type.

NOTE: Rather than using a single procedure, I've provided separate procedures for the most common data types to make them compatible with the 4D compiler.

Err:=Send Text(Target;Object;Data) Err:=Send Array(Target;Object;»Array) Err:=Send Enum(Target;Object;Value)

Changes the value of an object or object property.

Examples:

Err:=Send Text(Excel;Property("sele";Obj("docu";0;1));"R1C1:R10C1")
Selects cells R1C1 thru R10C1 in the topmost worksheet.

Err:=Send Array(Excel;ObjNamed(“crng”;Obj(“docu”;0;1);”R1C4:R10C4”);»aList)
Sends an array of numbers to a range in the topmost worksheet.

Err:=Get Pict(Excel;Obj(“chrt”;Obj(“docu”;0;1);1);”SPIC”;»aChart)
Copies the first chart in the topmost worksheet to a color picture.

Err:=Create Data(Excel;”chrt”)
Creates a new chart document.

HasChart:=Do ObjectsExist(Excel;Obj(“chrt”;Obj(“docu”;0;1);1))
Returns TRUE if the topmost worksheet contains a chart.

For more information, see the latest edition of the AppleEvent Registry, available from Apple Computer, Inc.

Appendix E: Network Access

Before you can send AppleEvents across the network you must configure any machines you wish to be able to access. Any machine which needs to receive remote AppleEvents must have "**Program Linking**" turned on with the **Sharing Setup** control panel (see figure 1). In addition, any machine you wish to connect to should have an entry in the **Users & Groups** file for you (optionally, if security isn't a concern, you can simply turn on the program linking checkbox for guests by double-clicking the <Guest> icon in the **Users & Groups** control panel - see figure 2). The first time you send an AppleEvent to a program on a remote Macintosh you will be asked to suply a user name and password. If guest access is enabled, you can simply click on the "**Guest**" button.

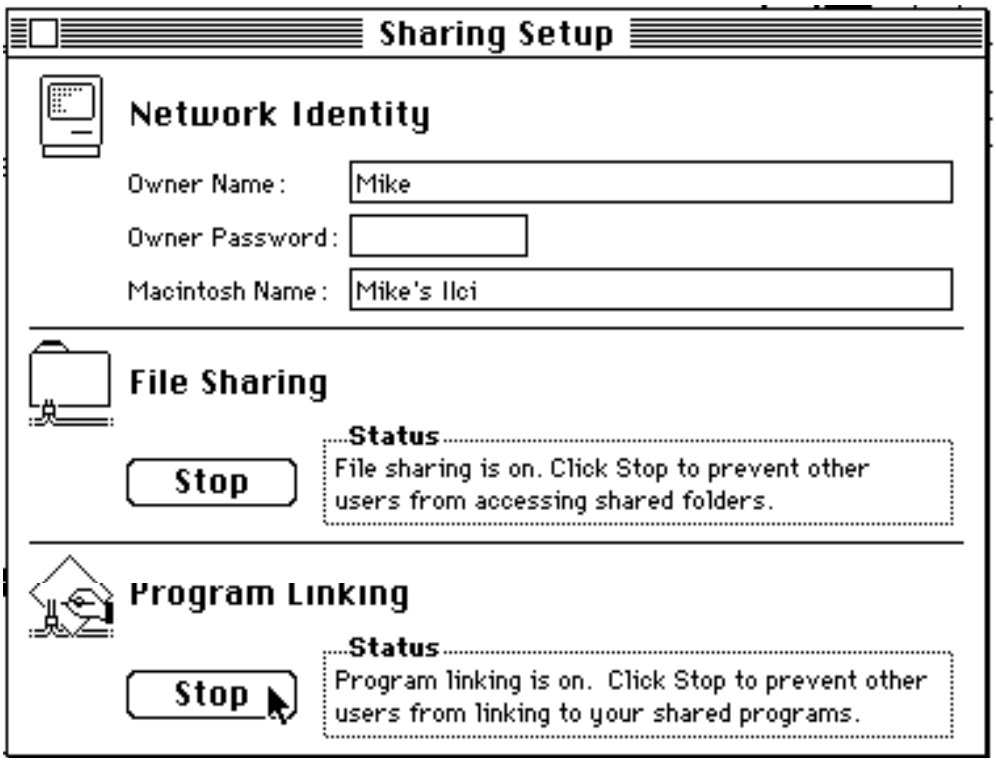

Figure 1: Turn on program linking

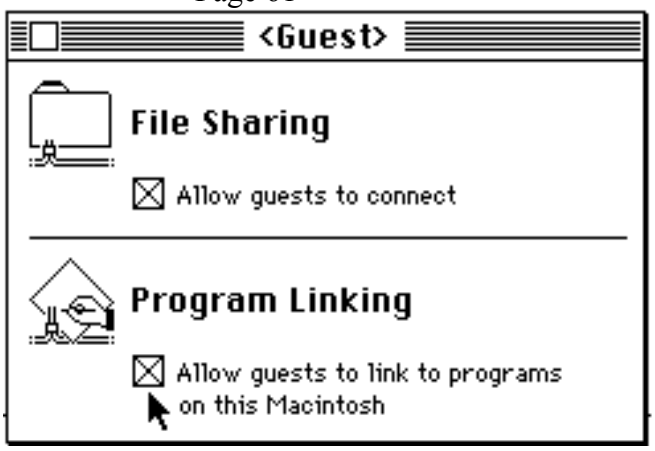

Figure 2: Allow program linking for guests

Appendix F: Quick Reference

L := AddrToString(Target;String1;String2;String3)
L := AE Process ID
L := AESend(Target;Class;ID;Text)
L := AESendPict(Target;Class;ID;Picture)
AllowAccess(N)
L := BringToFront(Signature)
L := CreateAERec(AERec[3])

L := CreateAEVT(Class;ID;Target;AEVT)
L := CreateXAEVT(Class;ID;Target;N;L;AEVT)
L := CopyDesc(AEVT **or** Target **or** OSPEC **or** AERec)
L := DisposeAddress(Target)
L := DisposeDesc(AEVT **or** Target **or** OSPEC **or** AERec)
L := DoScript(Target;Text)
L := Evaluate(Target;Text;Reply)
L := FindAppName(Signature;Name)
L := FindCreator(Name;Signature)
L := Frontier(Text;Reply)
L := GetAEInfo(AEVT[2,3];NumOfItems;ArrayOfStr4;ArrayOfStr4;ArrayOfLong)
GetAEMessage(Text)
L := GetAEPict(Picture Variable)
L := GetAERecord(AEVT[2,4],Str4;AERec)
GetAEType(Type String)
L := GetAliasParam(AEVT[2,3] ;Str4;String)
L := GetKeyword(AEVT[3];Str4;Str4)
L := GetList(AEVT[3];Str4;anyArray)
L := GetNthDesc(AEVT[3];integer;Str4;Str4;Long)
L := GetNthItem(AEVT[3];integer;Str4;Str4;Text)
L := GetRealParam(AEVT[3];Str4;aRealNum)
L := GetReturnID(AEVT[3])
L := GetShortParam(AEVT[3];Str4;Str4;L)
L := GetTransactionID(AEVT[3])
L := GetPicParam(AEVT[3];Str4;N;aPicture)
GetReturnAddr(Target)
L := GetShortParam(AEVT[3];Str4;Str4;N)
L := GetTextParam(AEVT[3];Str4;Text)
L := HandleAEVT(Class;ID;Name)
L := IgnoreAEVT(Class;ID)
L := IsRunning(Signature)
L := Launch(Signature;Name)
L := LaunchBehind(Signature;Name)
L := Long(Str4)
L := MakeAddress(Signature;Target)

[3]Any function which adds data to or extracts data from an AppleEvent can also take an AERecord.
[4]Any function which extracts data from an AppleEvent can be passed 0 to access the event currently being processed in a handler procedure

ospec := Obj(Str4;ospec[5];L)

[5]Object specifiers passed to these functions will be automatically disposed, to simplify creating nested containers

ospec := ObjNamed(Str4;ospec[3];String)
L := PrintDoc(Signature;Name)
ProcessAEVT
L := ProcessList(N;ArrayOfString4)
ospec := Property(Str4;ospec[3])
L := PutAERecord(AEVT;Str4;AERec[6])
L := PutAliasParam(AEVT;Str4;Name)
L := PutKeyword(AEVT;Str4;Str4)
L := PutList(AEVT;Str4;Str4;anyArray)
L := PutLongParam(AEVT;Str4;Str4;L)
L := PutObject(AEVT;Str4;ospec[3])
L := PutPicParam(AEVT;Str4;N;aPicture)
L := PutRealParam(AEVT;Str4;aRealNum)
L := PutShortParam(AEVT;Str4;Str4;N)
L := PutTextParam(AEVT;Str4;Text)
L := SendAppleEvent(AEVT;ReplyAEVT;L;L)
L := QuicKeys(Name)
L := QuitApp(Signature)
L := S7Version(Name)
L := SelectAddress(Name,Signature;Target)
L := SendWithReply(Target;Class;ID;Text;Reply)
SetTimeOut(N)
L := StringToAddress(String1;String2;String3;Target)
L := System7

L:	**a Long Integer value**
N:	**an Integer value**
Name:	**a string variable or field**
Signature:	**4 character application signature**
Class,ID:	**4 character strings**
Text:	**a Text variable or field**
Reply:	**a Text variable or field**
AEVT:	**a Long Integer representing an AppleEvent**
AERec:	**a Long Integer representing an AERecord**
ospec:	**a Long Integer representing an Object Specifier**
Target:	**a Long Integer representing a target address**
ospec:	**an object specifier**

[6]The AERecord passed to this function will automatically be disposed.

Appendix G: Version History

Version 1.0 - Dec. 1991

Initial release.

Version 2.0 - Feb. 1992

Added Frontier & QuicKeys support, Built-in AppleEvent handlers.

Many commands changed from version 1.0

Version 3.0 - May 1992

Added low-level commands.

Many new commands added but compatibility maintained with 2.0

Procedure Editor now lists commands in logical groupings.

Demo programs completely rewritten.

Versoion 3.1 - June 1992

Added AE Record support, needed for Claris Resolve.

Version 3.1.1 - June 1992

Fixed problem with StringToAddress

Version 3.2 - July 1992

Added Long() utility function & demos for Excel 4.0

Version 3.26 - Sep. 1992

Added LaunchBehind command.

Added many new commands for examining AppleEvents and provide more access to internal features for custom extensions to S7P.

*** Interim release until 4D v3.0 compatibility can be verified ***

Version 3.3 - Nov. 1992

Insure compatibility with 4D® Version 3.0.

About ISIS International

ISIS International was founded in 1987 and is recognized as one of the top 4th DIMENSION® developers. Our clients include a major hospital, film production companies, and the state of California. In addition to System 7 Pack™, we produce complete business solutions. Our other products include ISIS Notes™, an AppleEvent aware network messaging & file transfer utility, Antwerp™, a powerful invoice, inventory, & shipping management system; Horus™, a suite of sales, tech support, and customer service tools; ISIS Medical & Dental office systems; the ISIS Construction, Architrieve, and GraphicBid series, which features cost estimation, bidding and scheduling; and ISIS Production Office which allows a small production company to juggle multiple projects in various stages. We also sell Foundation™, our proprietary 4th DIMENSION® application shell which we used to develop all of our business products.

Tech Support

ISIS can be reached at (818) 788-4747 during West Coast business hours. We have the number forwarded to our respective homes after hours, so we can be reached at most times. We also provide support on America Online in the Mac Business forum (enter the keyword “ISIS”). We can also be reached on AppleLink at D6734. Our Compuserve IDs are 76367,1406 (Paul Harwitz) and 70375,350 (Mike Cohen).