The History behind Object Tools

Object Tools is a 4D plug-in from Automated Solutions Group that allows a developer to create “objects”. Objects are used to store 4D data items that can be retrieved on a random access basis by referencing an item by its “tag”.

Aparajita Fishman developed Object Tools in 1991 when he was working for Natural Intelligence, a consulting company located in Cambridge, Massachusetts. Object Tools was originally written for a mission-critical securities trading system that Aparajita was developing for a customer on Wall Street. Object Tools allowed Aparajita to supplement 4D’s procedural language with object-oriented programming techniques. Object Tools was matured during the 1990’s when it was updated for use in other 4D applications that were developed by Natural Intelligence.

After Aparajita joined Automated Solutions Group in 1999, he and Michael Erickson, President of Automated Solutions Group, decided to make Object Tools available to the 4D world as a commercial product.

When Object Tools was released in July, 1999, I read the press release but did not take the time to download the demo version and spend time working with it. I have used custom data types since version 3 of 4th Dimension and I didn’t think that Object Tools was all that different from using 4D BLOBs. I very wrong about that!

Soon after beginning to use Object Tools, I concluded that I could use Object Tools to dramatically improve many facets of the work that I do in 4D.

A Quick Overview

When an Object is created, the “OT New” command returns a unique long integer value called a “handle” that is used to reference the Object. The handle is used by almost all of the Object Tools commands and it represents the memory address of the object. Object Tools does not store data as process or interprocess memory. Similar to other plug-ins, data stored in an Object is maintained in the application memory heap.

Object Tools is easy to work with. The first thing that I did with Object Tools was to try something very simple. Even though I’ve been

using BLOBs and PictBundle to store data, I tried out Object Tools by filling a few arrays with data, putting the arrays into an Object, clearing the arrays, and then loading the arrays from the Object. With that simple experience under my belt, I used the documentation as a guide for the more advanced commands. Within a very short time of putting Object Tools through its paces, I felt comfortable working with its extensive command set.

The Object Tools manual groups the commands under these headings - Creation and Destruction, Putting Values, Getting Values, Object Info, Item Info, Item Utilities, Import and Export, and Miscellaneous Utilities.

The “Creation and Destruction” commands include OT New, OT Clear, OT Copy and OT ClearAll. The OT New command returns the long integer memory address (the “handle”) that is used in all of the commands that reference a specific Object. OT Clear destroys the Object by removing it from memory. OT Clear All should not be used in a production system since it deletes *all* OT objects.

The “Putting Values” commands are used to assign data to an Object. All “put” commands for variables and fields follow the syntax of OT Put+<VarType>. For example, the command OT PutReal is used to assign a real value while OT PutString is used to assign a string.

Most of 4D’s 12 data types can be assigned to an Object. The exceptions are integer, time, Boolean, and graph data. I don’t feel that this is a significant limitation. Recall that 4D does not support integers in compiled applications and, because there are no time arrays in 4D, we have to use long integer arrays. Boolean data can be supported by the values 1 or 0 in long integer arrays. Object Tools works with BLOBs, Object Tools Objects, pointers, and pictures. Arrays of all types are assigned to an object using one command OT PutArray.

The “Getting Values” commands are used to retrieve data from Objects. The “get” commands that return single values are called as functions, returning the value to a field, variable, a dereferenced pointer, or an array element.

When retrieving entire arrays, the array is passed as the third parameter. Sample code is shown below.

These command put a long integer and an array into the Object $HandleRecData:

```
OT PutLong ($HandleRecData;"$PrimaryKey_L";$PrimaryKey_L)

OT PutArray ($HandleRecData;"$WWW_ItemName_AT";$WWW_ItemName_AT)
```

The data is retrieved using this code:

```
$PrimaryKey_L:=OT GetLong ($HandleRecData;"$PrimaryKey_L")

OT GetArray ($HandleRecData;"$WWW_ItemName_AT";$WWW_ItemName_AT)
```

There is an alternative means of accessing array data that is stored in an Object. The command “OT GetArray”+<Array Type> is used to retrieve a specific array element from an Object. The code needed to retrieve the third element of the array COMP_Name_AS is:

```
$CompanyName_S:= OT GetArrayString($HandleCompanyData;"COMP_Name_AS";3)
```

This command allows you to create arrays in one process and have that data handled in another process without ever populating the arrays in the second process – very cool!

The Object Info commands include “OT IsObject”, “OT ObjectSize”, and “OT ItemCount”. “OT ObjectSize” is helpful during programming since it indicates how much data has been put in the Object. The item count returned by OT ItemCount is needed for code that deals with a varying number of values. This command is notably absent from the 4D BLOB command suite.

The “Item Info” commands are more involved than the Object Info command. “OT ItemExists” searches the object for an item of a specified tag; “OT GetAllProperties” returns the name, type, data size, and item size attributes of the Object contents while “OT GetItemProperties” returns those same attributes for the item specified by the tag.

The commands “OT DeleteItem” and “OT CopyItem” are Utility routines. The latter command copies an item from one Object to another Object.

The “Import and Export” commands deal with BLOBs. “OT BLOBToObject” and “OT ObjectToBLOB” are used to convert an Object to a BLOB and vice versa.

The “Miscellaneous Utilities” include “OT GetHandleList” which returns a list of all Object Tools handles. The command “OT SetErrorHandler” has a very nice touch - in addition to allowing the developer to set their own error handler, it returns the name of the current error handler. This is used to reset a custom error handler when the Object Tools error handler is no longer needed.

What about PictBundle?

PictBundle was a 4D plug in that was released in 1993 and was popular with 4D version 3. It was a stable, well supported, and inexpensive product that allowed a developer create a “bundle” and to store and retrieve all of 4D’s data types.

Bundles were declared as picture variables and, similar to BLOBs in 4D version 6, variables were needed to read data into and retrieve data from a bundle in a sequential manner. There were no catalog commands and PictBundle didn’t inherently support using a tag to access data.

4D BLOBs

With the release of 4D version 6, 4D introduced support for the Binary Large OBject data type, more commonly known by the acronym “BLOB”. In addition to introducing the BLOB as a field, 4D now supports BLOB variables. Unfortunately, 4D does not offer support for BLOB arrays.

There are just under two dozen BLOB commands. In addition to the more common Variable to BLOB and BLOB to Variable, data can be passed to and from BLOBs using commands such as INTEGER TO BLOB, TEXT TO BLOB, LONGINT TO BLOB, etc. Data is retrieved from a BLOB using the inverse of those commands - BLOB to integer, BLOB to longint, BLOB to text, for example.

Variable to BLOB and BLOB to Variable read and write data sequentially, while the more advanced commands allow the developer to read and write data at a location within the BLOB. The BLOB commands do not use tags and pointers or arrays of pointers cannot be stored in a BLOB.

Object Tools

Object Tools has an extremely strong command set and has a few key features that are valuable in overcoming the limitations of 4D BLOBs.

Storing and retrieving data using a “tag” is an immensely powerful feature in Object Tools. A “tag” is a string that is best described as the unique name of the item in an Object. In addition to using a tag to *reference* a value within an object, that same tag can be used to *describe* the data. This flexibility allows you to store the contents of a [People]First_Name field using the tag "First_Name" or you can store a pointer to [People]First_Name as "Ptr_To_FirstName". As will be discussed below, one powerful tag-naming technique is to create a tag that consists of the table number and field number of the field. This value, which I call a “Field ID” can be used to identify data in Objects that are being passed between different computers or different 4D systems.

Data in an Object is “private” data since the native 4D language cannot be used to access an Object. Passing Object handles as local variables is a sure-fire way to eliminate the extreme cost and great angst that invariably results when data is passed using process and interprocess variables.

Creating and populating an Object and then passing the Object handle to a subroutine will eliminate long parameter lists. The Object is referenced by the handle in the subroutine and data can be selectively extracted. Wise use of Object Tools will put an end to this type of code:

```
If (Count parameters>=5)
   `..Do something wonderful here
End if
If (Count parameters>=6)
   `..Do something wonderful here
End if
If (Count parameters>=7)
   `..Do something wonderful here
End if
```

Another strength of Object Tools is that it supports pointers and arrays of pointers. When used in conjunction with the tag feature, you can create Objects that allow you to reference an array based on a field pointer or vice versa.

Referencing array elements in an Object means that you're not forced to retrieve an entire array from an Object just to get at the data in the array. A technique that takes advantage of this feature is demonstrated in the "OT Code" database on the example disk.

Storing Preferences

A simple use of using Object Tools is to store and retrieve preferences. The headaches of reading from a text file were eliminated with the advent of the BLOB field in version 6. Despite that improvement, a lot of code is needed when dealing with a large number of preferences since there is no way to easily read and write BLOB data on a random access basis.

If you are working with an application that has only a few preference settings, Variable to BLOB and BLOB to variable may be well suited to the task. Problems can arise when dealing with dozens of preferences or when you want to load and store preference on a selective basis. More data items in a BLOB require more code which means more maintenance.

One application some readers may be familiar with is "DADS". One of the functions of DADS is to reverse engineer 4D Server and ODBC-compliant applications so that the user can create new application. Preferences in DADS are stored in one record consisting of 27 fields as well as one BLOB field that contains 17 preference values. Since there is no easy way to access BLOB data on a random basis, DADS reads all of the preferences into interprocess variables at startup and, when the user quits, writes all of those variables back to the BLOB.

This code can be greatly simplified by storing preferences in an Object that is stored as a BLOB in the data file. Thanks to Object Tools, the code in DADS can be simplified to read and write data from preferences Objects on an as-needed basis.

Sending a Record to Another Process

One of the issues that we face in developing databases is the conflict between speed in creating and updating records versus the need to access records quickly. In the case of creating and updating records, performance is improved by *reducing* the number of indexes. In contrast, performance of retrieving and manipulating records is improved by *increasing* the number of indexes. Object Tools can help solve this problem.

A “typical” 4D application, uses a detail form for data entry, calls “Save record”, and then calls “Add record” to create another record. Unfortunately, as the load on the system increases and as the number of records increases, the time needed to save a record may become significant.

The delay that a user experiences using this approach can be almost completely eliminated by a using a simple technique that takes full advantage of 4D’s powerful multi-process capability. Data from the detail form is put into an Object in the main process and the handle to the Object is passed to a paused process. The fields on the detail form are cleared and the user can start entering data for another record. This “clear the screen but don’t close the window” technique has been used for years to permit data entry from a dialog and there’s no particular reason *not* to use it when entering data using records. The code for this is straightforward and code maintenance is eliminated by using the “Field ID” as a tag value.

A “Field ID” is a string value that uniquely identifies a field in a database. It created by concatenating the string values of the field and the table numbers. In the example database that accompanies this article, the [People] table is the third table and [People]First_Name is the fourth field in the table. The Field ID for [People]First_Name is “00030004”. Field ID is calculated by the function “UTIL_FieldIDCreate”. An eight-digit field ID will support up to 9999 tables and 9999 fields, leaving plenty of room for growth!

When data from a field is loaded into an Object, the Field ID is passed as the tag for that item. Conversely, when data is retrieved from an Object, the Field ID can be converted to a pointer. When the field pointer is dereferenced, that pointer can be used to assign the contents of the Object item to a field. This technique is very fast, very simple to code, and, most of all, it eliminates virtually all bugs!

How can you combine the flexibility of Object Tools with a Field ID to speed up data entry? Here's a description of what's involved:

> As a detail form loads, create an Object with the handle PERS_FieldPtrHandle. Fill the Object at PERS_FieldPtrHandle with pointers to the fields that are on the form.
>
> Call an object method from the "OK" button (it's a "No Action" button).
>
> Code the object method for the OK button so that when the User clicks on the button, 4D creates an Object Tools Object with the handle $PERS_RecDataHandle.
>
> Loop through PERS_FieldPtrHandle. For each entry in PERS_FieldPtrHandle, get the field pointer that is stored in the Object, determine the field type, and use the appropriate "OT Put" command to pass data from the field pointer to the Object.
>
> Once the Object has been filled, send a message to our handy paused process "Gopher".
>
> Gopher will open the Object, create a record, read each tag, and convert each Field ID to a pointer. As the field pointer is retrieved, dereference the field pointer to assign the data from the item in the Object to the field in the record.

Refer to the code in the Object Tools sample database and trace through the code, starting with the [People]Input form.

It is very safe to add this technique to an existing database because it's not an "all or nothing" technique. Use this technique only in the places where it suits your need.

Sending a Record to Another Computer

Distributed database applications require that data is sent from one computer to another. An application that uses distributed processing might pass data from a 4D Server application to a 4D application or even to another 4D Server system. 4D provides the tools needed to architect a sophisticated system and Object Tools makes it easy to move the data.

Record data is stored in an Object either as an individual record or as a selection of records in that have been loaded into arrays. The Object is converted to a BLOB and then sent to another machine on the LAN or a WAN (keeping in mind that the Internet is this planet's largest WAN). BLOBs are sent from 4D to 4D or from 4D to 4D Client using Internet Toolkit while 4D Open is used to send records from 4D or 4D Client to 4D Server.

Sending data between computers and between applications requires that you store information that can identify an Object when it is received. This information, referred to as “header information”, not only reduces programming errors but it is vital in ensuring backward compatibility. In addition to storing Field ID’s and field data in an Object, the header information should include the username, a password or access code, the application name, the application version, and the date time stamp when the Object was created.

When a server application receives a BLOB, it converts the BLOB back to an Object and examines the header information. After validating the BLOB, data is extracted from the Object and is used to create or update records.

The command “OT GetArray”+<Array Type> makes it easy to move records between computers. The code for this technique is in the sample code database in the method “PPL_ArraysToObjToRecsDemo”. Please refer to that code and trace through it to see how the OT GetArray command is used to assigns data from an array element directly to a field - no variables needed!

Object Tools is Incredibly Versatile Product

In the decade that I've been working with 4D, I've had the opportunity to use almost every command in the 4D language and have made a conscious effort to investigate plug-ins when they are brought into the marketplace. With that as a background, I rank Object Tools as being as much of a “must have” product as the venerable AreaList Pro.

Both AreaList Pro and Object Tools are excellent products but they differ in the scope of their impact. AreaList Pro is a grid control that allows the developer to display data from fields and arrays. This categorizes AreaList Pro as an interface enhancement that improves the user experience. In contrast, Object Tools is used behind the scenes. It is a programmer’s tool. And it is an excellent product.

Object Tools adds flexibility to how 4D systems are architected, it allows you to improve how your code is designed and programmed, and can significantly reduce bugs in 4D code. Object Tools is an incredibly versatile product that should be added to every developer’s toolkit.

Download Information

A demo version of Object Tools can be downloaded from the Automated Solutions Group website at http://www.asgsoft.com.