REFERENCE MANUAL

ObjectTools

The ultimate 4D language extension

Product and documentation by Aparajita Fishman

TABLE OF CONTENTS

CHAPTER 1

Introduction

ObjectTools is a 4th Dimension plug-in which provides a set of routines that allow you to create *objects*: a single entity in which you can store and retrieve any amount of data of differing types.

While similar to 4D BLOBs and other plug-ins, ObjectTools has several important advantages:

- **Objects store data as named items:** Finally you are freed from the drudgery of using numeric offsets to store and retrieve data. With objects, you store and retrieve data as distinct items using a Unicode name.
- **Objects are random access:** Whereas in practical terms BLOBs must be written and read in the same order, with objects you can store and retrieve data items in any order.
- **Objects are modifiable:** You can replace, delete or copy an existing data item without recreating the entire object.
- **Objects can be stored in arrays:** Because objects are represented by a Longint handle, you can create arrays of objects. This ability makes objects the perfect tool for interprocess messaging.
- **Objects can store and retrieve complete records with one call:** This allows you to implement a kind of record-level undo.
- **Arrays within objects are directly accessible:** Once stored in an object, you can get the size of an array and directly access any given element, allowing you to iterate over an array within an object.
- **Objects can be embedded in objects:** You can store objects within objects and directly access embedded items, thus allowing you to easily model complex hierarchical data structures.
- **Objects reveal their structure:** ObjectTools has a full suite of routines that let you know everything about the structure of an object. In fact, ObjectTools ships with a sophisticated visual object editor that uses these routines to create, examine and modify the contents of any object.

How Can Objects Help Me?

While the uses of objects are virtually limitless, there are several common problems which they solve.

- They can be used to easily save and restore complex configuration data such as preferences.

- They can drastically reduce the use of process and interprocess variables by allowing you to place related data in one object instead of numerous variables.
- They can be used to save and restore entire records with one call.
- They can be used to store hierarchically structured data.
- They can facilitate an object-oriented style of programming.

How Do Objects Work?

In classical programming terms objects are implemented as an unordered dictionary.

A dictionary (also known as a *map* or an *associative array*) is a collection of key-value pairs, where the key uniquely identifies a value. In the case of objects, the key is the item reference, or tag. The value is whatever data was stored with the key.

Objects are unordered dictionaries, meaning that the internal order of the key-value pairs at any given time is indeterminate.

Registering ObjectTools

When you purchase ObjectTools you will receive a serial number. The serial number must be passed to the **OT Register** command in order to register your copy of the plugin. If **OT Register** is not called or is called with an incorrect serial number, ObjectTools will timeout after 15 minutes of use. Once ObjectTools has timed out, the next call to ObjectTools will cause an ObjectTools error to be generated, and subsequent calls will have no effect or return empty values if a values are expected.

System Requirements

ObjectTools 4 has the following minimum requirements:

- 4D v11.5+
- Mac OS X 10.6.8 running on Intel, or Windows XP SP2+/2000/Vista/7

If ObjectTools is loaded on a version of 4D less than v11.5, it will become inactive.

Resource Files

Within the ObjectTools plugin bundle are resource files used by ObjectTools. These resource files are located in ObjectTools.bundle/Contents/icu.framework/Versions/A/Resources. The resource files are:

- **ObjectTools_56l.dat:** Contains resources specific to ObjectTools.
- **icudt56l.dat:** Contains resources used by ICU, a code library used by ObjectTools. This is a very large file because it contains Unicode and internationalization data for every country and language in the world.

The default location for these files is within the plugin bundle. They may also be placed in the *<shared 4D folder>/com.aparajita/icu* folder. The shared 4D folder is the parent of the folder which is returned by **Get 4D folder(Licenses folder)** within 4D. For the location of this folder, please refer to the 4D documentation for the **Get 4D folder** command.

If you decided to use the shared "icu" folder, *both* resource files must be placed there.

What's New In Version 5

If you are upgrading from ObjectTools 2.5, the key difference is that this version runs as a native 4D v11+ plugin. This means you now have the following enhancements:

If you are upgrading from ObjectTools 4, you now have:

- Full 64-bit support on OS X.
- Support for 4D v15r2+.
- More information in the error handler.

If you are upgrading from ObjectTools 3, you now have:

- Full 64-bit support on Windows.
- Support for Active4D v6's ObjectTools interchange.

If you are upgrading from ObjectTools 2.x:

- There is now full support for Unicode text throughout ObjectTools.
- There is full support for the enhanced picture format used in 4D v11+.
- Depending on your usage, ObjectTools 4 may run significantly faster in a Unicode mode database than version 2.5.

CHAPTER 2

Working with Objects

To effectively use ObjectTools objects, you need to learn a few simple concepts about their creation and destruction.

Creating and Destroying Objects

Like 4D hierarchical lists, objects are represented by a *Longint* known as the *object handle*, or simply *handle*. You create a new object with the *OT New* method, like this:

```
C_LONGINT($object)
$object:=OT New
```

Once you have an object handle, you can then proceed to put values into and get values from the object, query the object for information about its structure, copy it to another object, and put it into a BLOB.

The data stored in an object takes up a certain amount of memory within your application. When you are completely finished with an object, it is critical that you release the object's memory by calling *OT Clear*, like this:

```
OT Clear($object:)  `$object will be set to zero by OT Clear
```

Once you have cleared an object with *OT Clear*, you will no longer be able to use its handle.

Memory Management with Objects

The memory used by objects is global to the memory space of a *single* instance of 4D (Standalone, Client/Remote, or Server). This means that a single object may be shared between all 4D processes within a single instance of 4D, no matter in which 4D process it was created. On the other hand, a single object may not be shared between separate instances of 4D, either on the same machine or on different machines. This includes Client/Remote and Server.

If you lose track of an object handle without clearing the object — either by storing a handle in a local variable and leaving the method in which it was created, or by using a process variable and leaving the process in which it was created — the memory occupied by the object will remain and you will not be able to clear it. This is known as a *leak*, and it is a Bad Thing, especially if this process is repeated many times over.

To help you track down leaks, ObjectTools keeps track of all objects that have been created but not cleared. This list is available by calling *OT GetHandleList*. ObjectTools comes with a 4D method for creating a log file of all leaked objects, showing their complete contents. If you call the leak logger from the *On Exit Database* method, you can get a good idea of what objects were leaked and then remedy the situation.

During development, you may want to reset the database to an "original" state without closing and opening the database. ObjectTools provides a method called *OT ClearAll* which is provided for this purpose. *OT ClearAll* clears all objects that are still existing, no matter where or how they were created.

Note: You should not rely on OT ClearAll as a way of managing object memory.

Using Item Tags

The key to storing values into and retrieving values from objects is to know how to use *item references*, or *tags*. Tags are the names you give to object items when you store (put) and retrieve (get) values from an object. For example, to put the *Real* value 27.13 into an object with the tag "my real", you would use:

```
OT PutReal ($object;"my real";27.13)
```

Assuming the object had just been created, this would create a new item in the object referenced by the handle *$object*. To retrieve the value stored with that tag, you would use:

```
$real:=OT GetReal ($object;"my real")
```

To replace the existing value of "my real", you would simply call *OT PutReal* again, like so:

```
OT PutReal ($object;"my real";827.1931)
```

Tag Characteristics

Tags can be up to 2GB Unicode characters, and may consist of any valid Unicode characters except period (.), as that is used to indicate embedded objects. Capitalization is normally not significant; thus "My Real" and "my real" are considered the same tag by ObjectTools. Note that diacritical marks are *always* significant in tags.

If necessary, you can set an option to have case be significant when matching tag names.

Item Types

Like 4D variables, every item in an object has a distinct type which can be accessed. When an item is created by putting a value into an object, it is assigned the type that was specified by the *OT Put<type>* call. Except for character types and embedded objects, the type of an item is identical to the equivalent 4D type.

ObjectTools defines several special item types for items that have no representation within 4D. Each of these types has a named constant defined which can be used to determine the item’s type.

Constant	Type	Value
OT Is Character	Characters	112
OT Character array	Character array	113
OT Is Object	Embedded ObjectTools object	114
OT Is Record	Record data	115

Except for character types, you must get values from objects with the same type used to put the value. In other words, the <*type*> in *OT Put*<*type*> and *OT Get*<*type*> must match. Otherwise ObjectTools will generate an error and return a null value. For example:

```
C_LONGINT($object)
$object:=OT New
OT PutReal ($object;"my real";13.27)

C_LONGINT($bad)
$bad:=OT GetLong ($object;"my real") `This generates an error
C_REAL($real)
$real:=OT GetReal ($object;"my real") `This is okay
```

The Character Item Type

Any characters put in an object, whether they start life as a *String* or as *Text*, have the item type *OT Is Character*.

Character items can be retrieved either as a *String* or as *Text* via *OT GetString*, *OT GetText*, or *OT GetVariable*. For example:

```
OT PutString ($object;"chars";"this was originally a string")

C_TEXT($text)
$text:=OT GetText ($object;"chars")

$text:="this was originally text"
OT PutText ($object;"chars";$text)

C_STRING(255;$str)
$str:=OT GetString ($object;"chars")
'Of course in Unicode mode C_TEXT and C_STRING are the same
```

Likewise, any *String* or *Text* arrays put into an object are stored with an item type of *OT Character array (113)*. Elements of this item type can then be retrieved either as a fixed width *String* or as *Text* via *OT GetArrayString* and *OT GetArrayText*.

Note: When running in Unicode mode, all text put into or retrieved from an object is Unicode text.

Putting and Getting Values Generically

In some situations it is constrictive to have to know the type of an item in order to choose which *OT Put* or *OT Get* command to use. ObjectTools allows you to put and get values generically, without having to know their type, by using the *OT PutVariable* and *OT GetVariable* commands.

These commands take pointers to a variable, through which values are stored and retrieved from an object. Thus you can generically pass a pointer to these commands without having to know in advance the type of the variable they point to, as long as the variable and item type match according to the rules mentioned above.

Embedded Objects

ObjectTools allows you to embed objects within objects. This lets you create hierarchically structured representations of heterogeneous data. An object stored within another object has a distinct item type (not *Is Longint*) to identify it as such.

Here's what a complex object might look like. Indentation denotes embedded objects:

Tag	Type	Contents
"fields"	Object	
"firstname"	Character	"John"
"lastname"	Character	"Doe"
"dialog"	Object	
"table"	Pointer	->[Contacts]
"form"	Character	"Input"
"left"	Longint	200
"top"	Longint	200
"width"	Longint	350
"height"	Longint	300
"title"	Character	"Contact Entry"

Here's the code to create this object:

```
C_LONGINT($object;$fields;$dialog)
$object:=OT New

OT PutString ($object;"fields.firstname";"John")
OT PutString ($object;"fields.lastname";"Doe")

OT PutPointer ($object;"dialog.table";->[Forms])
OT PuString ($object;"dialog.form";"ContactInput")
OT PutLong ($object;"dialog.left";200)
`And so on
```

As you can see from the above example, ObjectTools automatically creates embedded objects as necessary if they appear in the tag and don't yet in the object.

Accessing Embedded Objects

To access the "firstname" item in the "fields" Object, you would use:

```
$firstName:= OT GetString ($object;"fields.firstname")
```

This is what you would expect. But how would you access the "table" item in the embedded "dialog" object?

Embedded items are accessed using dot notation. Given an embedded object "foo", you access items within that object with the tag "foo.<item tag>".

So, for example, to access the "table" item inside the "dialog" object defined above, you would use:

```
C_POINTER($table)
OT GetPointer ($object;"dialog.table";$table)
```

If objects are nested more than one level deep, you just continue adding dots. So to access an item called "bar" inside an embedded object called "foo" inside an embedded object called "foobar", you would use "foobar.foo.bar".

Using Arrays with Objects

Frequently you will want to store and retrieve entire arrays in objects. To do so there are a pair of calls you use, *OT PutArray* and *OT GetArray*.

Except for *String* and *Text* arrays, you must put and get arrays into the same type of array variable. For example:

```
C_LONGINT($object)
$object:=OT New
ARRAY LONGINT($longs;1)
$longs{1}:=27
ARRAY REAL($reals;0)

OT PutArray ($object;"array";$longs)
`This generates an error
OT GetArray ($object;"array";$reals)
ARRAY LONGINT($longs;0)
OT GetArray ($object;"array";$longs)
`$longs is restored to its previous state
```

String and *Text* arrays, however, may be mixed and matched. For example:

```
ARRAY STRING(255;$str255s;1)
ARRAY TEXT($texts;1)

$str255s{1} := "this was originally a string"
$texts{1} := "this was originally text"

OT PutArray ($object;"char array";$texts)
OT GetArray ($object;"char array";$str255s)
`$str255s{1} contains "this was originally text"

C_STRING(255;$str255)
$str255:=OT GetArrayString ($object;"char array";1)
`$str255 contains "this was originally text"
```

Accessing Array Elements within Object Items

If you need to get or set individual elements of an array within an object, you can do so by using the *OT GetArray<type>* or *OT PutArray<type>*method, where *<type>* represents the array's type. For example, to get or set the seventh element of an array with the tag "strings", you would use:

```
$str:=OT GetArrayString ($object;"strings";7)
OT SetArrayString ($object;"strings";7;"This is a test")
```

In conjunction with the *OT SizeOfArray* method, this allows you to iterate over and retrieve the contents of an array within an object.

Other Array Utilities

ObjectTools also contains a full suite of commands for inserting and deleting array elements, as well as searching and sorting. This allows you to operate on arrays completely within an object without having to copy it out of the object, modify it, and then copy it back into the object.

Error Handling

It is virtually impossible to corrupt an object with any ObjectTools methods. ObjectTools uses extensive error checking to ensure that all object handles and item references are valid, and stops before any damage can be done.

When an error does occur, such as passing a bad object handle or item reference, ObjectTools generates an error, sets the *OK* variable to zero, and returns a null value. For more on ObjectTools error handling, see the documentation for *OT SetErrorHandler*.

The ObjectTools Log

ObjectTools 4 logs its internal operations to help you debug problems that are difficult to trace otherwise.

Logs are kept in <database structure directory>/Logs/ObjectTools, where the “Logs” directory is what would be returned by **Get 4D folder**(*Logs Folder*). Log files are rotated automatically when they reach 1MB in size. A total of seven log files are kept, with ObjectTools.0.log being the current log file, ObjectTools.1.log being the previous log file, and so on up to ObjectTools.6.log.

ObjectTools logs the following types of information in the log file:

- Information about the host environment
- Internal and runtime errors

Each log entry occupies one logical line and looks something like this:

```
Nov 20 17:08:34 ObjectTools: [notice] env: ObjectTools 4.0
[Macintosh/Intel, release]
```

Log entries contain the date and time of the entry, followed by “ObjectTools:”, followed by the entry type, followed by the message.

The log entry types are:

- **info:** General information about ObjectTools operations or environment

- **notice:** "Official" announcements
- **warn:** Conditions that may cause problems or errors and should be looked into
- **error:** Internal or runtime errors that should be attended to
- **debug:** Detailed information about ObjectTool's internal operations

Changing the Log Level

If the normal logging does not provide enough information to debug a problem, or if you would like to disable logging altogether, you can change the log level.

To change the log level, follow these steps:

1. In a text editor, create a new plain text document.
2. In the document, enter the text "debug" or "off".
3. Save the document as "log_level" in the ObjectTools log directory.
4. Restart 4D.

If ObjectTools finds "log_level" (or for backward compatibility, "log_debug_level") in the log directory and it contains "debug" or "off", the log level is set accordingly.

- When the log level is "debug", you will see many log extra entries of type "debug". This level gives you detailed information about the inner workings of ObjectTools.
- When the log level is "off", logging is completely turned off.

The default log level can be restored either my moving, renaming or deleting the "log_level" file or by deleting the text within the file, then restarting 4D.

CHAPTER 3

Command Reference

ObjectTools is comprised of a suite of plug-in routines and 4th DIMENSION methods, designed to extend the existing 4th DIMENSION Command Set, providing a variety of routines.

- **Creation/Destruction:** Used to create and destroy (delete) objects.
- **Putting Values:** Provide information for storing data in an object or sub-object(s).
- **Getting Values:** Provide information for retrieving data previously stored in an object or sub-object(s).
- **Array Utilities:** Utilities for manipulating arrays.
- **Object Info:** Obtain various state information about an object.
- **Item Info:** Obtain various state information about an object item.
- **Item Utilities:** Utility routines that operate on individual items.
- **Import/Export:** Routines for moving objects into and out of BLOBs.
- **Object Utilities:** Miscellaneous utility routines that operate on objects.

Documentation Conventions

In general, the conventions used for documenting plugin calls within this manual are the same as those within 4D's documentation. In addition, this manual uses a prefix for parameter names to indicate what happens to their data.

Prefix	Example	Meaning
in	inTitle	The parameter's data is read and left intact
out	outTitle	The parameter's data on entry to the call is ignored and is set by the call, replacing any previous data contained by the parameter
io	ioTitle	The parameter's data is read by the call and then either replaced or augmented

Creation and Destruction Routines

The following routines can be used to create and destroy ObjectTools objects. You must successfully create a valid object before using any other ObjectTools routine.

OT New

version 1

OT New → Longint

Parameter	**Type**		**Description**
Function result	Longint	←	A handle to a new, empty object

Discussion

Creates a new, empty object and returns it. The new object is added to internal list of objects. When you are finished with the object, call *OT Clear* to release the memory used by the object.

> **Warning:** Never attempt to pass any value to an ObjectTools routine other than that returned by *OT New*.

See Also

OT IsObject, OT Clear

OT Clear

version 1

OT Clear(ioObject)

Parameter	Type		Description
ioObject	Longint	↔	A handle to an object

Discussion

When you are finished with an object, you should always call *OT Clear* to release the memory occupied by the object. If you create an object and then lose track of its handle, you will no longer be able to release its memory. This is known as a *leak*, and it is considered a Bad Thing.

ObjectTools maintains an internal list of all objects that have been created but not cleared. When an object is disposed of with *OT Clear*, it is removed from the list. The current list of created objects is available with the *OT GetHandleList* method.

You can actually release the memory used by leaked objects with *OT GetHandleList* or *OT ClearAll*. However, using these as a means of object memory management is not recommended.

It you pass a variable directly to *OT Clear* (as opposed to an extremely lucky guess at a number constant), the variable will be set to zero.

Note: It is legal to pass a null handle (0) to *OT Clear*.

Examples

The sample code is below demonstrates creating two temporary objects to pass to a method. The first one will leak, while the second one is properly disposed with *OT Clear*.

```
C_LONGINT($leak;$notLeak)
$leak:=OT New
OT PutString ($leak;"name";[Contacts]Name)
$notLeak:=OT New
OT PutString ($notLeak;"address";[Contacts]Address)
MyMethod($leak;$notLeak)
OT Clear ($notLeak)   `The memory is released

`If we leave this method at this point, we will not be able to
`recover the value of $leak, so its memory will leak.
```

See Also

OT New, OT ClearAll

OT ClearAll

version 1

OT ClearAll

Discussion

This method disposes of all objects that have been created but not cleared via *OT Clear*. It is provided as a "fail-safe" way of cleaning up the memory used by objects, but this method should not be relied upon as a means of managing object memory usage.

The primary use for *OT ClearAll* is during development, when you frequently have to stop program execution. As a result it is quite possible that you may create a new object with *OT New* but never reach the code that calls *OT Clear*. In such cases you can execute a method that calls *OT ClearAll* to clear all of the existing objects. This way you can start over again without leaking memory and without having to close and open the database.

See Also

OT Clear, OT New, OT GetHandleList

OT Copy

version 1

OT Copy(inObject) → Longint

Parameter	Type		Description
inObject	Longint	→	A handle to an object
Function result	Longint	←	A handle to a new object

Discussion

OT Copy makes a complete copy of object and returns the copy. The copy is added to the ObjectTools handle list, and must be cleared with *OT IsObject* when it is no longer needed.

If memory cannot be allocated for the copy, an error is generated and *OK* is set to zero.

See Also

OT Clear, OT ClearAll

Put Value Routines

The following routines are used to store data in any ObjectTools object. After you have successfully created an object (see “Creation and Destruction Routines”), you can begin storing data into the object.

OT PutArray

version 1

OT PutArray(inObject; inTag; inArray)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inArray	Array	→	One-dimensional array to store

Discussion

OT PutArray puts *inArray* into *inObject*. The element count and current element are stored with the array elements and are restored by *OT GetArray*. You may not store two-dimensional arrays in objects.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has a compatible type (see below), its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

Array Type Compatibility

Except for *String* and *Text* arrays, you must put and get arrays into the same type of array variable. *String* and *Text* arrays, however, may be mixed and matched, because ObjectTools stores both types of array with an item type of *OT Character array* (113).

See Also

OT GetArray

OT PutArrayBLOB

v4.1r1

OT PutArrayBLOB(inObject; inTag; inIndex; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	BLOB	→	Value to set

Discussion

OT PutArrayBLOB sets an element of an array in *inObject*.

If the object is not a valid object handle, if no item in the object has the given tag, or if the 4D version is not v14 or later, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Blob array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayBLOB

OT PutArrayBoolean

version 2

OT PutArrayBoolean(inObject; inTag; inIndex; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Number	→	1=true, 0=false

Discussion

OT PutArrayBoolean sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Boolean array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set (0=false, 1=true).

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayBoolean

OT PutArrayDate

version 2

OT PutArrayDate(inObject; inTag; inIndex; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Date	→	Value to set

Discussion

OT PutArrayDate sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Date array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayDate

OT PutArrayLong

version 2

OT PutArrayLong(inObject; inTag; inIndex; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Number	→	Value to set

Discussion

OT PutArrayLong sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Longint array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayLong

OT PutArrayPicture

version 2

OT PutArrayPicture(inObject; inTag; inIndex; inValue)

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Picture	→	Value to set

Discussion

OT PutArrayPicture sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Picture array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayPicture

OT PutArrayPointer **version 2**

OT PutArrayPointer(inObject; inTag; inIndex; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Pointer	→	Value to set

Discussion

OT PutArrayPointer sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Pointer array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

> **Warning:** Under no circumstances should you attempt to store a pointer to a local or process variable in a compiled database and then try to retrieve that pointer in another process.

See Also

OT GetArrayPointer

OT PutArrayReal

version 2

OT PutArrayReal(inObject; inTag; inIndex; inValue)

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Number	→	Value to set

Discussion

OT PutArrayReal sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Real array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayReal

OT PutArrayString

version 2

OT PutArrayString(inObject; inTag; inIndex; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	String	→	Value to set

Discussion

OT PutArrayString sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *OT Character array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayString

OT PutArrayText

version 2

OT PutArrayText(inObject; inTag; inIndex; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Text	→	Value to set

Discussion

OT PutArrayText sets an element of an array in *inObject*.

If the object is not a valid object handle or if no item in the object has the given tag, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *OT Character array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayText

OT PutArrayTime

v4.1r1

OT PutArrayTime(inObject; inTag; inIndex; inValue)

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inIndex	Number	→	Index of array element to set
inValue	Time	→	Value to set

Discussion

OT PutArrayTime sets an element of an array in *inObject*.

If the object is not a valid object handle, if no item in the object has the given tag, or if the 4D version is not v14 or later, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Time array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is set to *inValue*.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated and *OK* is set to zero.

See Also

OT GetArrayTime

OT PutBLOB

version 1

OT PutBLOB(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	BLOB	→	4D BLOB to store

Discussion

OT PutBLOB puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is BLOB*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetBLOB

OT PutBoolean

version 2.5r3

OT PutBoolean(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Longint	→	Boolean value to store

Discussion

OT PutBoolean puts *inValue* into *inObject*. The value zero is considered False, any non-zero value is considered True.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is Boolean*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetBoolean

OT PutDate

version 1

OT PutDate(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Date	→	4D date to store

Discussion

OT PutDate puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is Date*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetDate

OT PutLong

version 1

OT PutLong(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Longint	→	Long to store

Discussion

OT PutLong puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is Longint*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetLong

OT PutObject

version 1

OT PutObject(inObject; inTag; inObject)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inObject	Longint	→	Handle of object to store

Discussion

OT PutObject puts *inObject* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *OT Is Object (114)*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

Note: An object put into another object still remains in memory. It is still your responsibility to clear it when you no longer need it by calling *OT Clear*. Do not do the following:

```
OT PutObject ($object;"bad thing!";OT New)
```

See Also

OT GetObject

OT PutPicture

version 1

OT PutPicture(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Picture	→	Picture to store

Discussion

OT PutPicture puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is Picture*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetPicture

OT PutPointer

version 1

OT PutPointer(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Pointer	→	Pointer to store

Discussion

OT PutPointer puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is Pointer*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

> **Warning:** Under no circumstances should you attempt to store a pointer to a local or process variable in a compiled database and then try to retrieve that pointer in another process.

See Also

OT GetPointer

OT PutReal

version 1

OT PutReal(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Real	→	Number to store

Discussion

OT PutReal puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is Real*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetReal

OT PutRecord

version 1.5

OT PutRecord(inObject; inTag; inTable)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inTable	Table/Field pointer	→	Table whose record you want to store

Discussion

OT PutRecord puts the current record into object in a packed format. The contents of the item can only be retrieved with *OT GetRecord*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *OT Is Record (115)*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero.

If table is not a valid table or field pointer, or if there is no current record for the given table, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

Warning: Once a record is stored with *OT PutRecord*, it must be retrieved into the same table. Otherwise the results are undefined (and potentially disastrous).

See Also

OT GetRecord, OT GetRecordTable

OT PutString

version 1

OT PutString(inObject; inTag; inValue)

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	String	→	String to store

Discussion

OT PutString puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *OT Is Character (112)*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See “The Character Item Type” on page 13 for more information on storing and retrieving strings.

See Also

OT PutText, OT GetString, OT GetText

OT PutText

version 1

OT PutText(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Text	→	Text to store

Discussion

OT PutText puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *OT Is Character (112)*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See “The Character Item Type” on page 13 for more information on storing and retrieving strings.

See Also

OT PutString, OT GetText, OT GetString

OT PutTime

v4.1r1

OT PutTime(inObject; inTag; inValue)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inValue	Time	→	Time to store

Discussion

OT PutTime puts *inValue* into *inObject*.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Is Time*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetLong

OT PutVariable

version 1.5

OT PutVariable(inObject; inTag; inVarPointer)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
inVarPointer	Variable pointer	→	Pointer to variable which contains value to store

Discussion

OT PutVariable puts the contents of the variable pointed to by *inVarPointer* into *inObject*. Every 4D variable type but 2D arrays can be stored with this command, including *Boolean* variables and arrays. Once stored, the data can either be retrieved with *OT GetVariable* or with the *OT Get<type>* command appropriate to the variable's type.

If *inObject* is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, a new item is created.

If an item with the given tag exists and has the type *Type(variablePointer->)*, its value is replaced.

If an item with the given tag exists and has any other type, an error is generated and *OK* is set to zero if the *OT VariantItems* option is not set, otherwise the existing item is deleted and a new item is created.

See Also

OT GetVariable

Get Value Routines

The following routines provide the ability to get the value of any object item. After you have successfully put data into an object item, you can begin retrieving data from the object.

OT GetArray

version 1

OT GetArray(inObject; inTag; outArray)

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to set
outArray	Array	←	Array to receive the item's contents

Discussion

OT GetArray gets an array value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and *outArray* is cleared.

If no item in the object has the given tag, *outArray* is cleared. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has a compatible type, the array's contents are replaced.

If an item with the given tag exists and has any other type, an error is generated, *OK* is set to zero, and array is cleared.

Array Type Compatibility

Except for *String* and *Text* arrays, you must put and get arrays into the same type of array variable. *String* and *Text* arrays, however, may be mixed and matched, because ObjectTools stores both types of array with an item type of *OT Character array (113)*.

Note: If you retrieve into a fixed width string array and your database is running in compatibility mode, the elements will be truncated to the width of the array.

See Also

OT PutArray

OT GetArrayBLOB v4.1r1

OT GetArrayBLOB(inObject; inTag; inIndex) → BLOB

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	BLOB	←	The array element's contents

Discussion

OT GetArrayBLOB gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle or if the 4D version is not v14 or later, an error is generated, *OK* is set to zero, and and empty BLOB is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Blob array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and an empty BLOB is returned.

See Also

OT PutArray, OT PutArrayBLOB

OT GetArrayBoolean

version 1

OT GetArrayBoolean(inObject; inTag; inIndex) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	Number	←	The array element's contents

Discussion

OT GetArrayBoolean gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Boolean array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned as a number (0=false, 1=true).

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and zero is returned.

See Also

OT PutArrayBoolean

OT GetArrayDate

version 1

OT GetArrayDate(inObject; inTag; inIndex) → Date

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	Date	←	The array element’s contents

Discussion

OT GetArrayDate gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and a null date (!00/00/00!) is returned.

If no item in the object has the given tag, a null date is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Date array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and a null date is returned.

See Also

OT PutArray, OT GetDate

OT GetArrayLong

version 1

OT GetArrayLong(inObject; inTag; inIndex) → Number

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	Number	←	The array element's contents

Discussion

OT GetArrayLong gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Longint array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and zero is returned.

See Also

OT PutArray, OT GetLong

OT GetArrayPicture **version 1**

OT GetArrayPicture(inObject; inTag; inIndex) → Picture

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	Picture	←	The array element's contents

Discussion

OT GetArrayPicture gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty picture is returned.

If no item in the object has the given tag, an empty picture is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Picture array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and an empty picture is returned.

See Also

OT PutArray, OT GetPicture

OT GetArrayPointer

version 1

OT GetArrayPointer(inObject; inTag; inIndex; outPointer)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
outPointer	Pointer	←	Receives the array element's contents

Discussion

OT GetArrayPointer gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and a nil pointer is returned.

If no item in the object has the given tag, a nil pointer is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Pointer array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and a nil pointer is returned.

Warning: Under no circumstances should you attempt to store a pointer to a local or process variable in a compiled database and then try to retrieve that pointer in another process.

See Also

OT PutArray, OT GetPointer

OT GetArrayReal

version 1

OT GetArrayReal(inObject; inTag; inIndex) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	Number	←	The array element's contents

Discussion

OT GetArrayReal gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Real array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and zero is returned.

See Also

OT PutArray, OT GetReal

OT GetArrayString

version 1

OT GetArrayString(inObject; inTag; inIndex) → String

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	String	←	The array element's contents

Discussion

OT GetArrayString gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty string is returned.

If no item in the object has the given tag, an empty string is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *OT Character array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and an empty string is returned.

See "The Character Item Type" on page 13 for more information on storing and retrieving strings.

Note: If your database is running in compatibility mode and the result of this method is assigned to a fixed width string variable, the item's contents will be truncated to the width of the variable. To retrieve more than 255 characters from a character array, use *OT GetArrayText* and assign to a text variable or field.

See Also

OT PutArray, OT GetArrayText

OT GetArrayText

version 1

OT GetArrayText(inObject; inTag; inIndex) → Text

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	Text	←	The array element's contents

Discussion

OT GetArrayText gets a value in *inObject* from the array item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty string is returned.

If no item in the object has the given tag, an empty string is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *OT Character array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and an empty string is returned.

See "The Character Item Type" on page 13 for more information on storing and retrieving text.

Note: If your database is running in compatibility mode and the result of this method is assigned to a fixed width string variable, the item's contents will be truncated to the width of the variable. To retrieve more than 255 characters from a character array, assign the result to a text variable or field.

See Also

OT PutArray, OT GetArrayString

OT GetArrayTime

v4.1r1

OT GetArrayTime(inObject; inTag; inIndex) → Time

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
inIndex	Number	→	Index of array element to retrieve
Function result	Time	←	The array element's contents

Discussion

OT GetArrayTime gets a value in *inObject* from the array item referenced by *inTag*.

the object is not a valid object handle or if the 4D version is not v14 or later, an error is generated, *OK* is set to zero, and the time ?00:00:00? is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Time array*, and *inIndex* is in the range (0..*OT SizeOfArray*(*inObject; inTag*)), the value of the requested element is returned.

If an item with the given tag exists and has any other type, or if the index is out of range, an error is generated, *OK* is set to zero, and the time ?00:00:00? is returned.

See Also

OT PutArray, OT PutArrayTime

OT GetBLOB

version 1

OT GetBLOB(inObject; inTag; outBLOB)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
outBLOB	BLOB	←	The retrieved item

Discussion

OT GetBLOB gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty BLOB is returned.

If no item in the object has the given tag, an empty BLOB is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is BLOB*, *outBLOB*'s contents are replaced.

If an item with the given tag exists and has any other type, *OK* is set to zero, and an empty BLOB is returned.

Warning: Do not attempt to pass a BLOB field or a dereferenced pointer to a BLOB field in the blob paremeter, as this will result in a crash. If you want to retrieve a BLOB item into a field, either use an intermediate local variable or assign the result of *OT GetNewBLOB* to the field. The same applies to passing a dereferenced pointer to a BLOB variable.

This command is being kept for backward compatibility. Because of the problems related to this command, it is recommended that you use *OT GetNewBLOB* instead, as this command may be removed in future versions.

See Also

OT PutBLOB, OT GetNewBLOB

OT GetBoolean

version 2.5r3

OT GetBoolean(inObject; inTag) → Longint

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Longint	←	The retrieved item

Discussion

OT GetBoolean gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is Boolean*, the value of the requested item is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and zero is returned.

See Also

OT PutBoolean, OT GetArrayBoolean

OT GetDate

version 1

OT GetDate(inObject; inTag) → Date

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Date	←	The retrieved item

Discussion

OT GetDate gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and a null date (!00/00/00!) is returned.

If no item in the object has the given tag, a null date is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is Date*, the value of the requested item is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and a null date is returned.

See Also

OT PutDate, OT GetArrayDate

OT GetLong

version 1

OT GetLong(inObject; inTag) → Longint

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Longint	←	The retrieved item

Discussion

OT GetLong gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is Longint*, the value of the requested item is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and zero is returned.

See Also

OT PutLong, OT GetArrayLong

OT GetNewBLOB

version 1.5

OT GetNewBLOB(inObject; inTag) → BLOB

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	BLOB	←	The retrieved item

Discussion

OT GetNewBLOB gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty BLOB is returned.

If no item in the object has the given tag, an empty BLOB is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is BLOB*, the value of the requested item is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and an empty BLOB is returned.

Warning: Because of the problems related to the *OT GetBLOB* command, it is recommended that you use this command instead.

See Also

OT PutBLOB, OT GetBLOB

OT GetObject

version 1

OT GetObject(inObject; inTag) → Longint

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Longint	←	A handle to a new object

Discussion

OT GetObject gets an object value in object from the item referenced by *inTag*. If this routine successfully returns a new object, the new object handle is added to the ObjectTools handle list.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and a null object handle (0) is returned.

If no item in the object has the given tag, a null object handle is returned.

If an item with the given tag exists and has the type *OT Is Object (114)*, its contents are returned as a new object.

If an item with the given tag exists and has any other type, an error is generated, *OK* is set to zero, and a null object handle is returned.

Warning: This method creates and returns a new object in memory. You are responsible for clearing it when you no longer need it by calling *OT IsObject*.

See Also

OT PutObject, OT IsObject

OT GetPicture

version 1

OT GetPicture(inObject; inTag) → Picture

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Picture	←	The retrieved item

Discussion

OT GetPicture gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty picture is returned.

If no item in the object has the given tag, an empty picture is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is Picture*, the value of the requested item is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and an empty picture is returned.

See Also

OT PutPicture, OT GetArrayPicture

OT GetPointer

version 1

OT GetPointer(inObject; inTag; outPointer)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
outPointer	Pointer	←	The retrieved item

Discussion

OT GetPointer gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and a nil pointer is returned.

If no item in the object has the given tag, a nil pointer is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is Pointer*, the value of the requested element is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and a nil pointer is returned.

> **Warning:** Under no circumstances should you attempt to store a pointer to a local or process variable in a compiled database and then try to retrieve that pointer in another process.

See Also

OT PutPointer, OT GetArrayPointer

OT GetReal

version 1

OT GetReal(inObject; inTag) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Number	←	The retrieved item

Discussion

OT GetReal gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If no item in the object has the given tag, zero is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is Real*, the value of the requested item is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and zero is returned.

See Also

OT PutReal, OT GetArrayReal

OT GetRecord

version 1.5

OT GetRecord(inObject; inTag)

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve

Discussion

OT GetRecord sets the current record of a table from the packed record data in the item referenced by *inTag*. The contents of the item must have been set with *OT PutRecord*. The table used to store the packed record is the table which will have its current record set.

If object is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in object has the given tag, nothing happens.

If an item with the given tag exists and has the type *OT Is Record*, the current record of the item’s original table is set.

If there is no current record for the item’s table or the current record is locked, an error is generated and *OK* is set to zero.

> **Warning:** Once a record is stored with *OT PutRecord*, it must be retrieved into the same table. Otherwise the results are undefined (and potentially disastrous). You can use the *OT GetRecordTable* command to find the source table for a stored record.

See Also

OT PutRecord, OT GetRecordTable

OT GetRecordTable

version 1.5

OT GetRecordTable(inObject; inTag) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Number	←	The retrieved item's table number

Discussion

OT GetRecordTable retrieves the table number from the packed record data in the item referenced by tag. The contents of the item must have been set with *OT PutRecord*. The table used to store the packed record is the table whose number will be returned.

If the object is not a valid object handle, or no item in object has the given tag, zero is returned, an error is generated and OK is set to zero.

If an item with the given tag exists and has the type *OT Is Record*, the number of the item's original table is returned.

If an item with the given tag exists and has any other type, zero is returned, an error is generated and OK is set to zero.

See Also

OT GetRecord, OT PutRecord

OT GetString

version 1

OT GetString(inObject; inTag) → String

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	String	←	The retrieved item

Discussion

OT GetString gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty string is returned.

If no item in the object has the given tag, an empty string is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *OT Is Character (112)*, the value of the requested element is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and an empty string is returned.

See "The Character Item Type" on page 13 for more information on storing and retrieving strings.

Warning: If your database is running in compatibility mode and the result of this method is assigned to a fixed width string variable, the item's contents will be truncated to the width of the variable. To retrieve more than 255 characters from a character item, use *OT GetText* and assign to a text variable or field.

See Also

OT PutString, OT GetText, OT GetArrayString

OT GetText

version 1

OT GetText(inObject; inTag) → Text

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Text	←	The retrieved item

Discussion

OT GetText gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and an empty string is returned.

If no item in the object has the given tag, an empty string is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *OT Is Character (112)*, the value of the requested element is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and an empty string is returned.

See "The Character Item Type" on page 13 for more information on storing and retrieving strings.

See Also

OT PutText, OT PutString, OT GetString, OT GetArrayText

OT GetTime

v4.1r1

OT GetTime(inObject; inTag) → Time

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
Function result	Time	←	The retrieved item

Discussion

OT GetTime gets a value in *inObject* from the item referenced by *inTag*.

If the object is not a valid object handle, an error is generated, *OK* is set to zero, and the time ?00/00/00? is returned.

If no item in the object has the given tag, the time ?00:00:00? is returned. If the *FailOnNoItem* option is set, an error is generated and *OK* is set to zero.

If an item with the given tag exists and has the type *Is Time*, the value of the requested item is returned.

If an item with the given tag exists and has any other type, *OK* is set to zero, and the time ?00:00:00? is returned.

See Also

OT PutTime, OT GetArrayTime

OT GetVariable

version 1.5

OT GetVariable(inObject; inTag; outVarPointer)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to retrieve
outVarPointer	Variable pointer	←	Pointer to a variable to receive the named item

Discussion

OT GetVariable gets a value in *inObject* from the item referenced by *inTag*. Every 4D variable type but 2D arrays can be retrieved with this command, including *Boolean* variables and arrays.

If the object is not a valid object handle, an error is generated and *OK* is set to zero.

If no item in the object has the given tag, nothing happens.

If an item with the given tag exists and has the same type as the type of the destination variable, the variable's data is replaced with the data stored in the object.

If an item with the given tag exists and has a type other than the type of the destination variable, an error is generated and *OK* is set to zero.

See Also

OT PutVariable

Array Utility Routines

The following routines provide commands for manipulating, searching and sorting arrays. These commands are analogous to the array commands in 4D. It is far more efficient to use these commands than to use *OT GetArray*, manipulate the array, then use *OT PutArray*.

OT DeleteElement

version 2

OT DeleteElement(inObject; inTag; inWhere {; inHowMany})

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the array item to change
inWhere	Number	→	Element to delete
inHowMany	Number	→	How many elements to delete

Discussion

OT DeleteElement deletes one or more elements from an array in *inObject*.

If *inObject* is not a valid object handle, if no item in the object has the given tag, or if the item's type is not an array type, an error is generated and *OK* is set to zero.

Elements are deleted starting at the element specified by *inWhere*. The *inHowMany* parameter is the number of elements to delete. If *inHowMany* is not specified or zero, then one element is deleted. The size of the array shrinks by *inHowMany* elements.

See Also

OT InsertElement

OT FindInArray

version 2

OT FindInArray(inObject; inTag; inValue {; inStart}) → Longint

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the array item to change
inValue	Text	→	Value to search for
inStart	Number	→	Element at which to start search
Function result	Number	←	The index of the first element found

Discussion

OT FindInArray searches an array in inObject for the value *inValue*.

If *inObject* is not a valid object handle, if no item in the object has the given tag, or if the item's type is not an array type, an error is generated, *OK* is set to zero, and -1 is returned.

If *inStart* is not specified or is zero, it defaults to 1. The text *inValue* is converted to the type appropriate for the array being searched. For example, for a *Longint array* or *Real array*, *inValue* is converted as if it where passed to the 4D *Num* command. The formats used to convert values are as follows:

Array type	Example inValue
Boolean array	"true" or "1" = true, "false" or "0" = false
Date array	String(!08/27/31!)
Longint array	String(7)
Real array	String(13.27)

The result of the command is the index of the first matching element, or -1 if no match is found.

Note: Wildcards may be used when searching string/text arrays just as in 4D.

OT InsertElement

version 2

OT InsertElement(inObject; inTag; inWhere {; inHowMany})

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the array item to change
inWhere	Number	→	Where to insert
inHowMany	Number	→	How many elements to insert

Discussion

OT InsertElement inserts one or more elements into an array in inObject.

If *inObject* is not a valid object handle, if no item in the object has the given tag, or if the item's type is not an array type, an error is generated and *OK* is set to zero.

The new elements are inserted before the element specified by *inWhere*, and are initialized to an empty value for the array type. All elements beyond *inWhere* are moved up to make room for the new elements.

If *inWhere* is greater than the size of the array, the elements are added to the end of the array.

See Also

OT DeleteElement

OT ResizeArray

version 2

OT ResizeArray(inObject; inTag; inSize)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the array item to change
inSize	Number	→	New array size

Discussion

OT ResizeArray resizes an array in *inObject*.

If *inObject* is not a valid object handle, if no item in the object has the given tag, or if the item's type is not an array type, an error is generated and *OK* is set to zero.

If *inSize* is greater than the current size of the array, empty elements are added to the end of the array. If *inSize* is less than the current size of the array, elements from *inSize* + *1* to the end of the array are deleted.

See Also

OT DeleteElement, OT InsertElement, OT SizeOfArray

OT SizeOfArray

version 1

OT SizeOfArray(inObject; inTag) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to query
Function result	Number	←	The size of the item’s array

Discussion

OT SizeOfArray returns the number of elements in an array item within an object.

If *inObject* is not a valid object handle, if no item in the object has the given tag, or if the item’s type is not an array type, an error is generated, *OK* is set to zero, and zero is returned.

See Also

OT PutArray, OT GetArray

OT SortArrays

version 1

OT SortArrays(inObject; inTag1; inDirection1 {; ...inTag7; inDirection7})

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag1	Text	→	Tag of the array to sort
inDirection1	String	→	Direction of sort

Discussion

OT SortArrays performs a multilevel sort on one or more arrays in *inObject*. You may sort up to seven arrays at once with this command.

If *inObject* is not a valid object handle, if no item in the object has the given tag, if the item's type is not a sortable array type, if all of the arrays do not have the same number of elements, or if a direction is not valid, an error is generated and *OK* is set to zero.

The direction should be one of three values to indicate how to sort the array:

Value	Sort direction
">"	Ascending
"<"	Descending
"*"	Move with previous array

For example, to sort parallel arrays of names and associated ids, you would use something like this:

```
OT SortArrays($object;"names";">";"ids";"*")
```

To sort a group of addresses by state and then city within each state, you would use something like this:

```
OT SortArrays($object;"states";">";"cities";">")
```

Object Info Routines

The following routines provide the ability to obtain complete information about an object as a whole. To obtain information about individual items within an object, see “Item Info Routines” on page 88.

OT IsObject

version 1

OT IsObject(inObject) → Longint

Parameter	Type		Description
inObject	Longint	→	A handle to an object
Function result	Longint	←	1 if inObject is an ObjectTools object handle, 0 if not

Discussion

To test whether a given *Longint* value is a valid object handle, use *OT IsObject*. If *inObject* points to a valid object, 1 is returned. If *inObject* is zero or points to some other type of object, zero is returned.

While it is possible to construct a BLOB that would fool ObjectTools into thinking it was a object, this is extremely unlikely.

All ObjectTools methods check the validity of the object handle passed in to prevent truly nasty things from happening. Unless you are unsure about the contents of a variable or field passed to ObjectTools as a object handle, there is no need to call *OT IsObject* first.

Example

If you try to retrieve an embedded object from another object and it does not exist, a null object handle is returned. In that case you would want to test the result as shown in the example below.

```
C_STRING(255;$tag)
$tag:=Request("Item tag:")

If ((OK=1) & (Length($tag)>0))
   C_LONGINT($embedded)
   $embedded:=OT GetObject (stMyObject;$tag)   `$tag may not be
valid!

   If (OT IsObject ($embedded))
      `Do something with the object
   End if
End if
```

The above example assumes that the requested tag is valid and tries to get the embedded object before checking the tag’s validity. Another approach would be to check the tag first by using *OT ItemExists*.

See Also

OT ItemExists

OT ItemCount

version 1

OT ItemCount(inObject {; inTag}) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of an embedded object
Function result	Number	←	The item count

Discussion

OT ItemCount returns the number of top level items in the referenced object. Items in embedded objects are not included in the count.

If *inObject* is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If the tag is not passed or is empty, the count of top level items in the object is returned.

If the tag is passed, is not empty, and is a valid item reference for an embedded object, the count of top level items in the embedded object is returned.

Otherwise an error is generated, *OK* is set to zero, and zero is returned.

See Also

OT IsObject, OT IsEmbedded

OT ObjectSize

version 1

OT ObjectSize(inObject) → Number

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
Function result	Number	←	The total size of the object in bytes

Discussion

OT ObjectSize returns the total size of an object in memory. If *inObject* is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

See Also

OT GetAllProperties, OT GetItemProperties

Item Info Routines

The following routines provide the ability to obtain various information about each item in an object. These routines are useful if you want to deal with objects in a generic way and need to know how to classify each item.

OT GetAllNamedProperties

version 3

OT GetAllNamedProperties(inObject; inTag; outNames {; outTypes {; outItemSizes {; outDataSizes}}})

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of an embedded object
outName	String/Text array	←	Receives item names
outTypes	Longint array	←	Receives item types
outItemSizes	Longint array	←	Receives item sizes in bytes
outDataSizes	Longint array	←	Receives item data sizes in bytes

Discussion

OT GetAllNamedProperties returns information about all items in the object (or embedded object) referenced by *inObject* and *inTag*. If *inTag* is empty, information on *inObject* is returned. The information is returned in the given arrays. The arrays will contain one element for each item in object.

If the object is not a valid object handle, *inTag* is non-empty and does not reference an embedded object, or if the arrays are not of the type specified, an error is generated, the arrays are cleared and *OK* is set to zero.

The sizes in *outItemSizes* represent the total size of the item within the object, including the item’s data, tag and other internal information. The sizes in *outDataSizes* represent the size of the item’s data.

Note: Item names are returned in an indeterminate order, so you may want to sort the arrays after making this call.

See Also

OT GetAllProperties, OT GetItemProperties, OT GetNamedProperties

OT GetAllProperties

version 1
modified version 2.0

OT GetAllProperties(inObject; outNames {; outTypes {; outItemSizes {; outDataSizes}}})

Parameter	Type		Description
inObject	Longint	→	A handle to an object
outName	String/Text array	←	Receives item names
outTypes	Longint array	←	Receives item types
outItemSizes	Longint array	←	Receives item sizes in bytes
outDataSizes	Longint array	←	Receives item data sizes in bytes

Discussion

OT GetAllProperties returns information about all items in *inObject* into the given arrays. The arrays will contain one element for each item in object.

If the object is not a valid object handle or if the arrays are not of the type specified, an error is generated, the arrays are cleared and *OK* is set to zero.

The sizes in *outItemSizes* represent the total size of the item within the object, including the item's data, tag and other internal information. The sizes in *outDataSizes* represent the size of the item's data.

Note: Item names are returned in an indeterminate order, so you may want to sort the arrays after making this call.

See Also

OT GetAllNamedProperties, OT GetItemProperties, OT GetNamedProperties

OT GetItemProperties

version 1
modified version 2.0

OT GetItemProperties(inObject; inIndex; outName {; outType {; outItemSize {; outDataSize}}})

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inIndex	Longint	→	An index from 1 to the number of items in the object
outName	Text	←	Receives the item's name
outType	Longint	←	Receives the item's type
outItemSize	Longint	←	Receives the item's size
outDataSize	Longint	←	Receives the item data's size

Discussion

OT GetItemProperties returns the properties of a given item. Items are numbered according to the number of items in an object, starting with 1. In conjunction with *OT ItemCount*, this allows you to iterate over all of the items in the object.

If *inObject* is not a valid object handle or if the index is out of range, an error is generated, *OK* is set to zero, and the return variables are left untouched.

Note: This command has been kept for backwards compatibility. It is recommended that you not use this command, as the object items are stored in indeterminate order, thus making the item index useless. You should use *OT GetNamedProperties* instead.

See Also

OT GetAllNamedProperties, OT GetAllProperties, OT GetNamedProperties

OT GetNamedProperties

version 1
modified version 2.0

OT GetNamedProperties(inObject; inTag; outType {; outItemSize
{; outDataSize {; outIndex}}}})

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	An item tag
outType	Longint	←	Receives the item's type
outItemSize	Longint	←	Receives the item's size including the tag
outDataSize	Longint	←	Receives the item's size excluding the tag
outIndex	Longint	←	Receives the item's index

Discussion

OT GetNamedProperties returns the properties of the item identified by the tag *inTag*.

If *inObject* is not a valid object handle or if no item in object has the given tag, an error is generated, *OK* is set to zero, and the return variables are left untouched.

Note: *outIndex* will always be zero, as it is meaningless. It has been kept for backwards compatibility.

See Also

OT GetAllNamedProperties, OT GetAllProperties, OT GetItemProperties

OT IsEmbedded

version 1

OT IsEmbedded(inObject; inTag) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to query
Function result	Number	←	1 if the given item is an embedded object, 0 if not

Discussion

OT IsEmbedded tests the item referenced by *inTag* to see if it is an embedded object.

If *inObject* is not a valid object handle or if no item in object has the given tag, an error is generated, *OK* is set to zero, and zero is returned.

If an item with the given tag exists and has the type *OT Is Object*, 1 is returned.

If an item with the given tag exists and has any other type, zero is returned.

See Also

OT ItemType, OT GetItemProperties, OT GetNamedProperties

OT ItemExists

version 1

OT ItemExists(inObject; inTag) → Number

Parameter	**Type**		**Description**
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to query
Function result	Number	←	1 if the given item exists, 0 if not

Discussion

OT ItemExists tests for the existence of the given item. *inTag* may refer to a top level item, an embedded object or an embedded item.

If *inObject* is not a valid object handle, an error is generated, *OK* is set to zero, and zero is returned.

If an item with the given tag exists, 1 is returned. Otherwise zero is returned.

OT ItemType

version 1

OT ItemType(inObject; inTag) → Number

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to query
Function result	Number	←	The type of the item

Discussion

OT ItemType returns the type of the item referenced by *inTag*.

If *inObject* is not a valid object handle or if no item in object has the given tag, an error is generated, *OK* is set to zero, and zero is returned.

If an item with the given tag exists, its type is returned.

See Also

OT GetAllNamedProperties, OT GetAllProperties, OT GetNamedProperties, OT GetItemProperties

Item Utility Routines

The following routines allow you to fold, spindle and otherwise manipulate individual items within an object.

OT CompareItems

version 1

OT CompareItems(inSourceObject; inSourceTag;
inCompareObject; inCompareTag) → Number

Parameter	**Type**		**Description**
inSourceObject	Longint	→	A handle to an object
inSourceTag	Text	→	An item tag
inCompareObject	Longint	→	A handle to an object
inCompareTag	Text	→	An item tag
Function result	Number	←	0 if not identical, 1 if identical, -1 if an error occurred

Discussion

OT CompareItems compares two items for equality. *inSourceObject* and *inCompareObject* may be the same object.

If *inSourceObject* or *inCompareObject* is not a valid object handle, if either of the two items do not exist, or if the two items do not have the same type, an error is generated, *OK* is set to zero, and -1 is returned.

Otherwise, the items are compared according to the rules of equality used for equivalent variable types in 4D, with the addition that you may compare array, *BLOB*, *Picture* and embedded object items.

- Arrays are considered identical if they are the same size and the corresponding elements would be considered equal in 4D. This means that when comparing elements of character arrays, case and diacriticals are not significant and wildcards are used.
- *BLOB* and *Picture* items are considered identical if they contain the same data byte for byte.
- Embedded objects are considered identical if each of their items are considered identical according to the rules for non-object types.

OT RenameItem **version 2**

OT RenameItem(inObject; inTag; inNewTag)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	A full item tag
inNewTag	Text	→	The new item tag

Discussion

OT RenameItem renames the item referenced by *inTag* to the item referenced by *inNewTag*. Note that *inTag* must be a full tag if the target item is in an embedded object, whereas *inNewTag* is the new item name only. For example:

```
OT RenameItem ($obj;"foo.bar.old_name";"new_name")
```

The above example will rename the item *old_name* to *new_name* within the embedded object *foo.bar*. To access the renamed item you would use the tag *"foo.bar.new_name"*.

If the object handle is invalid, or if the item does not exist, or if an existing item has the same name as *inNewTag*, an error is generated, *OK* is set to zero, and no rename is performed.

OT CopyItem

version 1

OT CopyItem(inSourceObject; inSourceTag; inDestObject; inDestTag)

Parameter	Type		Description
inSourceObject	Longint	→	A handle to an object
inSourceTag	Text	→	An item tag
inDestObject	Longint	→	A handle to an object
inDestTag	Text	→	An item tag

Discussion

OT CopyItem copies the item referenced by *inSourceTag* to the item referenced by *inDestTag*. The item referenced by *inDestTag* need not exist; it will be created if necessary. The source and destination objects may be the same, allowing either duplication of an item at the same level of embedding within an object, or copying an item from one level of embedding to another.

If either object handle is not valid, or if the source item does not exist, or if the source item and destination item do not have the same type, an error is generated, *OK* is set to zero, and no copy is performed.

Note: Copying an embedded object recursively copies all of its items.

OT DeleteItem

version 1

OT DeleteItem(inObject; inTag)

Parameter	Type		Description
inObject	Longint	→	A handle to an object
inTag	Text	→	Tag of the item to delete

Discussion

OT DeleteItem deletes an item from an object. *inTag* may refer to embedded items and objects.

If *inObject* is not a valid object handle or *inTag* refers to an item that does not exist, an error is generated, *OK* is set to zero, and no delete is performed.

Note: Deleting an embedded object recursively deletes all of its items.

Import/Export Routines

The following routines provide the ability to import and export objects to 4D *BLOB* variables. This allows you to easily save and restore objects either to the database or to files on disk.

OT BLOBToObject **version 1**

OT BLOBToObject(inBLOB {; ioOffset}) → Longint

Parameter	Type		Description
inBLOB	BLOB	→	A BLOB which contains an object
ioOffset	Longint	↔	The offset within the BLOB where the object can be found

Discussion

OT BLOBToObject retrieves an object from a *BLOB* into a new object handle. The Object must have been stored in the *BLOB* with *OT ObjectToBLOB*/*OT ObjectToNewBLOB*, not with *VARIABLE TO BLOB*.

If *ioOffset* is not passed in it defaults to zero.

If the bytes at the given offset do not describe an object stored with *OT ObjectToBLOB/OT ObjectToNewBLOB*, or if the serialized object contains BLOB or Time arrays and the version of 4D is not v14 or later, an error is generated, *OK* is set to zero, *ioOffset* is left untouched and a null handle (0) is returned.

OT BLOBToObject transparently converts *BLOBs* created with earlier versions of ObjectTools.

> **Warning:** The handle returned is a new object that is added to ObjectTools' internal list of objects. You must be sure to clear the new object with *OT Clear* when you no longer need it.

See Also

OT ObjectToBLOB, OT ObjectToNewBLOB

OT ObjectToBLOB

version 1

OT ObjectToBLOB(inObject; ioBLOB {; inAppend})

Parameter	Type		Description
inObject	Longint	→	An object handle
ioBLOB	BLOB	↔	A BLOB which receives the object
inAppend	Longint	→	0 to replace ioBLOB's contents, non-zero to append to ioBLOB

Description

OT ObjectToBLOB stores an object into a *BLOB*. The previous contents of the *BLOB*, if any, are completely replaced, unless a non-zero value is passed in *inAppend*, in which case the object is appended to *inBLOB*.

Once stored within a *BLOB*, you must retrieve an object from it with *OT BLOBToObject*, not with *BLOB TO VARIABLE*.

If *inObject* is not a valid object handle, if *ioBLOB* is not a valid *BLOB*, or if memory cannot be allocated to copy the object, an error is generated, *OK* is set to zero, and *ioBLOB* is cleared.

Warning: Do not attempt to open an object saved in ObjectTools 4 with a version earlier than v3.

Warning: Do not attempt to pass a *BLOB* field or a dereferenced pointer to a *BLOB* field in the *ioBLOB* paremeter, as this will result in a crash. If you want to retrieve a *BLOB* item into a field, either use an intermediate local variable or assign the result of *OT ObjectToNewBLOB* to the field.

The Object passed to *OT ObjectToBLOB* is copied into blob and remains in memory. You must be sure to clear it with *OT Clear* when you no longer need it.

See Also

OT BLOBToObject, OT ObjectToNewBLOB

OT ObjectToNewBLOB

version 1.5

OT ObjectToNewBLOB(inObject) → BLOB

Parameter	**Type**		**Description**
inObject	Longint	→	An object handle
Function result	BLOB	←	A new BLOB which contains the object

Description

OT ObjectToNewBLOB stores an object into a new *BLOB*.

Once stored within a *BLOB*, you must retrieve an object from it with *OT BLOBToObject*, not with *BLOB TO VARIABLE*.

If *inObject* is not a valid object handle or if memory cannot be allocated to copy the object, an error is generated, *OK* is set to zero, and an empty *BLOB* is returned.

Warning: Do not attempt to open an object saved in ObjectTools 4 with a version earlier than v3.

Warning: The Object passed to *OT ObjectToNewBLOB* is copied into blob and remains in memory. You must be sure to clear it with *OT Clear* when you no longer need it.

See Also

OT BLOBToObject, OT ObjectToBLOB

Object Utility Routines

The following routines provide various utility calls that deal with ObjectTools on a global basis.

OT CompiledApplication

version 1

OT CompiledApplication → Longint

Parameter	Type		Description
Function result	Longint	←	1 if the application is compiled, 0 if interpreted

Description

OT CompiledApplication is the same as the 4D command *Compiled application*. It is no longer necessary but has been kept for backwards compatibility.

OT GetHandleList

version 1

OT GetHandleList(outHandles)

Parameter	Type		Description
outHandles	Longint array	←	Receives a list of all current objects

Discussion

Any time an object is created, whether through *OT New*, *OT Copy*, *OT GetObject*, or *OT BLOBToObject*, ObjectTools adds the new object handle to an internal list. When an object is cleared with *OT Clear*, the object's handle is removed from the list.

OT GetHandleList retrieves this internal list into an array. This is mainly of use in debugging. Normally you would have no need to use this method.

See Also

OT New, OT Clear

OT GetOptions

version 1
modified version 2.0

OT GetOptions → Longint

Parameter	**Type**		**Description**
Function result	Longint	←	A set of 32 bit flags

Discussion

OT GetOptions returns a 32-bit number which contains bits representing the different options for ObjectTools. You can use the 4D *??* bit test operator to test the state of a given option and the ?+ (bit set) or ?- (bit clear) operators to set or clear individual options.

Currently, there are two options defined. ObjectTools provides named constants for testing the options.

Option	Bit	Default
OT FailOnItemNotFound	0	0 (off)
OT ExactTagMatch	1	0 (off)
OT AutoCreateObjects	2	1 (on)
OT VariantItems	3	0 (off)

OT FailOnItemNotFound

By default, if an item cannot be found, the *OT Get<type>* routines will return a default value. If the *OT FailOnItemNotFound* option is set, a default value will still be returned but ObjectTools will generate an error and set *OK* to zero.

OT ExactTagMatch (deprecated)

Because of changes in the way objects are stored, this option is no longer supported.

OT AutoCreateObjects (new in version 2.0)

Previous to ObjectTools 2.0, to add an embedded objects to another object, you had to create the embedded object, put it, then clear it. This was tedious and ultimately unnecessary.

ObjectTools 2.0 will auto-create embedded objects by default whenever you put an item with a tag that contains embedded item references. For example:

```
C_LONGINT($obj)
$obj:=OT New
OT PutString ($obj;"one.two.three";"way cool!")
```

In the above example, after the call to *OT PutString* $obj contains the embedded object "one", which contains the embedded object "two", which contains a string "three".

This facility will make it considerably easier to create complex object hierarchies than in ObjectTools 1.x.

OT VariantItems (new in version 2.0)

By default, if you try to put a value into an item of a different type, an error is generated. The rationale behind this behavior is to prevent unintended changing of the type by carelessness or negligence.

However, what if you *want* to change the type of items? If such is your desire, you can now do so by setting the *OT VariantItems* option. For example:

```
C_LONGINT($obj)
$obj:=OT New
OT PutString($obj;"test";"way cool!")
OT PutLong($obj;"test";7)  `This generates an error

`Set the flag to allow variant item types and try again
OT SetOptions(OT GetOptions | OT VariantItems)
OT PutLong($obj;"test";7)  `This will work, the item is now a
longint
```

See Also

OT SetOptions

OT GetVersion

version 1

OT GetVersion → Text

Parameter	Type		Description
Function result	Text	←	The current version of ObjectTools

Discussion

OT GetVersion returns a textual representation of the current numeric version of ObjectTools, along with information about the platform and build type.

OT Register

version 1
modified version 2.5

OT Register(inSerialNum) → Longint

Parameter	**Type**		**Description**
inSerialNum	Text	→	ObjectTools serial number
Function result	Longint	←	Result code

Description

OT Register registers your serial number with ObjectTools. If *inSerialNum* is valid, 1 is returned, otherwise zero is returned.

If *OT Register* is not called or is called with an incorrect serial number, ObjectTools will timeout after 15 minutes of use. Once ObjectTools has timed out, the next call to ObjectTools will cause an ObjectTools error to be generated, and subsequent calls will have no effect or will return an empty value.

OT SetErrorHandler

version 1
modified v5.0r1

OT SetErrorHandler(inNewHandler) → Text

Parameter	Type		Description
inNewHandler	Text	→	Name of a 4D method to execute
Function result	Text	←	The name of the previous error handler

Discussion

OT SetErrorHandler sets the action to perform when ObjectTools encounters an error. The previous error handler is returned.

By default, action is taken when an error occurs.

If you pass the name of an existing 4D method in *inHandler*, that method will get called when an error occurs. The method must take four parameters:

- **message (C_TEXT):** A description of the error that occurred.
- **method (C_TEXT):** The name of the ObjectTools method that was called when the error occurred.
- **object (C_LONGINT):** The longint reference of the object being operated on when the error occurred. If the error does not involve an object, this will be zero.
- **tag (C_TEXT):** The tag of the object item being referenced when the error occurred. If the error does not involve a tag, this will be empty.

Warning: If you are upgrading to ObjectTools 5 from a previous version, you must be sure to add the extra two parameters to your error handler methods.

Note: If you put a TRACE statement at the end of your error handler method, when an error occurs the 4D debugger will come up. If you then step one line, you will be at the line after the one that caused the error.

Whether or not an error handler is set, whenever an error occurs the *OK* variable is set to zero.

OT SetErrorHandler returns the old handler so that you may dynamically change the error handling within your code.

OT SetOptions

version 1
modified version 1.6

OT SetOptions(inOptions)

Parameter	**Type**		**Description**
inOptions	Longint	→	Set of 32 bit flags

Discussion

OT SetOptions sets all of the ObjectTools options using a 32-bit number, which contains bits representing the different options.

Because all of the options are set at once, this call should be preceded by a call to *OT GetOptions*, then the 4D bitwise operators should be used to set or clear individual bit flags.

See “OT GetOptions” on page 108 for a list of the current options.

See Also

OT GetOptions

INDEX OF COMMANDS