ObjectScript Reference

Note: In all examples below, words in italics are placeholders for actual values or variables.

Object/SQL Basics (calling methods, using properties and parameters)

Action	Code
Call a class method. **Note**: Place a period before each pass-by-reference argument.	`do ##class(package.class).method(arguments)` `set variable = ##class(package.class).method(arguments)`
Call an instance method. **Note**: Place a period before each pass-by-reference argument.	`do object.method(arguments)` `set variable = object.method(arguments)`
Write or set a property.	`write object.property` `set object.property = value`
Set a serial (embedded) property.	`set object.property.embeddedProperty = value`
Retrieve a stored property value of object directly.	`set variable = ##class(package.class).PropertyGetStored(id)`
Determine if a property value exists in index.	`set exists = ##class(package.class).IndexNameExists(value, .id)`
Write a class parameter.	`write ..#PARAMETER` `write ##class(package.class).#PARAMETER`

Object/SQL Basics (handling objects)

Action	Code
Create a new object.	`set object = ##class(package.class).%New()`
Open an existing object by ID.	`set object = ##class(package.class).%OpenId(id, concurrency, .status)`
Open an existing object by unique index value.	`set object = ##class(package.class).IndexNameOpen(value, concurrency, .status)`
Close an object (remove from process).	`set object = ""`
Save an object.	`set status = object.%Save()`
Retrieve the ID of a saved object.	`set id = object.%Id()`
Reload stored properties of object.	`do object.%Reload()`
Display all properties of an object.	`do $system.OBJ.Dump(object)` `zwrite object`
List all objects in process.	`do $system.OBJ.ShowObjects()`
Determine if an object was modified in memory. **Note:** returns 1 (true), 0 (false)	`if object.%IsModified()`
Link two objects.	`set object1.referenceProperty = object2`

Delete an existing object by ID.	`set status = ##class(package.class).%DeleteId(id)`
Delete an existing object by unique index value.	`set status = ##class(package.class).IndexNameDelete(value)`
Delete all saved objects of a class.	`do ##class(package.class).%DeleteExtent()` `do ##class(package.class).%KillExtent()`
Populate a class with new objects.	`do ##class(package.class).Populate(count, verbose)`
Clone an object.	`set clonedObject = object.%ConstructClone()`
Determine if variable is an object reference. **Note**: returns 1 (true), 0 (false).	`if $isobject(variable)`
Find classname of an object.	`set variable = $classname(oref)`
Retrieve the OID of a saved object.	`set oid = object.%Oid()`

Object/SQL Basics (validation, handling statuses and exceptions)

Action	Code
Validate an object without saving.	`set status = object.%ValidateObject()`
Validate a property without saving.	`set status = ##class(package.class).PropertyIsValid(object.property)`
Print status after error.	`do $system.Status.DisplayError(status)` `write $system.Status.GetErrorText(status)`
Convert status into exception.	`set ex = ##class(%Exception.StatusException).CreateFromStatus(status)`

Object/SQL Basics (SQL, IDE code completion)

Action	Code
Declare a variable's type for IDE code completion.	`#dim object as package.class`
Start the SQL shell.	`do $system.SQL.Shell()`
Check SQL privileges. **Note:** returns 1 (true), 0 (false)	`if $system.SQL.Security.CheckPrivilege()`

ObjectScript Commands

Command	Purpose
`Continue`	Stop current loop iteration, continue looping.
`Do`	Execute method, procedure, or routine.
`For {}, While {}, Do {} While`	Execute block of code repeatedly.
`Halt`	Stop process and close the Terminal.
`If {} ElseIf {} Else {}`	Evaluate conditions and branch.
`Kill`	Destroy variable(s). Remove all objects in process.
`Quit, Return`	Terminate method, procedure, or routine. Optionally return value to calling method. Terminate loops.
`Set`	Set value of variable.
`Try {} Catch {}, Throw`	Handle errors.
`Write`	Display text strings, value of variable or expression.
`ZWrite`	Display array, list, bit string, JSON object/array (v2019.1+)

ObjectScript Date/Time Functions and Special Variables

Action	Command
Date conversion (internal to external)	`$zdate(`*`internalDate, format`*`)`
Date conversion (external to internal)	`$zdateh("`*`mm/dd/yyyy`*`")`
Time conversion (internal to external)	`$ztime(`*`internalTime, format`*`)`
Time conversion (external to internal)	`$ztimeh("`*`hh:mm:ss`*`")`
Current local date/time string	`$horolog`
Current UTC date/time string	`$ztimestamp`

ObjectScript Branching Functions

Action	Command
Return result for value of expression.	`$case(`*`expression, value1:result1, value2:result2, …, :result`*`)`
Return result for first true condition.	`$select(`*`condition1:result1, condition2:result2, …, 1:resultN`*`)`

ObjectScript String Functions

Action	Command
Extract characters from string.	`$extract(`*`string, start, end`*`)`
Right-justify string within *width* characters.	`$justify(`*`string, width`*`)`
Retrieve length of string.	`$length(`*`string`*`)`
Replace/remove substring in string.	`$replace(`*`string, searchString, replaceString`*`)`
Replace/remove characters in string.	`$translate(`*`string, searchChars, replaceChars`*`)`
Reverse a string.	`$reverse(`*`string`*`)`

ObjectScript List and Delimited String Functions

Action	Command
Build a list.	`set list = $listbuild(substrings separated by comma)`
Retrieve substring from list.	`$list(list, position)`
Put substring into list.	`set $list(list, position) = substring`
Retrieve number of substrings in list.	`$listlength(list)`
Convert a delimited string into a list.	`set list = $listFromString(string, delimiter)`
Convert a list into a delimited string.	`set string = $listToString(list, delimiter)`
Retrieve piece from delimited string.	`$piece(string, delimiter, pieceNumber)`
Set piece into delimited string.	`set $piece(string, delimiter, pieceNumber) = piece`
Retrieve number of delimited pieces in string.	`$length(string, delimiter)`

ObjectScript Existence Functions

Action	Command
Determine if variable exists.	`$data(variable)`
Return value of existing variable, or default.	`$get(variable, default)`
Return next valid subscript in sparse array.	`$order(array(subscript))`

Additional ObjectScript Functions

Action	Command
Increment ^global by 1 (or increment).	`$increment(^global, increment)` `$sequence(^global)`
Match a regular expression.	`$match(string, regularexpression)`
Generate a random integer. **Note**: start through (start+count-1)	`$random(count) + start`

ObjectScript Special Variables

Action	Command
Process ID	`$job`
Current namespace	`$namespace`
Security roles	`$roles`
Security username	`$username`

Utilities

Action	Command

Change namespace.	`set $namespace = "namespace"`
	`do ^%CD`
	`znspace "namespace"`

Display a global.	`do ^%G` `zwrite global`

List Collections

Action	Command
Create a new standalone list.	`set listObject=##class(%ListOfDataTypes).%New()`
Work with a list property.	Use methods below on a list collection property.
Insert an element at the end of a list.	`do listObject.Insert(value)` `do object.listProperty.Insert(value)`
Insert an element into a list.	`do listObject.SetAt(value, position)` `do object.listProperty.SetAt(value, position)`
Remove an element from a list.	`do listObject.RemoveAt(position)` `do object.listProperty.RemoveAt(position)`
Retrieve an element of a list.	`set variable = listObject.GetAt(position)` `set variable = object.listProperty.GetAt(position)`
Retrieve the size of a list.	`set variable = listObject.Count()` `set variable = object.listProperty.Count()`
Clear all the elements of a list.	`do listObject.Clear()` `do object.listProperty.Clear()`

Array Collections

Action	Command
Create a new standalone array.	`set arrayObject=##class(%ArrayOfDataTypes).%New()`
Work with an array property.	Use methods below on an array collection property.
Insert an element into an array.	`do arrayObject.SetAt(value, key)` `do object.arrayProperty.SetAt(value, key)`
Remove an element from an array.	`do arrayObject.RemoveAt(key)` `do object.arrayProperty.RemoveAt(key)`
Retrieve an element of an array.	`set variable = arrayObject.GetAt(key)` `set variable = object.arrayProperty.GetAt(key)`
Retrieve next key and its element. **Note**: place a period before the pass-by-reference `key` argument.	`set variable = arrayObject.GetNext(.key)` `set variable = object.arrayProperty.GetNext(.key)`
Retrieve the size of an array.	`set variable = arrayObject.Count()` `set variable = object.arrayProperty.Count()`
Clear all elements of an array.	`do arrayObject.Clear()` `do object.arrayProperty.Clear()`

Relationships

Action	Command
Parent-to-children object linking	`do parentObject.childRefProperty.Insert(childObject)` `set childObject.parentRefProperty = parentObject`
One-to-many object linking	`do oneObject.manyRefProperty.Insert(manyObject)` `set manyObject.oneRefProperty = oneObject`
Retrieve a property of a child object.	`set variable = parentObject.childRefProperty.GetAt(position).property`
Retrieve a property of a many object.	`set variable = oneObject.manyRefProperty.GetAt(position).property`
Retrieve next key.	`set key = parentObject.childRefProperty.Next(key)` `set key = oneObject.manyRefProperty.Next(key)`
Retrieve the count of child/many objects.	`set variable = parentObject.childRefProperty.Count()` `set variable = oneObject.manyRefProperty.Count()`
Open a many/child object directly.	`set object = ##class(package.class).IDKEYOpen(parentID, childsub`
Retrieve the id of a child object.	`set status = ##class(package.class).IDKEYExists(parentID, childsub, .childID)`
Clear the child/many objects.	`do parentObject.childRefProperty.Clear()` `do oneObject.manyRefProperty.Clear()`

Streams

Action	Command

Create a new stream.	`set streamObject=##class(%Stream.GlobalCharacter).%New()`
	`set streamObject=##class(%Stream.GlobalBinary).%New()`
	
	(or use methods below on a stream property)

Add text to a stream.	`do streamObject.Write(text)` `do object.streamProperty.Write(text)`
Add a line of text to a stream.	`do streamObject.WriteLine(text)` `do object.streamProperty.WriteLine(text)`
Read len characters of text from a stream.	`write streamObject.Read(len)` `write object.streamProperty.Read(len)`
Read a line of text from a stream.	`write streamObject.ReadLine(len)` `write object.streamProperty.ReadLine(len)`
Go to the beginning of a stream.	`do streamObject.Rewind()` `do object.streamProperty.Rewind()`
Go to the end of a stream, for appending.	`do streamObject.MoveToEnd()` `do object.streamProperty.MoveToEnd()`
Clear a stream.	`do streamObject.Clear()` `do object.streamProperty.Clear()`
Display the length of a stream.	`write streamObject.Size` `write object.streamProperty.Size`

Unit Testing Macros

Action	Command
Assert equality.	`do $$$AssertEquals(value1, value2, message)`
Assert inequality.	`do $$$AssertNotEquals(value1, value2, message)`
Assert status is OK.	`do $$$AssertStatusOK(status, message)`
Assert status is not OK.	`do $$$AssertStatusNotOK(status, message)`
Assert condition is true.	`do $$$AssertTrue(condition, message)`
Assert condition is not true.	`do $$$AssertNotTrue(condition, message)`
Log that assertion was skipped.	`do $$$AssertSkipped(message)`
Log message.	`do $$$LogMessage(message)`

Other Macros

Action	Command
Return a good status.	`return $$$OK`
Return an error status.	`return $$$ERROR($$$GeneralError, message)`
Check if status is good.	`if $$$ISOK(status)`
Check if status is an error.	`if $$$ISERR(status)`
Throw an exception if status is an error.	`$$$ThrowOnError(status)`
Return a null object reference.	`return $$$NULLOREF`
Place a new line within a string.	`write string1_$$$NL_string2`