

ObjectScript Reference

Note: words in *italics* in the examples below are placeholders for actual values or variables.

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

• Call a class method	do ##class(package.class).method(arguments) set variable = ##class(package.class).method(arguments) Note: place a . before each pass-by-reference argument
• Call an instance method	do object.method(arguments) set variable = object.method(arguments) Note: place a . before each pass-by-reference argument
• Write or set a property	write object.property set object.property = value
• Set a serial (embedded) property	set object.property.embeddedProperty = value
• Retrieve stored property value of object directly	set variable = ##class(package.class).PropertyGetStored(id)
• Determine if property value exists in index	if ##class(package.class).IndexNameExists(value, .id) Note: returns 1 (true), 0 (false)
• Write a class parameter	write ..#PARAMETER write ##class(package.class).#PARAMETER

Object/SQL Basics (handling objects)

• 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	if object.%IsModified() Note: returns 1 (true), 0 (false)
• 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	if \$isObject(variable) Note: returns 1 (true), 0 (false)
• 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)

• 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)

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

ObjectScript Reference

Note: words in *italics* in the examples below are placeholders for actual values or variables.

ObjectScript Commands

• 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 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

• Date conversion (internal → external)	<code>\$zdate(internalDate, format)</code>
• Date conversion (external → internal)	<code>\$zdateh("mm/dd/yyyy")</code>
• Time conversion (internal → external)	<code>\$ztime(internalTime, format)</code>
• Time conversion (external → internal)	<code>\$ztimeh("hh:mm:ss")</code>
• Current local date/time string	<code>\$horolog</code>
• Current UTC date/time string	<code>\$timestamp</code>

ObjectScript Branching Functions

• Return result for value of expression	<code>\$case(expression, value1:result1, value2:result2, ..., :result)</code>
• Return result for first true condition	<code>\$select(condition1:result1, condition2:result2, ..., 1:resultN)</code>

ObjectScript String Functions

• Extract characters from string	<code>\$extract(string, start, end)</code>
• Right-justify string within <i>width</i> characters	<code>\$justify(string, width)</code>
• Retrieve length of string	<code>\$length(string)</code>
• Replace/remove substring in string	<code>\$replace(string, searchString, replaceString)</code>
• Replace/remove characters in string	<code>\$translate(string, searchChars, replaceChars)</code>
• Reverse a string	<code>\$reverse(string)</code>

ObjectScript List and Delimited String Functions

• Build a list	<code>set list = \$listbuild(substrings separated by comma)</code>
• Retrieve substring from list	<code>\$list(list, position)</code>
• Put substring into list	<code>set \$list(list, position) = substring</code>
• Retrieve number of substrings in list	<code>\$listlength(list)</code>
• Convert a delimited string into a list	<code>set list = \$listfromstring(string, delimiter)</code>
• Convert a list into a delimited string	<code>set string = \$listtostring(list, delimiter)</code>
• Retrieve piece from delimited string	<code>\$piece(string, delimiter, pieceNumber)</code>
• Set piece into delimited string	<code>set \$piece(string, delimiter, pieceNumber) = piece</code>
• Retrieve number of delimited pieces in string	<code>\$length(string, delimiter)</code>

ObjectScript Existence Functions

• Determine if variable exists	<code>\$data(variable)</code>
• Return value of existing variable, or default	<code>\$get(variable, default)</code>
• Return next valid subscript in sparse array	<code>\$order(array(subscript))</code>

Additional ObjectScript Functions

• Increment ^global by 1 (or <i>increment</i>)	<code>\$increment(^global, increment)</code> <code>\$sequence(^global)</code>
• Match a regular expression	<code>\$match(string, regularexpression)</code>
• Generate random integer	<code>\$random(count) + start</code> Note: <i>start</i> through (<i>start</i> + <i>count</i> -1)

ObjectScript Reference

Note: words in *italics* in the examples below are placeholders for actual values or variables.

ObjectScript Special Variables

• Process ID	<code>\$job</code>
• Current namespace	<code>\$namespace</code>
• Security roles	<code>\$roles</code>
• Security username	<code>\$username</code>

Utilities

• Change namespace	<code>set \$namespace = "namespace"</code> <code>do ^%CD</code> <code>znspc "namespace"</code>
• Display a global	<code>do ^%G</code> <code>zwrite global</code>

List Collections

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

Array Collections

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

ObjectScript Reference

Note: words in *italics* in the examples below are placeholders for actual values or variables.

Relationships

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

Streams

• Create a new stream	set <i>streamObject</i> =##class(%Stream.GlobalCharacter).%New() set <i>streamObject</i> =##class(%Stream.GlobalBinary).%New() or use methods below on a stream property
• Add text to a stream	do <i>streamObject.Write(text)</i> do <i>object.streamProperty.Write(text)</i>
• Add a line of text to a stream	do <i>streamObject.WriteLine(text)</i> do <i>object.streamProperty.WriteLine(text)</i>
• Read <i>len</i> characters of text from a stream	write <i>streamObject.Read(len)</i> write <i>object.streamProperty.Read(len)</i>
• Read a line of text from a stream	write <i>streamObject.ReadLine(len)</i> write <i>object.streamProperty.ReadLine(len)</i>
• Go to the beginning of a stream	do <i>streamObject.Rewind()</i> do <i>object.streamProperty.Rewind()</i>
• Go to the end of a stream, for appending	do <i>streamObject.MoveToEnd()</i> do <i>object.streamProperty.MoveToEnd()</i>
• Clear a stream	do <i>streamObject.Clear()</i> do <i>object.streamProperty.Clear()</i>
• Display the length of a stream	write <i>streamObject.Size</i> write <i>object.streamProperty.Size</i>

Unit Testing Macros

• Assert equality	do \$\$\$AssertEquals(<i>value1, value2, message</i>)
• Assert inequality	do \$\$\$AssertNotEquals(<i>value1, value2, message</i>)
• Assert status is OK	do \$\$\$AssertStatusOK(<i>status, message</i>)
• Assert status isn't OK	do \$\$\$AssertStatusNotOK(<i>status, message</i>)
• Assert condition is true	do \$\$\$AssertTrue(<i>condition, message</i>)
• Assert condition isn't true	do \$\$\$AssertNotTrue(<i>condition, message</i>)
• Log that assertion was skipped	do \$\$\$AssertSkipped(<i>message</i>)
• Log message	do \$\$\$LogMessage(<i>message</i>)

Other Macros

• Return a good status	return \$\$\$OK
• Return an error status	return \$\$\$ERROR(\$\$\$GeneralError, <i>message</i>)
• Check if status is good	if \$\$\$ISOK(<i>status</i>)
• Check if status is an error	if \$\$\$ISERR(<i>status</i>)
• Throw an exception if status is an error	\$\$\$ThrowOnError(<i>status</i>)
• Return a null object reference	return \$\$\$NULLREF
• Place a new line within a string	write <i>string1_\$\$\$NL_string2</i>