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

Object/SQL Basics

• 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
• Determine if variable is an object reference	if $isobject(variable) Note: returns 1 (true), 0 (false)

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 InterSystems IRIS process and close Terminal.
• If {} ElseIf {} Else {}	Evaluate conditions and branch.
• Kill	Destroy variable(s).
• Quit, Return (v2013.1+)	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 string, bit string.

ObjectScript Branching Functions

• 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

• Extract characters from string	$extract(*string, start, end*)
• Right-justify string within *width* characters	$justify(*string, width*)
• Retrieve length of string	$length(*string*)
• Retrieve number of delimited pieces in string	$length(*string, delimiter*)
• Build a list string	set *listString* = $listbuild(*substrings separated by comma)*
• Retrieve substring from list string	$list(*listString, position*)
• Put substring into list string	set $list(*listString, position*) = *substring*
• Retrieve number of substrings in list string	$listlength(*listString*)
• Retrieve piece from delimited string	$piece(*string*, *delimiter, pieceNumber)*
• Set piece into delimited string	set $piece(*string*, *delimiter, pieceNumber) = piece*
• Replace/remove substring in string	$replace(*string, searchString, replaceString*)
• Reverse a string	$reverse(*string*)
• Replace/remove characters in string	$translate(*string, searchChars, replaceChars*)

ObjectScript Existence Functions

• Check 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*))

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

Additional ObjectScript Functions

• Increment ^global by increment	$increment(^*global, increment*) $sequence(^*global, increment)*
• Match a regular expression	$match(*string*, *regularexpression*)
• Generate random number	$random(*count*) + *start*) **Note:** *start* through (*start+count-1)*

Utilities

• Change namespace	do ^%CD znspace "*namespace*"
• Display a global	do ^%G zwrite *global*

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

List Collections

• 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

• 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 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

• 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 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()

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

Streams

• 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

• Assert equality	do $$$AssertEquals(*value1, value2, message)*
• Assert inequality	do $$$AssertNotEquals(*value1, value2, message)*
• Assert status is OK	do $$$AssertStatusOK(*status, message)*
• Assert status isn't OK	do $$$AssertStatusNotOK(*status, message)*
• Assert condition is true	do $$$AssertTrue(*condition, message)*
• Assert condition isn't true	do $$$AssertNotTrue(*condition, message)*
• Log message	do $$$LogMessage(*message)*

Other Macros

• Return a good status	quit $$$OK
• Return an error status	quit $$$ERROR($$$GeneralError, *message*)
• Check if status is good	if $$$ISOK(*status*)
• Check if status is an error	if $$$ISERR(*status*)
• Return a null object reference	quit $$$NULLOREF
• Place a new line within a string	write *string1*_$$$NL_*string2*