Module UNO.CLS (OpenOffice.org)

The object-oriented interface support for ooRexx is realized by calling or requiring the ooRexx module `UNO.CLS`, which defines public routines, classes and the environment symbol `.UNO` (a directory containing UNO objects). You can get at that support in one of two ways:

```
call UNO.CLS        /* make UNO-support for ooRexx available */
```

or

```
::requires UNO.CLS  /* make UNO-support for ooRexx available */
```

`UNO.CLS` is based upon the BSF4Rexx support for ooRexx and therefore requires the ooRexx module `BSF.CLS`. As a consequence all of the BSF4Rexx features are available as well.

Some of the UNO subfunctions are made available as instance methods of the proxy class `UNO_PROXY`, prepended with the string "`uno.`".

The ooRexx class `UNO_PROXY` is used for representing UNO Java (class) objects. Its instances are proxy objects which forward received messages to the Java side for invocation.

Although this module can be used for interfacing with OpenOffice, it generically supports interfacing with UNO and can as such be used to drive any UNO based application.

```
/* create desktop service object, get its Desktop interface object
   and its ComponentLoader interface object (to load dcouments)   */
ocl=UNO.createDesktop()~XDesktop~XComponentLoader

   /* define document URL, also file-, ftp- or http-URL possible  */
url = "private:factory/swriter" /* "swriter": text component      */
otc=ocl~loadComponentFromURL(url, "_blank", 0, .UNO~noProps)

   /* get text component's dcoument interface object and retrieve */
oText=otc~XTextDocument~getText  /* its text object               */
oText~setString("Hello world from ooRexx on" date("s") time())

   /* show services, i.e., the type of the component, interfaces  */
str=ppd("services: "    otc~uno.getServiceNames,  " ","0a"x)  "0a"x -
    ppd(" interfaces: " otc~uno.getInterfaceNames," ","0a"x)
.bsf.dialog~messageBox(str, "Services/Interfaces", "information")

::requires UNO.CLS    -- get UNO support
```

Public Routines

1. `decodeUrl(url)` returns the decoded `url` (all characters escaped as `%xy` sequences are replaced by their single characters)
2. `encodeUrl(url)` returns the encoded `url` (see definition of URL characters, those not allowed are escaped as `%xy` hex-strings)
3. `ppd(string [,delimiter] [,replacement])` returns the `string` in a form, where each `delimiter` (blank by default) is replaced by the `replacement` string (default: `line.separator || TAB`)
4. `uno.addPath([path] [,envVar])` adds and returns 'path' to environment variable 'envVar' (defaults to 'PATH')
5. `uno.areSame(unoProxy1, unoProxy2)` returns `.true`, if both UNO proxy objects refer to the same UNO object, `.false` else
6. `uno.connect([unoURL] [,xComponentContext])` returns the xContext if no object of the local installation (use it to retrieve its `ServiceManager`) or returns the remoteObject in case the optional `unoURL` was supplied; the optional `xComponentContext` allows to determine which (already established) connection to use
7. `uno.convertFromUrl(url)` returns the file encoded as an `url` as a fully qualified file name matching the rules of the host operating system
8. `uno.convertToUrl(url)` returns the platform dependent, fully qualified file name encoded as an `url`
9. `uno.createArray(…)` same arguments as `bsf.createArray()`, but returns an instance of `UNO_ARRAY_PROXY`, which makes sure that the array objects are wrapped up using the public routine `uno.wrap(…)`
10. `uno.createDesktop([xContext])` returns the local OpenOffice desktop object or the destkop object of the `xContext` argument, if supplied
11. `uno.createProperty(name[,value])` creates (and returns) a `PropertyValue` object and sets its `name` and `value`. If `value` is omitted, `.nil` is used.
12. `uno.findInterfaceWithMember(o, name [,[bString] [,howMany] )` searches the service object `o` for an interface that contains `name` as a member. Returns the interface object, if `bString` is `.false` (default) or the fully qualified UNOIDL interface name else. In the latter case `howMany` (default: `1`) determines how many interfaces (delimited with a blank) having a matching member should be returned; a value smaller than `1` returns all matching interfaces.
13. `uno.getCachedInterfaceName(name [,delimiter])` returns a string with the fully qualified, mixed-case UNO_IDL name of the interface denoted by `name`, which can be in uppercase and unqualified (the string after the last dot). Should there be multiple fully qualified interfaces matching an unqualified `name`, then the string contains them all delimited with the `delimiter` string (defaults to blank).
14. `uno.getCell(xSheet, nameAddress)` returns the (upper-left) cell object of the spreadsheet `xSheet` using an alphanumeric address (e.g. "`B5`")
15. `uno.getCell(xSheet, x, y)` returns the cell object of the spreadsheet `xSheet` using the 0-based column (`x`) and row (`y`) coordinates.
16. `uno.getDefinition(o)` returns a string which encodes all UNOIDL information (see table "`UNOIDL String Encodings`"). `o` can be a service object or an UNO_IDL string.
17. `uno.getInterfaceName(o)` returns the interface name of the UNO proxy `o`
18. `uno.getInterfaceNamesViaReflection(o)` returns a blank delimited string of the interface names that are defined for the service object `o` using the UNOIDL definitions via reflection
19. `uno.getProperties(o)` returns a blank delimited, encoded string (see table "`UNO_IDL Encodings`") with all defined properties for the service object `o`
20. `uno.getScriptContext(slotArg)` returns a UNO proxy, if the ooRexx script was invoked by OpenOffice, `.nil` else. The UNO proxy object has the following methods, returning context related UNO proxy objects:
    - `getDocument` (the document service object, an `XModel`),
    - `getDesktop` (the desktop service object, an `XDesktop`), and
    - `getComponentContext` (the context object, an `XComponentContext`).
21. `uno.getScriptContextVersion(slotArg)` returns a string denoting the ooRexx Script framework version
22. `uno.getScriptMetaData(slotArg)` returns a UNO_proxy, understanding: `getClassPath`, `getDescription`, `getLanguage`, `getLanguageName`, `getLanguageProperties`, `getLocation`, `getLogicalName`, `setLogicalName`, `getLocationPlaceHolder`, `getParcelLocation`, `getScriptFullURL`, `getShortFormScriptUR`, `getSourceURL`, `hasSource`, `loadSource`, `getSource`, `getSourceBytes`
23. `uno.getServiceNamesViaReflection(o)` returns a blank delimited string of the service names that are defined for the service object `o` using the UNOIDL definitions via reflection.
24. `uno.getScriptPath([scriptUri])` returns system path to script
25. `uno.getTypeName(o)` returns `o`'s UNO datatype name (see table "UNO Datatype Names")
26. `uno.getXTypeProviderTypeNames(o)` returns a blank delimited string of the interface names that the object `o` implements. Note: it is possible that not all implemented interfaces are reported by the object `o`! You can exploit the UNOIDL definitions instead using the routines `uno.getDefinition(o)` or `uno.getInterfaceNamesViaReflection(o)`.
27. `uno.loadClass(className [,idx])` imports and returns the `className` UNO class object; in addition the uno proxy gets stored in the `.UNO` directory using the (uppercased) `idx` as index (defaults to the unqualified `className`, i.e., the class name after the last dot).
28. `uno.queryInterfaceName(o, name)` returns the fully qualified interface name of `o` which contains `name` (can be unqualified and in any case) as a member, returns blank "" string, if not found
29. `uno.queryInterfaceObjectByName(o, name)` returns the interface object for `o` which contains `name` (can be unqualified and in any case) as a member, returns `.nil`, if not found
30. `uno.queryServiceName(o, name)` returns the fully qualified serivce name of `o` which contains `name` (can be unqualified and in any case) as a member, returns blank "" string, if not found
31. `uno.removePath([path] [,envVar])` removes and returns 'path'from environment variable 'envVar' (defaults to 'PATH')
32. `uno.setCell(xSheet, nameAddress, content)` sets the (upper-left) cell object of the spreadsheet `xSheet` using an alphanumeric address (e.g. "`B5`") using `setFormula(content)` which works for strings *and* formulas
33. `uno.setCell(xSheet, x, y, content)` sets the cell object of the spreadsheet `xSheet` using the 0-based column (`x`) and row (`y`) coordinates using `setFormula(content)` which works for strings *and* formulas
34. `uno.supportsService(o, serviceName)` returns `.true` if service object `o` supports `serviceName`, `.false` else
35. `uno.wrap(bsfObject)` returns an UNO proxy object, if `bsfObject` is a BSF (Java) proxy object

Class UNO_PROXY

This is the ooRexx proxy class for representing UNO Java proxy classes. ooRexx messages sent to its instances cause the invocation of the appropriate methods. Most of the methods starting with `uno.` are pass-through methods and their arguments (except for the first one, which is the `UNO_PROXY` object itsefl) are documented in the "`Public Routines`" section. This class is able to handle messages that are named after UNO interfaces (either the fully qualified name or the unqualified name, i.e., the name after the last dot; the unqualified name must start with the letter "`X`" to qualify as an interface name), returning the appropriate interface object.

UNO_PROXY's instance methods

1. `uno.bsfObject` returns the wrapped BSF proxy object
2. `uno.findInterfaceWithMember(name[,[bString][,howMany])` see public routine
3. `uno.getDefinition` see public routine
4. `uno.getInterfaceName` see public routine
5. `uno.getInterfaceNames` see public routine `...ViaReflection`
6. `uno.getProperties` see public routine
7. `uno.getServiceNames` see public routine `...ViaReflection`
8. `uno.getTypeName` see public routine
9. `uno.getXTypeProviderTypeNames` see public routine
10. `uno.isSame(otherProxyObject)` returns `.true`, if this proxy object is the same as `otherProxyObject`, `.false` else
11. `uno.queryInterfaceName(name)` see public routine
12. `uno.queryInterfaceObjectByName(name)` see public routine
13. `uno.queryServiceName` see public routine
14. `uno.supportsService` see public routine

Class UNO_ARRAY_REFERENCE

`UNO_ARRAY_REFERENCE` is a subclass of `UNO_PROXY` that allows interacting with Java array objects (stored in the BSF registry) as if they were ooRexx arrays (e.g. index values start with `1`, and the ooRexx array methods `AT`, `[]`, `DIMENSION`, `ITEMS`, `MAKEARRAY`, `PUT`, `[]=`, `SUPPLIER` are available). If returning an object from the array it will get wrapped up as an `UNO_PROXY`.

The public routine `uno.wrap` will use this class to create the ooRexx proxy object, if it detects that the supplied proxy object refers to an array object (i.e., it is an instance of the class `BSF_ARRAY_REFERENCE`).

Class UNO_DIRLIKE

`UNO_DIRLIKE` is the superclass for the public classes UNO_CONSTANTS and UNO_ENUM which allow easy access to the definitions either by name or value employing the ooRexx directory class semantics.

UNO_DIRLIKE's instance methods

1. `directory` returns a *copy* of the directory (`unoDirectory`) that stores all definitions
2. `entry(index)` returns the item associated with `index` or `.nil`, if `index` is not used
3. `hasEntry(index)` returns `.true` if an item is associated with `index`, `.false` else
4. `init(unoidl_className)` retrieves the UNOIDL defintions of `unoidl_className` and if successful, sends the message `setup` to the newly created instance (mplemented in the subclass), which sets up the `unoDirectory` and `unoNameQueue` accordingly
5. `makearray` returns an array of names in `unoNameQueue` order
6. `nameQueue` returns a *copy* of the queue (`unoNameQueue`) which contains the names in definition order
7. `supplier` returns a supplier where the index values follow the `unoNameQueue` order
8. *`unoDirectory`* returns the directory that stores all definitions, a *private* method (message needs to be sent to `self` to succeed)
9. `unoidl_definition` returns the string which encodes all UNOIDL information (see table "`UNO_IDL Encodings`")
10. `unoidl_name` returns the fully qualified UNOIDL name (a string)
11. `unoidl_typeName` returns the UNOIDL type name (a string, see table "UNO Datatype Names")
12. *`unoNameQueue`* returns the queue which contains the names in definition order, a *private* method (message needs to be sent to `self` to succeed)

Public Class UNO_CONSTANTS

`UNO_CONSTANTS` is a subclass of `UNO_DIRLIKE` which is able to store all defined constants in an ooRexx directory object. Sending the name of a constant to an instance of this class returns the associated numeric value or `.nil`, if the constant name is not defined. In addition it is possible to send the numeric value to it, which would return the constant's name or `.nil`, if no constant is defined for that value.

See also the public routines: `bsf.getConstant(JavaClassName, fieldName)` and `bsf.wrapStaticFields(unoidl_className)`

UNO_CONSTANTS' instance methods

1. `decode(number)` returns a blank delimited string listing the constant names that together yield `number`.
2. `encode(string)` returns a number representing the constants of the blank delimited `string`, which may consist of constant names, constant numeric values or a mixture of both.
3. `makestring` encodes all its constants as the required string value
4. *`setup`* *private* method which sets up the object by processing the UNOIDL definition of the constants, invoked via the superclass' constructor.

Public Class UNO_ENUM

`UNO_ENUM` is a subclass of `UNO_DIRLIKE` which is able to store all individual enumeration objects in an ooRexx directory object. Sending the name or its numeric value to an instance of this class returns the associated enum object or `.nil`, if the enumeration name is not defined. An enum object returned by this class possesses the methods `name` and `value`.

See also the public routines: `bsf.getConstant(JavaClassName, fieldName)` and `bsf.wrapStaticFields(unoidl_className)`

UNO_ENUM's instance methods

1. *`setup`* *private* method which sets up the object by processing the UNOIDL definition of the constants, invoked via the superclass' constructor.
2. `makestring` encodes all its enum values as the required string value

Environment Object .UNO (A Directory Object)

`UNO.CLS` will initialize a directory object accessible via the environment symbol `.UNO` to store important UNO/OOo objects. In addition it serves as a cache for interface class objects that have been instantiated while running an application as well as for classes that were loaded with the help of the public routine `uno.loadClass(unoidl_className [,idx])`. The following table lists the initial content of this directory object.

Index	Description
ANY	Class object `com.sun.star.uno.Any`
ANYCONVERTER	Class object `com.sun.star.uno.AnyConverter`
BAUTORESOLVE	If `.true`, then attempts to query interface to resolve a member (in case of a runtime error) and proceeds.
BEXTENDSEARCH	Boolean value determining whether reflection should exploit the UNOIDL definitions if interface not found in XTypeProvider list, preset to: `.true`
BOOTSTRAP	Class object com.sun.star.comp.helper.Bootstrap
NIL	UNO '.NIL', i.e. the field `com.sun.star.uno.Any.VOID`
NOPROPS	Empty array object of type com.sun.star.beans.PropertyValue; use, if property array must be given as an argument, but no properties need to be set.
PROPERTYVALUE	Class object `com.sun.star.beans.PropertyValue`
RGFREFLECTUNO	Class object org.oorexx.uno.RgfReflectUNO
UNORUNTIME	Class object `com.sun.star.uno.UnoRuntime`
VERSION	UNO.CLS version (a string)
XINTERFACES	Directory object containing a mapping of fully and unqualified interface names to their exact cased, fully qualified UNOIDL name.
XINTERFACES.DUPES	Relation object containing a mapping of unqualified interface names (which got already used in the XINTERFACES directory) to their exact cased, fully qualifed UNOIDL name (as they cannot be stored with the XINTERFACES directory).
XPROPERTYSETAUTOCASE	If `.true`, then the case of a property is not significant
XPROPERTYSETAUTOBOX	If .true, then autobox values in setPropertyValue()

Table "UNO Datatype Names"

The following names are derived from the names defined by the enum `com.sun.star.uno.TypeClass` and prepended with the string "`UNO_`".

"UNO_ANY"	"UNO_ENUM"	"UNO_MODULE"	"UNO_TYPE"
"UNO_ARRAY"	"UNO_EXCEPTION"	"UNO_PROPERTY"	"UNO_TYPEDEF"
"UNO_BOOLEAN"	"UNO_FLOAT"	"UNO_SEQUENCE"	"UNO_UNION"
"UNO_BYTE"	"UNO_HYPER"	"UNO_SERVICE"	"UNO_UNKNOWN"
"UNO_CHAR"	"UNO_INTERFACE"	"UNO_SHORT"	"UNO_UNSIGNED_HYPER"
"UNO_CONSTANT"	"UNO_INTERFACE_ATTRIBUTE"	"UNO_SINGLETON"	"UNO_UNSIGNED_LONG"
"UNO_CONSTANTS"	"UNO_INTERFACE_METHOD"	"UNO_STRING"	"UNO_UNSIGNED_SHORT"
"UNO_DOUBLE"	"UNO_LONG"	"UNO_STRUCT"	"UNO_VOID"

Table "Mapping UNO to Java Datatypes"

UNO Datatype	Java Datatype
UNO_ANY	com.sun.star.uno.Any or java.lang.Object
UNO_VOID	void
UNO_BOOLEAN	boolean
UNO_BYTE (8-bit)	byte
UNO_CHAR (16-bit)	char
UNO_SHORT (16-bit)	short
UNO_UNSIGNED_SHORT (16-bit)	short
UNO_LONG (32-bit)	int
UNO_UNSIGNED_LONG (32-bit)	int
UNO_HYPER (64-bit)	long
UNO_UNSIGNED_HYPER (64-bit)	long
UNO_FLOAT	float
UNO_DOUBLE	double

Table "UNOIDL String Encodings"

The following table defines the string encodings of the fundamental UNO datatypes as returned e.g. by the method (or public routine) `uno.getDefinition`. Definition groups are delimited by a blank "`␣`". Constituents of a definition group are delimited with a vertical bar "`|`", elements of a collection are delimited with a comma "`,`". Additional characters used as delimiters for parsing are highlighted in yellow. Items enclosed in square brackets ("`[]`") are optional and can be left out. An ellipses ("`…`") indicates that the preceding type/group may be repeated.

Encoding Definition
`UNO_CONSTANTS``|fully-qualified-name␣member-name|value|datatype…`
`UNO_ENUM``|fully-qualified-name|default-value␣member-name|value…` *Remark:* the individual values are always of type `UNO_LONG`.
`UNO_EXCEPTION``|fully-qualified-name␣member-name|datatype…`
`UNO_INTERFACE``|fully-qualified-name␣member-name|member-definition…` where "`member-definition`" is one of: • **`UNO_ATTRIBUTE`**`|[READONLY]|datatype…` • **`UNO_METHOD`**`|[ONEWAY]|retValue-datatype|[argName:datatype[,…]]|[exception[,…]]`
`UNO_MODULE``|fully-qualified-name␣member-name|UNO_Datatype…`
`UNO_SERVICE``|fully-qualified-name|[implName]␣memberName|definition…` where "`definition`" is one of: • **`UNO_INTERFACE`**`|[OPTIONAL]|defined_by_service` • **`UNO_SERVICE`**`|[OPTIONAL]|defined_by_service` • **`UNO_PROPERTY`**`|[modifier[,…]]|datatype|defined_by_service` *Remark:* if a service object is reflected that implements more than one service definition, than the "`fully-qualified-name`" of that compound service is created by concatenating all service names with the plus sign (+). Each of these constituting service definitions (if available via reflection) is then used to create the entire definition of that "compound service" object in hand, documenting all defined interfaces, services and properties.
`UNO_SINGLETON``|fully-qualified-name|[old-style-servicename]`
`UNO_STRUCT``|fully-qualified-name␣memberName|datatype…`
`UNO_TYPEDEF``|fully-qualified-name␣referenced-type|UNO-Type`