UTILITY ROUTINES AND UTILITY CLASSES FOR OBJECT REXX

Rony G. Flatscher

Department of Management and Information Systems

Vienna University of Economics and Business Administration

„8th International Rexx Symposium“, Heidelberg/Germany, April 22nd-24th, 1997

ABSTRACT

Over the past years, starting out with the beta-versions of Object Rexx in 1995, various utilities (routines and classes) have been devised. Some of them proofed useful, so the author wanted to share them with the Rexx community.

Among the utility routines one can find routines to sort collections like arrays, stems and directories etc., but also sort by the value part of some collections, where the sort values themselves get retrieved via dynamically invoked messages according to the message-names passed into the routine; determine (like with IBM's SOM) whether an object is a class object, an instance of a specific class and the like. Also some basic National Language support (upper/lowercasing, sorting) is documented.

.

1 INTRODUCTION

Object Rexx was introduced as a product in the fall of 1996 with IBM's PC operating system OS/2 Warp 4. In February 1997 a version for Windows 95 and Windows NT was introduced by IBM.

Object Rexx is backwardly compatible with Rexx, but enhances the language with powerful new features, some being independent of object-orientedness, yet most of the added power can be directly attributed to the innovative and powerful object oriented concepts, introduced into the language.

In the course of working with beta-versions of Object Rexx (started as early as in 1995) the author devised a couple of utility functions, classes and routines which should ease and enhance the development of Object Rexx programs.

Because of the wealth of the different utilties it became necessary to split the talk and the paper into two parts, where the second part draws sometimes on the features of the first one. The following Object Rexx utility modules with their public definitions are documented in this paper:

`get_answ.cmd`	get an option (for command line interface, aka "CLI")
`routine_find_file.cmd`	find a file along a path
`routine_ok.cmd`	turn *.true/.false* into strings
`routine_pp.cmd`	"cheap pretty print"
`routine_pp_number.cmd`	simple formatting of numbers
`routine_strip_quote.cmd`	simple strip/add quotes
`Is_Util.cmd`	Object Rexx test support
`nls_util.cmd`	National Language support ("NLS"), requires *Is_util.cmd*
`sort_util.cmd`	sort utilities for collections, requires *nls_util.cmd*
`rgf_util.cmd`	miscellaneous utilities and focal module for the following required modules: *rgf_class.cmd, routine_find_file.cmd, routine_ok.cmd, routine_pp.cmd, routine_pp_number.cmd, routine_strip_quote.cmd* and *sort_util.cmd*

The following Object Rexx utility modules with their public definitions are documented in a different paper[1)]:

[1)] Cf. [Flat97].

`class_rel.cmd`	some subclasses of *.Relation*
`routine_USIfy.cmd`	turn string into 7-Bit alphanumeric chars with embedded blanks being replaced with an underscore
`class_ref.cmd`	anchor/reference classes, allows for forward referencing; requires *class_rel.cmd* and *Routine_USIfy.cmd*
`sgmlEntity_util.cmd`	SGML-entity support to translate non 7-Bit characters according to the codepage into SGML-entity chars; requires *nls_util.cmd*
`html_util.cmd`	utilities and classes for creating HTML files, lists and tables; requires *class_ref.cmd, rgf_util.cmd* and *sgmlEntity_util.cmd*

All of these Rexx programs[2)] are made freely available via the Internet (e.g. cf. [W3Hobbes] or [W3RexxLA]). In the case that a particular interesting feature of Object Rexx is employed, it will get explained in detail. As a rule, short routines will get shown in full source code in this paper.

Thruout this paper it is assumed that the reader is acquainted with the basic principles of Object Rexx as layed out e.g. in the articles of [Flat96a] and [Flat96b], or in the books of [Ende97], [TurWah97], [VeTrUr96] and [WahHolTur97].[3)]

The syntax diagrams use square brackets ("[", "]") to denote optional arguments and curly brackets ("{", "}") to indicate that a mandatory argument has to be chosen from the list of arguments which themselves are delimited with a bar ("|"). Three dots ("ellipsis") indicate that the preceding construct may be repeated. An example:

```
xyz( arg1, { arg2 | arg3 }  [ , optional_arg4 ] ... )
```

arg1 is a mandatory argument to *xyz* followed by a comma followed by a mandatory argument (either *arg2* or *arg3*), followed optionally by one (*arg4*) or more optional arguments.

---

2) Please note that the term *program* in the context of this paper may be used interchangeably with the term *module*. A *module* is a *program*, but was intentionally created for being required by other Rexx programs via the `::REQUIRES` directive and as a rule would not be too useful to be executed on its own.

3) [W3ObjRexx] and [W3RexxLA] contain among other things articles and brief introductions into the Object Rexx language directly accessible via the Internet.

All examples make sure that all public definitions (routines and classes) of the described module are loaded by *CALL*ing it. Please note, that one usually would acquire access to those public definitions via the Object Rexx *":: REQUIRES"*[4] directive.

2 MODULE `GET_ANSW.CMD`

This module supports one single public routine: `get_answer()`.[5]

2.1 Public Routine `Get_Answer()`

Usage: *`Get_Answer( letterString [, [upperBound] [,Force]])`*

Returns what the user entered via the keyboard in a command line interface (aka "CLI"). Can be either an empty string ("") if just the *ENTER* key is pressed, or "1B"x for the *ESCAPE* key, or a single letter from the *letterString* (key-stroke), or an integer number with a value of 0 or larger, but not larger than the given *upperBound* argument.

The *BACKSPACE* key allows for editing an entered number with more than one digit, if *upperBound* is given. Whenever a user entry becomes unambiguous than the function automatically ends the entry mode and returns the received value. If just the *ENTER* key is pressed, an empty string is returned.

Arguments:

- ✓ ***letterString*** - string of valid letters for the user to enter, e.g. *"YN"* for either the key "Y", "y", "N" or "n".
- ✓ ***upperBound*** - optional positive number, the user may enter 0 thru the value of *upperBound*; e.g. if *upperBound* has a value of *32* then the user may enter any number between 0 and 32 inclusively.

---

[4] Cf. [Flat96a].

[5] *The code for this routine was extracted from the author's "showini.cmd"-package, a Rexx written package to display, edit, delete, backup and restore OS/2 INI-files. The program with the entire source may be found, e.g.* `<ftp://hobbes.nmsu.edu/os2/textutil/shwini32.zip>` *as of 1997-06-22.*

✓ ***Force*** - forces user to enter a valid value, i.e. pressing just the *ENTER* key without supplying a value is not allowed; this argument takes any argument, e.g. *"F".*

Example 1 (using *letterString*):

```
CALL get_answ        /* load "Get_Answ.cmd" public definitions          */

SAY "Do you want to erase drive A: ? (Y/N)"

answer = get_answer( "YN" ) /* "Y", "N", ENTER or ESCAPE                */
IF answer = "Y" THEN ...
```

Example 2 (using *letterString* and *Force*):

```
CALL get_answ        /* load "Get_Answ.cmd" public definitions          */

SAY "Do you want to erase drive A: ? (Y/N)"

answer = get_answer( "YN", , "F" )      /* "Y", "N" or ESCAPE           */
IF answer = "Y" THEN ...
```

Example 3 (using *upperBound*, i.e. a number has to be entered):

```
CALL get_answ        /* load "Get_Answ.cmd" public definitions          */

SAY "(1) add user"
SAY "(2) delete user"

answer = get_answer( , 2 )  /* 0, 1, 2, ENTER or ESCAPE                 */
IF answer = 1 THEN ...            /* add user  ...                      */
```

Example 4 (using *letterString* and *upperBound*, i.e. a letter or a number may be entered at the same time):

```
CALL get_answ        /* load "Get_Answ.cmd" public definitions          */

SAY "(1) add user"
SAY "(2) delete user"
SAY "(Q)quit, (A)bort immediately:"

answer = get_answer( "AQ", 2 )    /* 0. 1, 2, "A", "Q", ENTER or ESCAPE */
IF answer = 1 THEN ...            /* add user  ...                      */
```

3 MODULE `ROUTINE_FIND_FILE.CMD`

This module supports one single public routine: `find_file()`.

3.1 Public Routine `Find_File()`

Usage: *`Find_File( file_name [, [ path ] [, extensions] ] )`*

Returns the full path of the Rexx program denoted with *file_name* or empty string if not found. If *path* is given, the search is restricted to the given *path*, where each entry is

delimited by a semi-colon. If the third optional argument *extensions* is supplied then the routine will attempt to find the *file_name* by blindly appending each space separated *extensions*-item until the *file_name* is found or returns an empty string.

Arguments:

✓ ***file_name*** - any valid file name, e.g. *"html_util"*

✓ ***path*** - omitted or any valid path, e.g. *"c:\tools;d:\tmp\rexx"*; if omitted, then the environment variable *"PATH"* is used, prepended with a dot for the actual directory

✓ ***extensions*** - omitted or any extension which should be appended to *file_name* until the file is found, e.g. *".cmd .erx"*; if omitted defaults to the value *".orx .cmd .bat .rex .cls".*

Example 1:

```
CALL routine_find_file /* load "routine_find_file.cmd" public definitions*/
path = find_file( "html_util" )
```

... looks up the present directory and all directories given in the environment variable *"PATH"* for the file(s) *"html_util", "html_util.orx", "html_util.bat", "html_util.rex"* and *"html_util.cls"*. Returns first found file, if not found an empty string instead.

Example 2:

```
CALL routine_find_file /* load "routine_find_file.cmd" public definitions*/
path = find_file( "html_util", "d:\apps\os2\rexx\", ".orx .cls" )
```

... looks whether the file *"html_util", "html_util.orx"* or *"html_util.cls"* exists in the directory *"d:\apps\os2\rexx"*.

4 MODULE `ROUTINE_OK.CMD`

This module supports one single and simple public routine: `OK()`.

4.1 Public Routine `OK()`

Usage: *`OK( { .true | .false } )`*

Returns the string *`"*** not o.k. ! ***"`*, if the argument is *.false*, the string `"o.k."` in all other cases.

Source code of routine:

```
:: ROUTINE ok                      PUBLIC
   IF ARG(1) = 0 THEN RETURN "*** not o.k. ! ***"
                 ELSE RETURN "o.k."
```

Example 1:

```
CALL routine_ok            /* load "routine_ok.cmd" public definitions   */

SAY "our test is" ok( .false )
```

... yields the following output:

```
*** not o.k. ! ***
```

Example 2:

```
CALL routine_ok            /* load "routine_ok.cmd" public definitions   */

tmp = SysSetObjectData( "<WP_CLOCK>", "NOMOVE=YES" )
SAY "Making System Clock unmovable worked" ok( tmp )
```

5 MODULE ROUTINE_PP.CMD

This module supports one single and simple public routine: `PP()`.[6)]

5.1 Public Routine PP()

Usage: *pp(someText [, [leftBracket] [, rightBracket]])*

Returns the string value of the given argument *someText,* enclosed in given brackets. If the optional *leftBracket* is omitted the opening square bracket (*"["*) is used by default, if the optional *rightBracket* is omitted the closing square bracket (*"]"*) is used by default.

Source code of routine:

```
:: ROUTINE PP                      PUBLIC
   PARSE ARG argument, left_bracket, right_bracket

   IF left_bracket  = "" THEN left_bracket  = "["       /* default           */
   IF right_bracket = "" THEN right_bracket = "]"       /* default           */

   RETURN left_bracket || argument || right_bracket
```

Example:

```
CALL routine_pp      /* load "routine_pp.cmd" public definitions         */

SAY "Test:" pp( " " ) pp( " This is a test" )
```

... yields the following output (note the now visible blanks within the brackets):

```
Test: [ ] [ This is a test]
```

---

6) "*PP"* stands for "cheap" "**p**retty **p**rint".

6 MODULE ROUTINE_PP_NUMBER.CMD

This module supports one single and simple public routine: PP_Number().[7)]

6.1 Public Routine PP_Number()

Usage: *pp_number(someNumber)*

Returns the given number edited in US-style, e.g. the number *"123456.78"* will be returned as "123,456.78" by the routine.

Source code of routine:

```
:: ROUTINE PP_Number                    PUBLIC
   PARSE ARG number

   PARSE VAR number number "." fraction

   tmpResult = ''
   DO WHILE length( number ) > 3
      tmpResult = ',' || RIGHT( number, 3 ) || tmpResult
      number = LEFT( number, LENGTH( number ) - 3 )
   END

   IF fraction <> "" THEN tmpResult = tmpResult || "." || fraction

   RETURN number || tmpResult
```

Example:

```
CALL routine_pp_number  /* load "routine_pp_number.cmd" public definitions */

SAY "Test:" pp_number( 123456.78 )
```

... yields the following output:

```
Test: 123,456.78
```

[7)] The code for this routine was extracted from the *"DRIVE.RES"*-example of IBM's OS/2 product *"DrDialog"* as distributed with IBM's "Developer Connection" CD-ROM program since 1995.

7 MODULE ROUTINE_STRIP_QUOTE.CMD

This module supports one single and simple public routine: `Strip_Quotes`().

7.1 Public Routine `Strip_Quotes()`

Usage: `Strip_Quotes`*(string [, [bStrip] , [quote]])*

Returns the *string* with removed enclosing quotes (*bStrip = .true*) or returns the *string* embedded with quotes (*bStrip = .false*). Optional *quote* may be a single quote (') or a double quote ("), which serves as the default for embedding. The optional boolean value *bStrip* defaults to *.true*, i.e. to stripping quotes from the given string. In stripping mode *quote* is ignored; the quotes get stripped only, if the quote in position 1 of the string is a valid quote and present as the last character of that string also.

If the string contains the surrounding quote too then Rexx' rules for escaping quotes as part of a string are applied.

Source code of routine:

```
:: ROUTINE strip_quotes          PUBLIC
   USE ARG string, bStrip, quote

  bStrip = ( bStrip <> .false )   /* default to stripping quotes    */

  IF bStrip THEN                  /* strip quotes                   */
  DO
     string = STRIP( string )     /* remove leading/trailing blanks */
     quote  = LEFT( string, 1 )   /* determine quote                */

     IF POS( quote, '"' || "'" ) > 0 THEN /* check for quote        */
     DO
        IF RIGHT( string, 1 ) = quote THEN  /* closing quote present ? */
        DO
                /* remove surrounding quotes:    */
           string = SUBSTR( string, 2, LENGTH( string ) - 2 )
                /* take care of escaped quotes:  */
           string = CHANGESTR( quote || quote, string, quote )
        END
     END
  END
  ELSE           /* add quotes, duplicate existing quotes */
  DO
     IF \ VAR( "quote" ) THEN quote = '"'  /* default quote            */
           /* escape quotes */
     string = quote || CHANGESTR( quote, string, quote || quote ) || quote
  END
  RETURN string
```

Examples:

```
CALL routine_strip_quote /* load "routine_strip_quote.cmd" public definitions */
SAY strip_quotes( "This is a text.", .false, "'" )
```

... yields the following output:

```
'This is a text.'
```

```
SAY strip_quotes( "'This isn''t a text, is it?'", .true , "'" )
```

... yields the following output (note quote within text):

```
This isn't a text, is it?
```

```
SAY strip_quotes( "'This isn''t a text, is it?'",       , "'" )
```

... yields the following output (same as above, because of defaults):

```
This isn't a text, is it?
```

8 MODULE IS_UTIL.CMD

This module defines the following public routines: `IsA(), IsA2(), IsClassObject(), IsDescendedFrom(), IsInstanceOf()` and `get_methods_FROM_is_util()`.

For the purpose of enhancing any class with the appropriate Object Rexx methods, the following "floating methods"[8] are defined: `IsA, IsA2, IsClassObject, IsDescendedFrom, IsInstanceOf`. Invoked methods call the appropriate routines supplying *self* as their first argument; if necessary, the method argument will be given to the routine as the second argument. In order for programs which require *Is_Util.cmd* to access these floating methods the public routine `get_methods_FROM_is_util()` is defined.

8.1 Public Routine IsA()

Usage: `IsA( object, class_object )`

Returns *.true* if *class_object* is a superclass of *object*'s class, i.e. *object* is a direct or indirect instance of *class_object*, returns *.false* else. If *class_object* is not a class object, then a syntax error is raised.

Remark: Modelled after IBM's SOM *"somIsA()*".

---

[8] Cf. [Flat96a] and [Flat96b].

Examples:

```
CALL Is_Util         /* load "Is_Util.cmd" public definitions     */

tmpObject1 = .set ~ new           /* create a .set-object         */
tmpObject2 = .table ~ new         /* create a .table-object       */

SAY IsA( tmpObject1, .set )       /* returns .true                */
SAY IsA( tmpObject1, .table )     /* returns .true                */
SAY IsA( .set, .table )           /* returns .false               */
SAY IsA( .set, .table )           /* returns .false               */
SAY IsA( .set, tmpObject2 )       /* raises a syntax-error        */
```

8.2 Public Routine `IsA2()`

Usage: `IsA2( [class_]object1, [class_]object2 )`

Works like *IsA(),* except that either argument may be an object or a class object. If an argument is not a class object, then its class object is automatically used by sending the passed in object the "class" message. Therefore no syntax errors will be raised.

Returns *.true* if *class_object2* is identical to *class_object1*, or if *class_object2* is a superclass of *class_object1*, i.e. *class_object1* is a direct or indirect subclass of *class_object2*, returns *.false* else.

Examples:

```
CALL Is_Util         /* load "Is_Util.cmd" public definitions     */

tmpObject1 = .set ~ new           /* create a set-object          */
tmpObject2 = .table ~ new         /* create a table-object        */

SAY IsA2( tmpObject1, .set )      /* returns .true                */
SAY IsA2( tmpObject1, tmpObject2 )/* returns .true                */
SAY IsA2( tmpObject1, .table )    /* returns .true                */
SAY IsA2( .set, tmpObject2 )      /* returns .true                */
SAY IsA2( .set, .table )          /* returns .true                */
SAY IsA2( .set, .set )            /* returns .true                */

SAY IsA2( .set, .message)         /* returns .false               */
```

8.3 Public Routine `IsClassObject()`

Usage: `IsClassObject( object )`

Returns *.true* if the single argument *object* is a *class object*, i.e. it may be used to create instances of itself by sending it the *new*-message; returns *.false* else.

Examples:

```
CALL Is_Util         /* load "Is_Util.cmd" public definitions     */

tmpObject1 = .set ~ new           /* create a .set-object         */

SAY IsClassObject( .set )         /* returns .true                */
SAY IsClassObject( tmpObject1 )   /* returns .false               */
```

8.4 Public Routine `IsDescendedFrom()`

Usage: `IsDescendedFrom( class_object1, class_object2 )`

Returns *.true* if *class_object2* and *class_object1* are identical, or if *class_object2* is a superclass of *class_object1*, i.e. *class_object1* is a direct or indirect subclass of *class_object2*, returns *.false* else. Raises a syntax error if the passed in objects are not class objects.

Remark: Modelled after IBM's SOM *"somDescendedFrom()".*

Examples:

```
CALL Is_Util         /* load "Is_Util.cmd" public definitions           */

tmpObject1 = .set ~ new                 /* create a .set-object         */

SAY IsDescendedFrom( tmpObject1, .set ) /* raises a syntax-error        */
SAY IsDescendedFrom( .table, tmpObject1 )/* raises a syntax-error       */
SAY IsDescendedFrom( .set, .table )     /* returns .true                */
SAY IsDescendedFrom( .set, .set )       /* returns .true                */

SAY IsDescendedFrom( .table, .set )     /* returns .false               */
```

8.5 Public Routine `IsInstanceOf()`

Usage: `IsInstanceOf( object, class_object )`

Returns *.true* if *object* is an instance of *class_object*, returns *.false* else.

Remark: Modelled after IBM's SOM *"somInstanceOf()".*

Source code of routine:

```
:: ROUTINE IsInstanceOf                  PUBLIC
   USE ARG Object, Class_Object

   RETURN ( Object ~ class = Class_Object )
```

Examples:

```
CALL Is_Util         /* load "Is_Util.cmd" public definitions            */

tmpObject1 = .set ~ new                 /* create a set-object          */

SAY IsInstanceOf( tmpObject1, .set )    /* returns .true                */
SAY IsInstanceOf( .set, .set )          /* returns .false               */
SAY IsInstanceOf( .set, tmpObject1 )    /* returns .false               */
```

8.6 Public Routine `get_methods_FROM_is_util()`

Usage: `get_methods_FROM_is_util()`

Returns the *.methods*[9] directory containing all floating methods of *IS_Util.cmd*.

Source code of routine:

```
:: ROUTINE get_methods_FROM_is_util PUBLIC  /* return the method-directory */
   RETURN .methods          /* returns Is_util.cmd's .method directory     */
```

Example:

```
CALL Is_Util         /* load "Is_Util.cmd" public definitions              */

tmpUtilMethods = get_methods_FROM_is_util() /* get ".methods" directory    */

  /* create an instance of ".set" enhanced with the floating methods
     defined in Is_util.cmd:                                               */
tmpObject = .set ~ enhanced( tmpUtilMethods )

SAY tmpObject ~ IsA( .table )     /* test the IsA-method (.true)           */
SAY tmpObject ~ IsClassObject     /* test the IsClassObject-method (.false)*/
```

9 MODULE `NLS_UTIL.CMD`[10]

This module requires *Is_Util.cmd*. All documentation and examples pertaining to *NLS_UTIL.CMD* employs the following rule: if either the *country* or *codepage* arguments are omitted from routines or methods which need them, the default *country* and/or default *codepage* values of the session the program is running in, is used instead. Note, that for the purpose of retrieving the collating sequence table from the system (via the RexxUtil

[9] The special variable *.methods* is a directory containing all floating methods of a particular module. In order to be able to access it from the "outside" a public routine needs to be defined, which allows for establishing access to it.

[10] Starting with the description of the public utility routines, class and its public methods, the format of the documentation changes in order to preserve space and to keep the attention of the reader. The overview part will also contain the signatures and briefly document the purpose of the public utilities.

SysGetCollate-function) a syntax error may occur, if the given combination of country and codepage is not valid.

According to IBM's online documentation of the OS/2 control program APIs as of DevCon 12 (January 1997 edition) the following codepage settings and countries are available for OS/2:[11)]

		Codepage	
Country	**Country-Code**	Primary	Secondary
Asian English	099	437	850
Australia	061	437	850
Belgium	032	437	850
Canadian French	002	863	850
Czechoslovakia	042	852	850
Denmark	045	865	850
Finland	358	437	850
France	033	437	850
Germany	049	437	850
Hungary	036	852	850
Iceland	354	850	861
Italy	039	437	850
Japan	081	932	437, 850
Japan SAA	081	942	437, 850
Korea	082	934	437, 850
Korea SAA	082	944	437, 850
Latin America	003	437	850
Netherlands	031	437	850
Norway	047	865	850
People's Republic of China	086	936	437, 850
People's Republic of China SAA	086	946	437, 850
Poland	048	852	850
Portugal	351	860	850
Spain	034	437	850
Sweden	046	437	850
Switzerland	041	437	850
Taiwan	088	938	437, 850

[11)] According to the documentation some Asian codepages are available only on Asian versions of OS/2 and Asian versions of hardware resp. hardware drivers.

Taiwan SAA	088	948	437, 850
Turkey	090	857	850
United Kingdom	044	437	850
United States	001	437	850
Yugoslavia	038	852	850

In the initialization part of *NLS_util.cmd* a default object gets created using the session's *codepage* and *country* settings.

The *NLS_UTIL.CMD* module defines the following public routines:

`GET_NLS_DEFAULT_OBJECT()`

Returns the default *NLS*-object, also retrievable by sending the attribute class method *default_NLS* directly to the class object *.NLS*.

`SET_NLS_DEFAULT_OBJECT( [country] [,codepage] )`
`SET_NLS_DEFAULT_OBJECT( [NLS-object] )`

Sets the default *NLS*-object according to the arguments and returns the new default *NLS*-object.

`NLS_COLLATE( string )`

Returns the *string* argument translated according to the collating sequence of the default *NLS*-object.

`NLS_COMPARE( string1, string2 )`

Returns the values -1, 0, 1 indicating whether *string1* is smaller, equal or larger than *string2* according to the default *NLS*-object.

`NLS_LOWER( string )`

Returns the *string* argument translated to lowercase according to the default *NLS*-object.

`NLS_UPPER( string )`

Returns the *string* argument translated to uppercase according to the default *NLS*-object.

In addition this module defines a class *"NLS"* with the following public *class* and *instance* methods:

Class methods of class *NLS*:

`INIT` — Creates class-level attributes and initializes them.

`default_NLS` — Attribute method, allows access to the class-level attribute *"default_NLS"* containing the default *NLS*-object.

`entry( [country] [, codepage] )`

Returns the *NLS*-object created for the given *country* and *codepage* or *.nil* if not found (*.Directory* semantics).

`hasentry( [country] [, codepage] )`

Returns *.true*, if for the given *country* and *codepage* a *NLS*-object is available, *.false* else (*.Directory* semantics).

`setentry( [country] [, [codepage] [, object]] )`	Returns the entry (the *NLS*-object) for the given *country* and *codepage* and sets it to the passed in *object*. If no *object* is given, the entry is removed and the associated *NLS*-object is returned; if no entry is available then *.nil* (*.Directory* semantics) is returned.
`supplier`	Returns a supplier of all *NLS*-objects created so far.

Instance methods of class *NLS*:

`INIT( [ country ] [, codepage ] )`	Optional *country* and *codepage* as supplied as arguments for the *NEW*-message sent to the class object *.NLS*; creates and returns the appropriate *NLS*-object. If a *NLS*-object for the given *country* and *codepage* already exists, it is retrieved and returned.
`codepage`	Returns the codepage number.
`coll_lower`	Returns the significant lowercase letters for the NLS-related collating (sorting) sequence.
`coll_upper`	Returns the significant uppercase letters for the NLS-related collating (sorting) sequence.
`collating_table`	Returns all 256 characters of the *NLS*-object's *codepage*, where those characters are replaced with their uppercase representation which are regarded to be of equal value for determining the collating (sorting) sequence of that particular *country*.
`country`	Returns the country number.
`dump`	Dumps the lowercase, uppercase and collating sequence letters resp. letterings to `.error`.
`lowercase`	Returns all NLS lowercase letters.
`makestring`	Returns a string representing the *NLS* object.
`nls_collate`	Returns a string translated with the appropriate collating (sorting) sequence.
`nls_compare( string1, string2 )`	Returns the values -1, 0, 1 indicating whether *string1* is smaller, equal or larger than *string2* according to the *NLS*-object. [This method uses the RexxUtil supplied function named *SysNationalLanguage()*.]
`nls_lower( string )`	Returns a NLS lowercased string.
`nls_upper( string )`	Returns a NLS uppercased string.
`uppercase`	Returns all NLS uppercase letters.

The public routines work according to the default *NLS*-object stored with *.NLS* (the class object representing the NLS-class) in the class attribute *"default_NLS"*.[12)]

Each instance of this class will get stored with a class variable[13)], which is indirectly accessible by sending the (class) methods *ENTRY, HASENTRY, SETENTRY and SUPPLIER*-methods to the class object *.NLS*.

The output of the following two examples refers to a codepage setting of "850,437" and a country code of "043" (Austria).

Example 1:

```
CALL NLS_Util        /* load "NLS_Util.cmd" and all of its definitions   */

.nls ~~ new( 0, 850 ) ~~ new( 0, 437 )   /* create two NLS-objects       */
tmpSupp = .nls ~ supplier          /* get supplier for NLS-objects       */

DO WHILE tmpSupp ~ AVAILABLE
   tmpSupp ~ ITEM ~ dump           /* get object and let it dump itself  */
   tmpSupp ~ NEXT
END
```

... produces the following output for codepage settings "850, 437" and country "049"; note that the NLS-definitions for the default object (country=0 and codepage=0) are the same as for codepage 850 (country= 0 and codepage=850):

```
NLS-object:

   country: [0] codepage: [850]

lowercase: [abcdefghijklmnopqrstuvwxyzüéâäàåçêëèïîìæôöòûùøáíóúñãðõþý]
uppercase: [ABCDEFGHIJKLMNOPQRSTUVWXYZÜÉÂÄÀÅÇÊËÈÏÎÌÆÔÖÒÛÙØÁÍÓÚÑÃÐÕÞÝ]

effect of collating table:

 low (  1- 60): [abcdefghijklmnopqrstuvwxyzÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÿÖÜø£Ø áíó]
  up (  1- 60): [ABCDEFGHIJKLMNOPQRSTUVWXYZCUEAAAACEEEIIIAAEAAOOOUUYOUO$O$AIO]

 low ( 61- 97): [úñÑ¿¡«»ÁÂÀ¢¥ãÃ¤ðÐÊËÈ ÍÎÏÌÓßÔÒõÕÞÚÛÙýÝ]
  up ( 61- 97): [UNN?!""AAA$$AA$DDEEEIIIIIOSOOOOÞUUUYY]
```

[12)] If it becomes necessary that the routines use a specific NLS-object, then one merely needs to create the appropriate NLS-object first and have the class attribute *"default_NLS"* point to it, e.g.:

```
.NLS ~ default_NLS = .NLS ~ new( 041, 437 ) /* use a Swiss NLS-object instead */
```

[13)] This class variable is a directory for which the methods *ENTRY*, *HASENTRY* and *SETENTRY* have been specialized in order to serve the specific purpose of the NLS-class (the key to identify NLS-objects uniquely is built with two strings representing a *country* code and *codepage* code). In addition access to the *SUPPLIER* method is established by defining a pass-thru method at the class level. Note: it is not possible to get direct access to that class variable from outside of the class method level, which is named *nlsDirectory* (cf. the *EXPOSE*-keyword statements with some of the class method definitions) .

```
NLS-object:

   country: [0] codepage: [0]

lowercase: [abcdefghijklmnopqrstuvwxyzüéâäàåçêëèïîìæôöòûùøáíóúñãðõþý]
uppercase: [ABCDEFGHIJKLMNOPQRSTUVWXYZÜÉÂÄÀÅÇÊËÈÏÎÌÆÔÖÒÛÙØÁÍÓÚÑÃÐÕÞÝ]

effect of collating table:

 low (  1- 60): [abcdefghijklmnopqrstuvwxyzÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÿÖÜø£Ø áíó]
  up (  1- 60): [ABCDEFGHIJKLMNOPQRSTUVWXYZCUEAAAACEEEIIIAAEAAOOOUUYOUO$O$AIO]

 low ( 61- 97): [úñÑ¿¡«»ÁÂÀ¢¥ãÃ¤ðÐÊËÈ ÍÎÏÌÓßÔÒõÕÞÚÛÙýÝ]
  up ( 61- 97): [UNN?!""AAA$$AA$DDEEEIIIIIOSOOOOÞUUUYY]

NLS-object:

   country: [0] codepage: [437]

lowercase: [abcdefghijklmnopqrstuvwxyzüéâäàåçêëèïîìæôöòûùÿáíóúñ]
uppercase: [ABCDEFGHIJKLMNOPQRSTUVWXYZÜEAÄAÅÇEEEIIIÆOÖOUUYAIOUÑ]

effect of collating table:

 low (  1- 60): [abcdefghijklmnopqrstuvwxyzÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÿÖÜø£Ø× áí]
  up (  1- 60): [ABCDEFGHIJKLMNOPQRSTUVWXYZCUEAAAACEEEIIIAAEAAOOOUUYOU$$$$$AI]

 low ( 61- 69): [óúñÑ¿¡«»ß]
  up ( 61- 69): [OUNN?!""S]
```

Example 2:

```
CALL NLS_Util        /* load "NLS_Util.cmd" and all of its definitions   */

a = "Ärger über den Wölkchen auf dem Boot."    /* German NLS-string      */

SAY nls_upper( a )   /* returns "ÄRGER ÜBER DEN WÖLKCHEN AUF DEM BOOT."  */
SAY nls_lower( a )   /* returns "ärger über den wölkchen auf dem boot."  */
SAY nls_collate( a ) /* returns "ARGER UBER DEN WOLKCHEN AUF DEM BOOT."  */
```

10 MODULE `SORT_UTIL.CMD`

This module requires *NLS_Util.cmd* and defines the following public routines: `sort()`, `sortArray()`, `sortStem()`, `sortCollection()` and `BinFindArray()`.

For the purpose of determining whether a string is smaller, equal or larger than another string this module employs (and therefore requires) the module *NLS_Util.cmd*, namely the public routine *NLS_collate()* which uses the collating sequence of the default NLS-object.

10.1 Public Routine `sort()`

Usage: `sort(` *`CollObject [, option ] )`*

Returns a single-dimensioned array object containing all the string elements of *CollObject* in ascending sorted order. If option *option* is given and starts with the letter "*D*" then the sorting will be in descending order. *Sort* first renders the *CollObj* into a single dimensioned array or produces a copy in case a single dimensional array is given and invokes *sortArray()* to do the actual work.

Example:

```
CALL Sort_Util       /* load "Sort_Util.cmd" and all of its definitions  */

tmpObject = .set ~ of( "Hi", "there", "how", "are", "you?" )

tmpArr  = sort( tmpObject )       /* sort the strings in set             */
tmpArr2 = sort( tmpObject, "d" )  /* sort the strings in set descendingly*/

DO i = 1 TO tmpArr ~ ITEMS
   SAY "asc:" LEFT( tmpArr[ i ], 5 ) "| desc:" tmpArr2[ i ] /* show items*/
END
```

... yields the following output:

```
asc: are   | desc: you?
asc: Hi    | desc: there
asc: how   | desc: how
asc: there | desc: Hi
asc: you?  | desc: are
```

10.2 Public Routine `sortArray()`

Usage: `sortArray( array [, option ] )`

Sorts the single-dimensioned *array* object in place, containing the string elements in ascending sorted order. If option *option* is given and starts with the letter "*D*" then the sorting will be in descending order.

Example:

```
CALL Sort_Util       /* load "Sort_Util.cmd" and all of its definitions  */

tmpArr = .array ~ of( "Hi", "there", "how", "are", "you?" )

CALL sortArray tmpArr/* sort the strings in array                        */

DO i = 1 TO tmpArr ~ ITEMS
   SAY tmpArr[ i ]   /* show item                                        */
END
```

... yields the following output:

```
are
Hi
how
there
you?
```

10.3 Public Routine `sortStem()`

Usage: `sortStem( stem [, option ] )`

Note: this routine works with so-called *stem*-"arrays"[14] only! Such a *stem* object usually has positive integer number indices only, ranging from *1* to an upper bound stored with the stem itself at *stem.0*.

Sorts the stem-array in ascending sorted order. If option *option* is given and starts with the letter "*D*" then the sorting will be in descending order.

[14] "Stem-arrays" are employed by many utility functions of *RexxUtil.DLL* which is supplied with Object Rexx, e.g. the function *SysFileTree()*. The online documentation calls stems constructed in this way "stem variable collection".

Example:

```
CALL Sort_Util       /* load "Sort_Util.cmd" and all of its definitions  */

stem.1 = "Hi"        /* element 1                                        */
stem.2 = "there"     /* element 2                                        */
stem.3 = "how"       /* element 3                                        */
stem.4 = "are"       /* element 4                                        */
stem.5 = "you?"      /* element 5                                        */
stem.0 = 5           /* five elements in stem                            */

CALL sortStem stem.  /* sort the strings in set                          */

DO i = 1 TO stem.0
   SAY stem.i        /* show item                                        */
END
```

... yields the following output:

```
are
Hi
how
there
you?
```

10.4 Public Routine `sortCollection()`

Usage: `sortCollection( collection [, [message] [, "Descending" ]] )`

Returns a two-dimensional array sorted ascendingly by subscript 1 and the corresponding item of *collection* in subscript 2. If the optional argument *message* is given, then all items of *collection* are first sent that message and the result gets stored in subscript 1 (and the item in subscript 2). If the optional third argument starts with a "*D*", then the sort will be carried out descendingly.

Arguments:

✓ ***collection*** - any collection of items

✓ ***message*** - optional *message* argument which gets sent to each item in *collection* and may be one of the following three types:

- a message object,
- a plain string denominating the name of the message to be sent, or
- a single-dimensioned array object containing the string denominating the name of the message to be sent at subscript 1; optionally, at subscript 2 there may be another array object containing additional arguments for the method at subscript 1.

If a *collection* is a stem object and *message* is not given, then this routine will use the stem subscripts as the keys to sort by.

✓ ***"D[escending]"*** - sorts descendingly (only first character significant)

Example 1:

```
CALL Sort_Util       /* load "Sort_Util.cmd" and all of its definitions     */

tmpObj = .set ~ of( "Hi", "there", "how", "are", "you?" )

tmpArr  = sortCollection( tmpObj )      /* sort the strings in set         */
CALL dump tmpArr,  "1 - string value"

              /* sort collection with a message object                     */
msgObj = .message ~ new( "", "SUBSTR", "I", 3, 2 ) /* create message object */
tmpArr1 = sortCollection( tmpObj, msgObj )
CALL dump tmpArr1, "2 - message object for sending '~ SUBSTR( 3, 2 )'"

              /* sort collection with arguments in array object            */
tmpArrObj = .array ~ of( "SUBSTR", .array ~ of( 3, 2 )) /* create array object */
tmpArr2 = sortCollection( tmpObj, tmpArrObj )
CALL dump tmpArr2, "3 - array object for sending '~ SUBSTR( 3, 2 )'"

tmpArr3 = sortCollection( tmpObj, "REVERSE" )  /* use reversed strings      */
CALL dump tmpArr3, "4 - simple REVERSE-message"

:: ROUTINE dump
SAY ARG( 2 ); SAY          /* show title                                    */
DO i = 1 TO ARG( 1 ) ~ items / 2
   SAY "index" pp( ARG(1)[ i, 1 ] ) "item" pp( ARG( 1 )[ i, 2 ] )
END
SAY

:: ROUTINE pp; RETURN LEFT( "[" || ARG( 1 ) || "]", 10 )
```

... yields the following output:

```
1 - string value

index [are]      item [are]
index [Hi]       item [Hi]
index [how]      item [how]
index [there]    item [there]
index [you?]     item [you?]

2 - message object for sending '~ SUBSTR( 3, 2 )'

index [  ]       item [Hi]
index [e ]       item [are]
index [er]       item [there]
index [u?]       item [you?]
index [w ]       item [how]

3 - array object for sending '~ SUBSTR( 3, 2 )'

index [  ]       item [Hi]
index [e ]       item [are]
index [er]       item [there]
index [u?]       item [you?]
index [w ]       item [how]

4 - simple REVERSE-message

index [?uoy]     item [you?]
index [era]      item [are]
index [ereht]    item [there]
index [iH]       item [Hi]
index [woh]      item [how]
```

Example 2:

```
CALL Sort_Util       /* load "Sort_Util.cmd" and all of its definitions   */

stemObj.Hi           = "hallo"
stemObj.[ "there" ]  = "Du"
stemObj.how          = "wie"
stemObj.[ "are" ]    = "geht's"
stemObj.you          = "Dir?"

tmpArr1 = sortCollection( stemObj. )
CALL dump tmpArr1,  "1 - stem-object without a message (sort by subscript)"

tmpArr2 = sortCollection( stemObj. , "REVERSE" )
CALL dump tmpArr2, "2 - send REVERSE-message to stem.'s items, sort by results"

:: ROUTINE dump
SAY ARG( 2 ); SAY          /* show title                                    */
DO i = 1 TO ARG( 1 ) ~ items / 2
   SAY "index" pp( ARG(1)[ i, 1 ] ) "item" pp( ARG( 1 )[ i, 2 ] )
END
SAY

:: ROUTINE pp; RETURN LEFT( "[" || ARG( 1 ) || "]", 10 )
```

... yields the following output:

```
1 - stem-object without a message (sort by subscript)

index [are]      item [geht's]
index [HI]       item [hallo]
index [HOW]      item [wie]
index [there]    item [Du]
index [YOU]      item [Dir?]

2 - send REVERSE-message to stem.'s items, sort by results

index [?riD]     item [Dir?]
index [eiw]      item [wie]
index [ollah]    item [hallo]
index [s'theg]   item [geht's]
index [uD]       item [Du]
```

10.5 Public Routine `binFindArray()`

Usage: `BinFindArray( array, searchKey [ , bReturnArray ] )`

Returns the index in the one- or two-dimensional, ascendingly sorted array where the item with the given *searchKey* is stored with subscript 1; if there is no entry with the given *searchKey* then *.nil* is returned instead. If the optional *bReturnArray* is set to *.true*, then an array object is returned which contains the index for the found *searchKey* or *.nil* (if not found) in subscript 1, and the index into the array where the item was found or the index where the *searchKey* should be stored at in subscript 2.

Example:

```
CALL Sort_Util       /* load "Sort_Util.cmd" and all of its definitions  */

tmpObj = .set ~ of( "Hi", "there", "how", "are", "you?" )

tmpArr = sortCollection( tmpObj )
SAY "'there' is stored at:" binFindArray( tmpArr, "there" )

tmpArr2 = binFindArray( tmpArr, "how", .true )
SAY "'how'   is stored at:" tmpArr2[ 1 ] "should be stored at:" tmpArr2[ 2 ]

tmpArr2 = binFindArray( tmpArr, "oh", .true )
SAY "'oh'    is stored at:" tmpArr2[ 1 ] "should be stored at:" tmpArr2[ 2 ]
```

... yields the following output:

```
'there' is stored at: 4
'how'   is stored at: 3 should be stored at: 3
'oh'    is stored at: The NIL object should be stored at: 4
```

11 MODULE `RGF_UTIL.CMD`

This module defines the following public routines: `dump()`, `sayDebug()`, `sayError()` and `sayLog()`. It requires the following modules and as a result makes their public definitions available: `rgf_class.cmd`, `routine_find_file.cmd`, `routine_ok.cmd`, `routine_pp.cmd`, `routine_pp_number.cmd`, `routine_strip_quote.cmd` and `sort_util.cmd`.

Additionally, the initialization code sets up some variables with the directory object *rgf.util* stored with *.local:*

`.rgf.util ~ this_op_sys`	Operating system Object Rexx is running on
`.rgf.util ~ this_full_path`	Full path to *RGF_Util.cmd*
`.rgf.util ~ this_call_type`	Call type of *RGF_Util.cmd*
`.rgf.util ~ this_name`	The string: *"RGF_Util.cmd"*
`.rgf.util ~ debugLevel`	Usually set to *0*
`.rgf.util ~ indent`	String to prepend to arguments for the "*sayDebug()*"-, "*sayError()*"- and "*sayLog()*"-routines
`.rgf.util ~ Log`	Stream object for logging purposes (see "*sayLog()"*-routine*)*, default: *.stdout*
`.rgf.util ~ Error`	Stream object for error messages (see "*sayError()"*-routine*),* default: *.stderr*
`.rgf.util ~ Debug`	Stream object for debug messages (see "*sayDebug()*"-routine), default: *.stderr*

11.1 Public Routine `dump()`

Usage: `dump( collObject [, [title] [, [stream] [, indent]]] )`

Iterates over items in *collObject* and prints them.

Arguments:

✓ ***collObj*** - a collection of items

✓ ***title*** - optional title, defaulting to *"--- not given ---"*

✓ ***stream*** - optional stream object to write item to, defaults to *.rgf.util ~ debug*

✓ ***indent*** - optional string to prepend items before writing them to the *stream*, defaults to the string defined with *.rgf.util ~ indent*

Example:

```
CALL RGF_Util        /* load "RGF_Util.cmd" public definitions          */

tmpObject = .array ~ of( "Hi", "there", "how", "are", "you?" )
CALL dump tmpObject, "Dumping the object 'tmpObject'"
```

... yields the following output:

```
begin of dump ...-----------------------------------------------------
got [The Array class] title [Dumping the object 'tmpObject'] items [5]

index [1] item [Hi]
index [2] item [there]
index [3] item [how]
index [4] item [are]
index [5] item [you?]
end of dump ..-------------------------------------------------------
```

11.2 Public Routine `SayDebug(), SayError()` and `SayLog()`

Usage: `SayDebug( string ), SayError( string ), SayLog( string )`

Writes *string* (or string representation of an object) to the appropriate stream object, prepending it with the *.rgf.util ~ indent* string.

Source code of routines:

```
:: ROUTINE sayDebug              PUBLIC  /* write to debug-stream  */
   PARSE ARG arg
   .rgf.util ~ debug ~ lineout( .rgf.util~indent arg )

:: ROUTINE sayError              PUBLIC  /* write to error-stream  */
   PARSE ARG arg
   .rgf.util ~ error ~ lineout( .rgf.util~indent "***" arg )

:: ROUTINE sayLog                PUBLIC  /* write to log-stream    */
   PARSE ARG arg
   .rgf.util~log~lineout( .rgf.util~indent arg )
```

Example :

```
CALL RGF_Util        /* load "RGF_Util.cmd" public definitions    */

CALL sayDebug "This is sent to the debug stream:" .rgf.util ~ debug ~ current
CALL sayError "This is sent to the error stream:" .rgf.util ~ error ~ current
CALL sayLog   "This is sent to the log   stream:" .rgf.util ~ log ~ current
```

... yields the following output:

```
This is sent to the debug stream: STDERR
*** This is sent to the error stream: STDERR
This is sent to the log   stream: STDOUT
```

12 SUMMARY

This paper discussed and documented the public routines and public classes of the following modules in a brief manner: `get_answ.cmd`, `Is_Util.cmd`, `NLS_Util.cmd`, `RGF_Util.cmd`, `routine_find_file.cmd`, `routine_ok.cmd`, `routine_pp.cmd`, `routine_pp_number.cmd`, `routine_strip_quote.cmd` and `Sort_Util.cmd`. The source code of these utility modules should serve as the ultimate point of reference in case of unclearity.

Additional utilities for Object Rexx programs are presented in [Flat97]. The netnews newsgroup `<comp.lang.rexx>` should be used for discussing issues with respect to these utilities.

13 REFERENCES

[Ende97] Ender T.: "Object-Oriented Programming with REXX", John Wiley & Sons, New York et.al. 1997.

[Flat96a] Flatscher R.G.: "Local Environment and Scopes in Object REXX", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

[Flat96b] Flatscher R.G.: "Object Classes, Meta Classes and Method Resolution in Object REXX", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

[Flat96c] Flatscher R.G.: "ORX_ANALYZE.CMD - a Program for Analyzing Directives and Signatures of Object REXX Programs", in: Proceedings of the "7th International REXX Symposium, May 12-15, Texas/Austin 1996", The Rexx Language Association, Raleigh N.C. 1996.

[Flat97] Flatscher R.G.: "Utility Routines and Utility Classes for Object Rexx, Part II", in: Proceedings of the "8th International Rexx Symposium, April 22nd-24th, Heidelberg/Germany 1997", The Rexx Language Association, Raleigh N.C. 1997.

[TurWah97] Turton T., Wahli U.: "Object Rexx for OS/2 Warp", Prentice-Hall, London 1997.

[VeTrUr96] Veneskey G., Trosky W., Urbaniak J.: "Object Rexx by Example", Aviar, Pittsburgh 1996.

[WahHolTur97] Wahli U., Holder I., Turton T.: "Object REXX for Windows 95/NT, With OODialog", Prentice Hall, London 1997.

[W3Hobbes] URL (97-06-18): `http://hobbes.nmsu.edu/`

[W3ObjRexx] URL (97-06-18): `http://www2.hursley.ibm.com/orexx/`

[W3Rexx] URL (97-06-18): `http://www2.hursley.ibm.com/rexx/`

[W3RexxLA] URL (97-06-18): `http://www.RexxLA.org`

Additional information:

Online documentations for Object Rexx as delivered with OS/2 Warp 4 in the fall of 1996, the Object Rexx OS/2 developer edition as supplied with IBM's "Developer Connection" CD-ROM program between 1995 and 1997, and the Windows 95/NT products and developer editions as of February 1997 and June 1997.

Various postings on the internet newsgroup "comp.lang.rexx" between 1995 and 1997.

Date of Article: 1997-07-19.

Published in: Proceedings of the "8th International REXX Symposium", Heidelberg/ Germany, April 22nd-24th, 1997", The Rexx Language Association, Raleigh N.C. 1997.

Presented at: "8th International Rexx Symposium", Heidelberg/Germany, April 22nd-24th, 1997.