UTILITY ROUTINES AND UTILITY CLASSES FOR OBJECT REXX, PART II

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 e.g. the module *HTML_UTIL.CMD* which defines routines and classes to be used for creating HTML-documents containing tables, lists and supports cross-referencing. It is possible to use references for anchors which get defined later under program control, so in effect forward referencing becomes feasible.

For the code-pages 437 and 850 translation of the extended codes (above "7F"x) to and from SGML-entities has been defined already and can be easily extended for other codepages.

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 talks and the papers into two parts, where the second part draws sometimes on the features of the first one. The following Object Rexx modules serve as building blocks and are documented in a different paper[1]:

`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*

[1] Cf. [Flat97].

The following Object Rexx utility modules with their public definitions are documented in this paper:

`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 ] ... )
```

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.

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.

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 `CLASS_REL.CMD`

This module supports the following public classes which all are subclasses of Object Rexx' builtin *Relation* class: `RelTable`, `RelBijective`, `RelBijectiveSet` and `RelDir`. As a consequence of subclassing, all methods defined with *Relation* remain available. Most of the following classes are documented in [Flat96b] too.

2.1 Public Class `RelTable`

RelTable directly subclasses the Object Rexx class *Relation*. It overrides the methods `"[]="` and `"PUT"` in order to change the default behaviour to a table-like one, by allowing just one item per index. In contrast to the built-in *Table* class *RelTable* allows for determining whether a specific item is in the collection and navigating from an item to its index.

4) Cf. [Flat96a].

Example:

```
CALL class_rel /* load "Class_Rel.cmd" public definitions                */

tmpColl1 = .Relation ~ new  /* create an instance of the Relation class   */
tmpColl1[ 1 ] = "one"
tmpColl1[ 1 ] = "one - another one"
tmpColl1[ 2 ] = "two"
tmpColl1[ 3 ] = "two"
tmpColl1[ "two" ] = "this has the word 'two' as its index"
tmpColl1[ "two" ] = "another entry with an index of 'two'"

CALL dump tmpColl1, "Built-in instance of class Relation"

tmpColl2 = .RelTable ~ new  /* create an instance of the RelTable class       */
tmpColl2[ 1 ] = "one"
tmpColl2[ 1 ] = "one - another one" /* replaces previous entry! (index exists) */
tmpColl2[ 2 ] = "two"
tmpColl2[ 3 ] = "two"
tmpColl2[ "two" ] = "this has the word 'two' as its index"
tmpColl2[ "two" ] = "another entry with an index of 'two'"

CALL dump tmpColl2, "Instance of class RelTable, behaving like a table"
SAY "Index of '2' present in relTable:" tmpColl2 ~ HASINDEX( 2 )
SAY "Item 'two' associated with index:" tmpColl2 ~ ALLINDEX( 'two' )[ 1 ]

:: ROUTINE dump
   USE ARG coll

   SAY "Dumping" coll":"
   DO index OVER coll
      SAY "index:" LEFT("["index"]", 7) "item: ["coll[ index ]"]"
   END
   SAY LEFT( "", 40, "-" )
```

... yields the following output:

```
Dumping a Relation:
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
index: [2]     item: [two]
index: [3]     item: [two]
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
----------------------------------------
Dumping a RELTABLE:
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
index: [2]     item: [two]
index: [3]     item: [two]
----------------------------------------
Index of '2' present in relTable: 1
Item 'two' associated with index: 3
```

2.2 Public Class `RelBijective`

RelBijective directly subclasses the Object Rexx class *Relation*. It overrides the methods `"[]="` and `"PUT"` in order to change the default behaviour to a table-like one, by allowing just one item per index. In addition to the behaviour of *RelTable* an item must not occur more than once in the entire collection[5].

[5] This could be used to store married couples such, that a man and/or a woman may be married only once:

Example:

```
CALL class_rel /* load "Class_Rel.cmd" public definitions          */

tmpColl1 = .Relation ~ new  /* create an instance of the Relation class   */
tmpColl1[ 1 ] = "one"
tmpColl1[ 1 ] = "one - another one"
tmpColl1[ 2 ] = "two"
tmpColl1[ 3 ] = "two"
tmpColl1[ "two" ] = "this has the word 'two' as its index"
tmpColl1[ "two" ] = "another entry with an index of 'two'"

CALL dump tmpColl1, "Built-in instance of class Relation"

tmpColl2 = .RelBijective ~ new  /* create an instance of the RelTable class */
tmpColl2[ 1 ] = "one"
tmpColl2[ 1 ] = "one - another one"/* replaces previous entry! (index exists)*/
tmpColl2[ 2 ] = "two"
tmpColl2[ 3 ] = "two"        /* replaces previous entry! (item exists)  */
tmpColl2[ "two" ] = "this has the word 'two' as its index"
tmpColl2[ "two" ] = "another entry with an index of 'two'"

CALL dump tmpColl2, "Instance of class RelTable, behaving like a table"

:: ROUTINE dump
   USE ARG coll

   SAY "Dumping" coll":"
   DO index OVER coll
      SAY "index:" LEFT("["index"]", 7) "item: ["coll[ index ]"]"
   END
   SAY LEFT( "", 40, "-" )
```

... yields the following output:

```
Dumping a Relation:
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
index: [2]     item: [two]
index: [3]     item: [two]
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
----------------------------------------
Dumping a RELBIJECTIVE:
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
index: [3]     item: [two]
----------------------------------------
```

2.3 Public Class `RelBijectiveSet`

RelBijectiveSet directly subclasses the Object Rexx class *Relation*. It overrides the methods `"[]="` and `"PUT"` in order to change the default behaviour to a *RelBijective* one, by allowing just one item per index, and no duplicate items. In addition to the behaviour of *RelBijective* an index must not occur as an item, and an item must not occur as an index in the entire collection.[6)]

the `index` will be used for persons of one sex and the associated `item` for the other sex.

Example:

```
CALL class_rel /* load "Class_Rel.cmd" public definitions          */

tmpColl1 = .Relation ~ new  /* create an instance of the Relation class   */
tmpColl1[ 1 ] = "one"
tmpColl1[ 1 ] = "one - another one"
tmpColl1[ 2 ] = "two"
tmpColl1[ 3 ] = "two"
tmpColl1[ "two" ] = "this has the word 'two' as its index"
tmpColl1[ "two" ] = "another entry with an index of 'two'"

CALL dump tmpColl1, "Built-in instance of class Relation"

tmpColl2 = .RelBijectiveSet ~ new /* create an instance of the RelTable class */
tmpColl2[ 1 ] = "one"
tmpColl2[ 1 ] = "one - another one" /* replaces previous entry! (index exists) */
tmpColl2[ 2 ] = "two"
tmpColl2[ 3 ] = "two"               /* replaces previous entry! (item exists)  */

  /* the previous entry gets removed as "two" gets inserted (as an index) */
tmpColl2[ "two" ] = "this has the word 'two' as its index"
  /* the previous entry gets removed as "two" gets inserted (as an index) */
tmpColl2[ "two" ] = "another entry with an index of 'two'"

CALL dump tmpColl2, "Instance of class RelTable, behaving like a table"

:: ROUTINE dump
   USE ARG coll

   SAY "Dumping" coll":"
   DO index OVER coll
      SAY "index:" LEFT("["index"]", 7) "item: ["coll[ index ]"]"
   END
   SAY LEFT( "", 40, "-" )
```

... yields the following output:

```
Dumping a Relation:
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
index: [2]     item: [two]
index: [3]     item: [two]
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
----------------------------------------
Dumping a RELBIJECTIVESET:
index: [two]   item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
----------------------------------------
```

2.4 Public Class `RelDir`

RelDir directly mixinclasses the Object Rexx class Relation. This way RelDir may be added into the list of additional superclasses of any subclass of the class *Relation*. Unlike the previous classes *RelDir* does not override predefined methods, but adds the methods `"ENTRY", "SETENTRY", "HASENTRY"` and `"UNKNOWN"` and implements them in such a way that the semantics of the class *Directory* are introduced.

6) This can be used e.g. to create pairs from a set of tennis players for building a tennis tournament plan.

Source code of entire class definition:

```
:: CLASS RelDir         MIXINCLASS Relation     PUBLIC
:: METHOD  ENTRY
   USE ARG name
                            /* returns an item associated with name       */
   RETURN self ~ at( TRANSLATE( name ))

:: METHOD  HASENTRY  /* returns .true if entry exists, .false else       */
   USE ARG name

   RETURN self ~ hasindex( TRANSLATE( name ))

:: METHOD  SETENTRY  /* stores "value" with uppercased "name"            */
   USE ARG name, value

   self ~ PUT( value, TRANSLATE( name ))

:: METHOD  UNKNOWN   /* define an unknown method                         */
   USE ARG messageName, messageArgs

   IF RIGHT( messageName, 1 ) = "=" THEN /* trailing '=' in message name  */
   DO                /* remove trailing '=' and invoke 'SETENTRY' method */
      index = LEFT( messageName, LENGTH( messageName ) - 1 )
      FORWARD MESSAGE ( "SETENTRY" ) ARRAY ( index, messageArgs[ 1 ] )
   END
   ELSE              /* invoke 'ENTRY' method                            */
      FORWARD MESSAGE ( "ENTRY" )    ARRAY ( messageName )
```

A defined "UNKNOWN" method will get invoked by the Object Rexx interpreter if a message was received for which no method has been defined. The **uppercase** name of that unknown message is used as the index into the collection. Depending on the last character either the "ENTRY" or the "SETENTRY" (last character must be an equal sign '=') message is sent.

Example 1:

```
CALL class_rel /* load "Class_Rel.cmd" public definitions                 */

tmpColl1 = .RelDir ~ new    /* create an instance of the Relation class   */
tmpColl1[ 1 ] = "one"
tmpColl1[ 1 ] = "one - another one"
tmpColl1[ 2 ] = "two"
tmpColl1[ 3 ] = "TWO"
tmpColl1[ "two" ] = "this has the word 'two' as its index"
tmpColl1[ "two" ] = "another entry with an index of 'two'"
tmpColl1[ "three" ] = "another entry with an index of 'two'"
tmpColl1 ~ two = "hi"
tmpColl1 ~ two = "there"

CALL dump tmpColl1, "Built-in instance of class Relation"
SAY "tmpColl1 ~ 1    :" tmpColl1 ~ 1     /* return item associated with "1" */
SAY "tmpColl1 ~ two  :" tmpColl1 ~ two   /* return item associated with "TWO" */
SAY "tmpColl1 ~ three:" tmpColl1 ~ three /* no entry with "THREE" ! */

:: ROUTINE dump
   USE ARG coll

   SAY "Dumping" coll":"
   tmpSet = .set ~ new
   DO index OVER coll
      IF tmpSet ~ HASINDEX( index ) THEN ITERATE
      tmpSet ~ PUT( index )
      DO item OVER coll ~ allat( index )
         SAY "index:" LEFT("["index"]", 7) "item: ["item"]"
      END
   END
   SAY LEFT( "", 40, "-" )
```

... yields the following output:

```
Dumping a RELDIR:
index: [two]   item: [another entry with an index of 'two']
index: [two]   item: [this has the word 'two' as its index]
index: [TWO]   item: [there]
index: [TWO]   item: [hi]
index: [2]     item: [two]
index: [3]     item: [TWO]
index: [three] item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
index: [1]     item: [one]
----------------------------------------
tmpColl1 ~ 1    : one - another one
tmpColl1 ~ two  : there
tmpColl1 ~ three: The NIL object
```

Example 2 (show useful usage of multiple inheritance):

```
tmpColl1 = .Test ~ new            /* create an instance of the TEST class */
tmpColl1[ 1 ] = "one"
tmpColl1[ 1 ] = "one - another one"
tmpColl1[ 2 ] = "two"
tmpColl1[ 3 ] = "TWO"
tmpColl1[ "two" ] = "this has the word 'two' as its index"
tmpColl1[ "two" ] = "another entry with an index of 'two'"
tmpColl1[ "three" ] = "another entry with an index of 'two'"
tmpColl1 ~ two = "hi"
tmpColl1 ~ two = "there"

CALL dump tmpColl1, "Built-in instance of class Relation"
SAY "tmpColl1 ~ 1    :" tmpColl1 ~ 1
SAY "tmpColl1 ~ two  :" tmpColl1 ~ two
SAY "tmpColl1 ~ three:" tmpColl1 ~ three /* no entry with "THREE" ! */

:: REQUIRES class_rel /* load "Class_Rel.cmd" public definitions   */

/* intermixing RelBijectiveSet with RelDir via multiple inheritance,
   hence behaviour of both classes are present ! (Could be applied for
   RelTable or RelBijective too, or any other class descending from
   Object Rexx' Relation class.)                                    */
:: CLASS TEST SUBCLASS RelBijectiveSet INHERIT RelDir

:: ROUTINE dump
   USE ARG coll

   SAY "Dumping" coll":"
   tmpSet = .set ~ new
   DO index OVER coll
      IF tmpSet ~ HASINDEX( index ) THEN ITERATE
      tmpSet ~ PUT( index )
      DO item OVER coll ~ allat( index )
         SAY "index:" LEFT("["index"]", 7) "item: ["item"]"
      END
   END
   SAY LEFT( "", 40, "-" )
```

... yields the following output:

```
Dumping a TEST:
index: [TWO]   item: [there]
index: [three] item: [another entry with an index of 'two']
index: [1]     item: [one - another one]
----------------------------------------
tmpColl1 ~ 1    : one - another one
tmpColl1 ~ two  : there
tmpColl1 ~ three: The NIL object
```

3 MODULE `ROUTINE_USIFY.CMD`

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

3.1 Public Routine `USify()`

Usage: *`USify( string )`*

Returns a string containing US-letters (actually English letters), numbers and single underscores only. The string will never start or end with an underscore.

Source code of routine:

```
/* Initialization part, run only once, when first called:   */

tmp = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456790"
table_in  = XRANGE( )   /* get all chars                            */

/* set all non-alphanumeric chars to blank:                         */
table_out = TRANSLATE( table_in, tmp, tmp || table_in, " " )

.local ~ us.table_in  = table_in        /* save into local environment  */
.local ~ us.table_out = table_out       /* save into local environment  */

/* just allow US-letters and numbers, replace anything else
   with *ONE* underscore                                   */
:: ROUTINE USIfy                        PUBLIC
   USE ARG string

   RETURN SPACE( TRANSLATE( string, .us.table_out, .us.table_in ), 1, "_" )
```

Example:

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

a = "    über <den> Wölkchen, muß die Freiheit & ..."
SAY pp( a )
        /* show the effect of using TRANSLATE() only  */
SAY pp( TRANSLATE( a, .us.table_out, .us.table_in) )
        /* show the effect of USify()                 */
SAY pp( usify( a ) )

:: ROUTINE pp  /* private PP() version          */
   RETURN "[" || ARG( 1 ) || "]"
```

... yields the following output:

```
[    über <den> Wölkchen, muß die Freiheit & ...]
[     ber  den  W lkchen  mu  die Freiheit      ]
[ber_den_W_lkchen_mu_die_Freiheit]
```

4 MODULE `CLASS_REF.CMD`

This module requires *Class_Rel.cmd* and *Routine_USify.cmd* and supports the following public classes which all are subclasses of Object Rexx' builtin *Relation* class: `Anchor` and `Ref`. The purpose of this module is to define the basic functionality for allowing to define unique anchors and referring them at any time based on the anchor domain and the Object Rexx unique object ID. With other words, an anchor object may be used to have a unique surrogate string generated for and associated with any given object.

Due to the architecture it is possible to refer to an object for which no anchor was defined at that particular point in time. If such anchors are not referred to, the *Ref* class methods `setOfAnchorNames`, `setOfReferences` and `SayStatistics` may be used to debug them.

4.1 Public Class `Anchor`

The *Anchor* class allows for initializing anchor objects which pertain to specific anchor domains. All instances of class *Anchor* will get collected by the *Anchor* class and stored with the class variable *AnchorDir* which is accessible via the class method *`AnchorDir`*. An anchor object will create a unique string for any given object which follows the USify-rules as described above, this unique string is called a "surrogate string" in the context of this paper. The uniqueness of such surrogate strings pertains to the individual anchor objects.

Class methods of class *ANCHOR:*

`INIT`	Creates class-level attribute *AnchorDir,* a directory.
`AnchorDir`	Attribute method, allows access to the class-level attribute *"AnchorDir"* containing all instances of the *Anchor* class.

Instance methods of class *ANCHOR:*

`INIT( [ domain ] )`	
	Optional *domain* string of this anchor object. This *domain* string will be used as a prefix for generating unique ("surrogate") strings for objects. A given *domain* will get "US-ified" via the *USify()* routine. If not given, it defaults to the string `"ORX"`.
`getAnchorName( object )`	
	Assigns and returns a surrogate string for the given *object*. If the object was processed already, then the assigned surrogate string is retrieved from the RelTable object and returned. If *object is omitted then .nil* is returned.
`ObjCounter`	
	Returns the number of objects for which surrogate strings have been created and assigned to.
`AnchObjTable`	
	Returns the RelTable object which stores all objects as indices and their assigned surrogate strings as their values (items).

Example:

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

obj1 = .object ~ new /* an object               */
obj2 = "hi..."       /* another object          */
obj3 = 12            /* yet, another object     */

bk1Anch = .anchor ~ new( "Book 1" )      /* domain for book # 1 */
bk2Anch = .anchor ~ new( "Book 2" )      /* domain for book # 2 */

SAY "anchor for domain 'Book 1':"
CALL show obj1, "bk1Anch", bk1Anch
CALL show obj2, "bk1Anch", bk1Anch
CALL show obj3, "bk1Anch", bk1Anch
SAY "  ---> obj2 has alreay a string surrogate:"
CALL show obj2, "bk1Anch", bk1Anch
SAY LEFT( "", 40, "-" )

SAY "anchor for domain 'Book 2':"
CALL show obj1, "bk2Anch", bk2Anch
CALL show obj2, "bk2Anch", bk2Anch
CALL show obj3, "bk2Anch", bk2Anch
SAY "  ---> obj3's string surrogate in domain 'bk1Anch':"
CALL show obj3, "bk1Anch", bk1Anch

:: ROUTINE show      /* private PP() version          */
   SAY "  surrogate: ["ARG( 3 ) ~ getAnchorName( ARG( 1 ) )"]",
       "anchor domain: ["ARG( 2 )"] object: ["ARG( 1 )"]"
```

... yields the following output:

```
anchor for domain 'Book 1':
  surrogate: [Book_1_1] anchor domain: [bk1Anch] object: [an Object]
  surrogate: [Book_1_2] anchor domain: [bk1Anch] object: [hi...]
  surrogate: [Book_1_3] anchor domain: [bk1Anch] object: [12]
  ---> obj2 has alreay a string surrogate:
  surrogate: [Book_1_2] anchor domain: [bk1Anch] object: [hi...]
----------------------------------------
anchor for domain 'Book 2':
  surrogate: [Book_2_1] anchor domain: [bk2Anch] object: [an Object]
  surrogate: [Book_2_2] anchor domain: [bk2Anch] object: [hi...]
  surrogate: [Book_2_3] anchor domain: [bk2Anch] object: [12]
  ---> obj3's string surrogate in domain 'bk1Anch':
  surrogate: [Book_1_3] anchor domain: [bk1Anch] object: [12]
```

4.2 Public Class `Ref`

The class *Ref* builds on the *Anchor* class. It defines class methods only and uses the class ID string as the default prefix for the anchor domain.

Class methods of class *REF:*

`INIT`	Creates class-level attributes.
`AnchorObjectDir`	Private attribute method.
`setOfAnchorNames`	Returns a set containing those objects which are explicitly intended to be used as anchors.
`setOfReferences`	Returns a set containing those objects which are explicitly intended to refer to an anchor.
`getAnchorObject( object [, domain ] )`	Private attribute method, uses an *anchor* object to create the surrogate string. If the optional *domain* is omitted, then the class ID string of *object* is used instead.
`Reference( object [, domain ] )`	Returns the surrogate string by using an *anchor* object to create it; uses the private class method *getAnchorObject* to retrieve the anchor object for the given *domain*. If the optional *domain* is omitted, then the class ID string of *object* is used instead.
`createReference( object [, domain ] )`	Returns the surrogate string, uses the class method *Reference*. If the optional *domain* is omitted, then the class ID string of the *object* is used instead. In addition this method will save *object* in the class variable named *setOfAnchorNames*.
`getReference( object [, domain ] )`	Returns the surrogate string, uses the class method *Reference*. If the optional *domain* is omitted, then the class ID string of the *object* is used instead. In addition this method will save *object* in the class variable named *setOfReferences*.
`SayStatistics`	Writes statistical data to *.ERROR* total number of objects processed with method *createReference* and with method *getReference*. If there are objects which were referred to, but never created explicitly an appropriate message will be generated. If there are *objects* which were created for reference purposes but never referred to then an appropriate message will be generated.

Example:

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

obj1 = .object ~ new        /* an object                */
obj2 = "hi..."              /* another object           */
obj3 = 12                   /* yet, another object      */
obj4 = .set ~ new           /* yet, another object      */
obj5 = .set ~ new           /* yet, another object      */

.ref ~       reference( obj1, "Book 1" ) /* create and get surrogate      */
.ref ~ createReference( obj2, "Book 1" ) /* create reference explicitly   */
.ref ~ createReference( obj3, "Book 1" ) /* never referred to explicitly  */
.ref ~ createReference( obj4, "Book 1" ) /* never referred to explicitly  */

SAY .ref ~ getReference( obj2, "Book 1" )      /* refer to explicitly    */
SAY .ref ~ getReference( obj5, "Book 1" )      /* not explicitly created */

.ref ~ SayStatistics        /* display statistics                        */
```

... yields the following output:

```
Book_1_2
Book_1_5
==> class [REF] references created explicitly: [3] references used: [2]
==>             [2] object-references were created, but never referred to!
==>             [1] object-references were created implicitly.
```

5 MODULE `SGMLENTITY_UTIL.CMD`

This module requires *NLS_Util.cmd* and supports one public class `"sgmlEntity"` and the following two public routines: `CPString2SGMLEntity` and `CPFile2SGMLEntity`. It supplies the basic functionality to translate codepage dependent characters into codepage independent SGML entities[7] and vice versa. All defined translation tables are stored with the class *sgmlEntity* which itself is nothing else but a directory which holds all translation tables indexed with their appropriate codepage table number.

sgmlEntity_Util.cmd contains the definitions for the author's codepages 850 and 437 which get set up once at initialization time. It is easy to extend the support to additional codepages by creating the appropriate translation tables and inserting them into the class *sgmlEntity*.[8]

[7] An SGML entity is enclosed between an ampersand "&" and a semi-colon ";" and either has a symbolic name or a standardized number. E.g. the "hypertext markup language" (HTML) employs SGML entities.

[8] If you devise additional codepage support for this module, then please let the author know, so that it can be added to the official distribution.

5.1 Public Class `sgmlEntity()`

This class is defined such that it becomes easy to address it via the local environment. It camouflages the fact that it is a bare directory holding the translation tables. It allows for sending class methods only (methods directed at the class object *".sgmlEntity"*).

Source code of entire class definition:

```
:: CLASS SGMLEntity           PUBLIC

:: METHOD init CLASS
   EXPOSE cpDirectory

   cpDirectory = .directory ~ new       /* directory to contain codepages */

:: METHOD unknown       CLASS           /* forward messages to .directory */
   EXPOSE cpDirectory
   USE ARG mess_name, mess_args

   FORWARD MESSAGE (mess_name) ARGUMENTS (mess_args) TO ( cpDirectory )
```

A defined "UNKNOWN" method will get invoked by the Object Rexx interpreter if a message was received for which no method has been defined. In this case the unknown message together with its arguments is **forwarded** to the class variable named *cpDirectory* which in turn is an instance of the Directory class.

5.2 Public Routine `CPString2SGMLEntity()`

Usage: `CPString2SGMLEntity( string [, [ codepage ] [, "Reverse"] ] )`

Returns the *string* translated to the appropriate SGML entitities according to the given *codepage*. If the optional third argument *Reverse* is given the translation is reversed from SGML entities to the characters according to *codepage*. If *codepage* is omitted, then the codepage of the actual default NLS-object of *NLS_Util.cmd* is used instead.

Source code of routine:

```
:: ROUTINE CPString2SGMLEntity  PUBLIC
   USE ARG string, CodePage, Reverse

   IF \ VAR( "CodePage" ) THEN CodePage = get_nls_default_object() ~ codepage
   IF \ VAR( "Reverse" )  THEN Reverse  = ""
        /* does given CP exist, if not use default CP, try
           to retrieve CP-2-SGMLentity table  */
   cpTable = .SGMLEntity ~ entry( codepage )

   IF cpTable = .nil THEN /* SGMLentity-table for codepage not found, error */
   DO
      SIGNAL ON SYNTAX
      RAISE SYNTAX 40.904 ARRAY ( "CPString2SGMLEntity()", '2 ("CodePage")',,
                                  "supported SGMLEntity codepages", codepage )
   END
                           /* call proc according to third argument        */
   IF ( TRANSLATE( LEFT( Reverse, 1 ) ) = "R" ) THEN
      RETURN sgml2chars( string )       /* reverse      */
   ELSE
      RETURN chars2sgml( string )
/* --------------- procedure ----------------- */
CHARS2SGML :
   tmpSupp = cpTable ~ supplier   /* get a supplier from table     */
   DO WHILE tmpSupp ~ available
                  /* translate chars to HTML/SGML entities      */
      string = CHANGESTR( tmpSupp ~ index, string, tmpSupp ~ item )
      tmpSupp ~ next
   END
   RETURN string


/* --------------- procedure ----------------- */
SGML2CHARS :
   tmpSupp = cpTable ~ supplier   /* get a supplier from table     */
   DO WHILE tmpSupp ~ available
                  /* translate chars to HTML/SGML entities      */
      string = CHANGESTR( tmpSupp ~ item, string, tmpSupp ~ index )
      tmpSupp ~ next
   END
   RETURN string

SYNTAX : RAISE PROPAGATE        /* raise error in caller           */
```

Example:

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

SAY CPString2SGMLEntity( "über den Wölkchen, muß ...", 850 )

a = "German umlauts: &auml;&ouml;&uuml; &Auml;&Ouml;&Uuml; &szlig;"
SAY CPString2SGMLEntity( a, 850, "R")
```

... yields the following output:

```
&uuml;ber den W&ouml;lkchen, mu&szlig; ...
German umlauts: äöü ÄÖÜ ß
```

5.3 Public Routine `CPFile2SGMLEntity()`

Usage: `CPFile2SGMLEntity( file [, [ codepage ] [, "Reverse"] ] )`

Translates the content of the given *file* to the appropriate SGML entitities according to the given *codepage*. If the optional third argument *Reverse* is given the translation is reversed from SGML entities to the characters according to *codepage*. If *codepage* is omitted, then the codepage of the actual default NLS-object of *NLS_Util.cmd* is used instead.

Example:

```
CALL sgmlEntity_util /* load "routinesgmlEntity_util.cmd" public definitions */

  /* make sure HTML-file "home.html" contains SGML-entities only */
CALL CPFile2SGMLEntity( "home.html" )
```

6 MODULE `HTML_UTIL.CMD`

This module requires `sgmlEntity_util.cmd`, `RGF_Util.cmd` and `Class_Ref.cmd` and supports the following public classes: `HTML_DOC`, `HTML_Table`, `HTML_List` and `HTML.Reference`. In addition the following public routines are defined: `A_HREF()`, `A_NAME()`, `BreakLines()`, `CAPITALIZE()`, `InitCap()`, `PlainText()`, `SmartCap()`, `WWW_TAG()`, `X2BARE()`, `X2BLANK()` and `htmlComment()`.

Although the module is not finalized as of yet, it may proof useful to some[9].

6.1 Public Classes

The `HTML_DOC`, `HTML_List` and `HTML_Table` classes are designed such that they behave like a storage container which has to be explicitly closed. The `HTML_List` class allows for rendering the list in different styles. The `HTML.Reference` class adapts the *Ref* class from *Class_Rel.cmd* for the purpose of generating valid HTML-code (`"<a "`-tags).

[9] The reader is asked to look up the source code of `ORX_ANALYZE_ASCII.CMD` and compare it with the source code of `ORX_ANALYZE_HTML.CMD`, which uses *`HTML_Util.cmd`* extensively. In order to understand the source code of the first two modules the reader is referred to [Flat96c].

6.1.1 Class `HTML_DOC`

This class serves as the container to receive and edit strings for producing valid HTML-files.

Instance methods of class *HTML_DOC:*

`INIT( file [, [title] [, [bReplace] [, [addHeader] [, addFooter] ] ] ] )`	Sets up an instance and associates it with the given *file*. If the optional title is not given the string "No title supplied !" will be used instead. The third optional argument *bReplace* defaults to *.false*. If *bReplace* is set to *.false* a syntax error will be raised, if the given *file* exists already. The optional *addHeader* if given must consist of valid HTML-code and will get inserted right before the autogenerated *"</HEAD>"* tag. The optional *addFooter* if given must consist of valid HTML-code and will get inserted right before the autogenerated *"</BODY>"* tag.
`UNKNOWN`	This method intercepts unknown messages to the HTML_DOC instance. If the message name is *LINEOUT* then the argument (a string assumed) is written directly to the HTML-file without altering its contents. Therefore the string must contain valid HTML-code. Any other message is interpreted as a valid HTML-tag with the following assumptions: 1) message name is the first tag 2) first argument is the string which gets embedded within the tag 3) second argument contains the attributes for the first tag, if any 4) arg(3...n) each contain a tag *together* with their attributes, if any
`HTML_Header`	Private method for creating the HTML-header, invoked by the *INIT*-method.
`HTML_Footer_Close`	Private method for creating the HTML-footer, invoked by the *CLOSE*-method
`Close`	Method to end working with this *HTML_DOC* instance, invokes method *HTML_Footer_Close*.
`UNINIT`	Method which makes sure that the *CLOSE*-method of this *HTML_DOC* instance is called in order to make sure that the mandatory footer information is written.

6.1.2 Class `HTML_List`

This class serves as the container to receive list-items and list-descriptions for producing valid ordered, unordered and definition HTML-lists.

Class methods of class *HTML_LIST:*

`INIT`	Defines the class variable named *counter* and initializes it.
`Counter`	Returns the value of *counter* increased by 1.

Instance methods of class *HTML_LIST:*

`INIT( [ ListTagOpen [, [ListTagItem] [, ListTagDescription ]]]  )`	
	Sets up an instance and invokes method *SetListType* to determine default list type.
`SetListType( [ ListTagOpen [, [ListTagItem] [, ListTagDescription ]]]  )`	
	This method determines the default list type. If optional *ListTagOpen* is not given, then the default list type is an unordered list. If the optional *ListTagItem* is not given, then a definition list is assumed, otherwise the tag *LI* is used for *ListTagItem* and *ListTagDescription*. This method may be invoked at any time, effectively allowing to change the type of the HTML-list on the fly.
`Item( string [, attributes ] )`	
	Adds *string* as a list item, respects optional *attributes* verbatim. Functional only as long as the *CLOSE* message has not been sent to the *HTML_LIST* instance.
`Term( string [, attributes ] )`	
	Adds *string* as a definition term item, respects optional *attributes* verbatim. Functional only as long as the *CLOSE* message has not been sent to the *HTML_LIST* instance.
`Description( string [, attributes ] )`	
	Adds *string* as a description for a definition list, respects optional *attributes* verbatim. Functional only as long as the *CLOSE* message has not been sent to the *HTML_LIST* instance.
`Close`	Method to end working with this *HTML_LIST* instance.
`htmlText`	Returns the instance rendered to a string containing the HTML-code for the entire list.

6.1.4 Class `HTML_Table`

This class serves as the container to receive table definition items for producing valid HTML-tables.

Class methods of class *HTML_TABLE:*

`INIT`	Defines the class variable named *counter* and initializes it.
`Counter`	Returns the value of *counter* increased by 1.

Instance methods of class *HTML_TABLE:*

`INIT( [ tableAttributes ]  )`	
	Sets up an instance of class *HTML_TABLE*. If the optional argument *tableAttributes* is omitted than the default attributes `"BORDER CELLPADDING=5"` are filled in.
`SetCaption( Caption [, Attributes] )`	
	Sets up a *Caption*. If the optional argument *Attributes* is omitted than the default attribute `"ALIGN=BOTTOM"` is filled in.
`putColumn( string [, attributes ] )`	
	Adds *string* as a column item, respects optional *attributes* verbatim. If string is omitted or empty an HTML-comment is filled in instead. Functional only as long as the *CLOSE* message has not been sent to the *HTML_LIST* instance.
`putHeader( string [, attributes ] )`	
	Adds *string* as a header column item, respects optional *attributes* verbatim. If string is omitted or empty an HTML-comment is filled in instead. Functional only as long as the *CLOSE* message has not been sent to the *HTML_LIST* instance.
`newRow( [ bFixup ] )`	
	Ends the current row definition. If a value for *bFixup* and the instance variable *bSkipActive* is *.true* by previously sending the message *SkipCOLUMN,* missing columns up to the end of the row will be closed with a `"COLSPAN="` attribute.
`skipColumn( [nrOfColumns] )`	
	Skips *nrOfColumns* columns with a `"COLSPAN="` attribute. *bSkipActive* will be set to *.true* (cf. method *newRow)*. The optional *nrOfColumns* defaults to 1.
`emptyColumn( [nrOfColumns] )`	
	Inserts *nrOfColumns* empty columns. The optional *nrOfColumns* defaults to 1.
`Close`	Method to end working with this *HTML_TABLE* instance.
`htmlText`	Returns the instance rendered to a string containing the HTML-code for the entire table.

6.1.5 Class `HTML.Reference`

This class serves as the reference mechanism for HTML "<A name=" and "<A href=" tags. It allows to produce valid HTML-anchor names for any Object Rexx object. As it subclasses the class *REF* from *CLASS_REF.CMD* it is also possible to use *href*-style references before an anchor is defined, i.e. employing forward referencing. As the entire code for the necessary specializations is remarkeably short, the entire class definition is shown.

Source code of entire class definition:

```
:: CLASS html.reference SUBCLASS ref  PUBLIC    /* subclass the anchor-manager */

:: METHOD A_Name CLASS      /* retrieve a reference name, else create it */
   USE ARG anObject, text

   RETURN '<A NAME="' || self ~ createReference( anObject ) || '">' text "</A>"

:: METHOD A_Href CLASS      /* retrieve a reference name, else create it */
   USE ARG anObject,  text, htmlFile

   IF ARG( 3, "O" ) THEN htmlFile = ""   /* supply default empty string  */

   RETURN '<A HREF="' || htmlFile || "#" ||, self ~ getReference( anObject ) ||,
                 '">' || text || "</A>"
```

6.2 Public Routine `A_NAME()`

Usage: `a_name( name, text )`

Returns a HTML-compliant anchor-name definition, having *name* as its anchor name underlying the *text*.

Source code of the public routine:

```
:: ROUTINE A_NAME                PUBLIC
   PARSE ARG name, text

   RETURN '<A NAME="' || name || '">' text "</A>
```

6.3 Public Routine **`A_HREF()`**

Usage: `a_href( target_document, name, text )`

Returns a HTML-compliant anchor-name definition, having *name* as its target anchor in the *target_document* underlying the *text*.

Source code of the public routine:

```
:: ROUTINE A_HREF              PUBLIC
   PARSE ARG doc, name, text

   RETURN '<A HREF="' || doc || "#" || name || '">' || text || "</A>"
```

6.4 Public Routine **`BreakLines()`**

Usage: `breakLines( CRLFstring [ , nrOfChars] )`

Returns the *CRLFstring* edited such that between carriage-return and line-feed characters there are no more than *nrOfChars* characters. The optional argument *norOfChars* defaults to 100 characters per line.

6.5 Public Routine **`Capitalize()`**

Usage: `capitalize( string, FirstMarkUp, RestMarkUp  )`

Returns the edited string in uppercase. Each capital letter in *string* is enclosed within the *FirstMarkUp* tag all other characters within the *RestMarkUp* tag. If both, *FirstMarkUp* and *RestMarkUp* contain an empty string, then the markup for *RestMarkUp* will be set to `"FONT SIZE=-1"`.

6.6 Public Routine `InitCap()`

Usage: `InitCap(` *`string, FirstMarkUp, RestMarkUp`* `)`

Returns the edited string in uppercase. The first character of each word will be enclosed within the *FirstMarkUp* tag, all other characters within the *RestMarkUp* tag. If both, *FirstMarkUp* and *RestMarkUp* contain an empty string, then the markup for *RestMarkUp* will be set to `"FONT SIZE=-1"`.

6.7 Public Routine `SmartCap()`

Usage: `SmartCap(` *`string, type, bBlanks`* `)`

Returns the *string* either initcapped or capitalized depending on the value of *type* ("I" or "C"). If the optional *type* is omitted, the string will be capitalized. By default all blanks are translated into the SMGL entity *"&nbsp;"*, the character for non-breaking space. If *bBlanks* is set to false, blank characters will remain regular blanks in the returned string.

Source code of the public routine:

```
/* ------------------------------------------------------------------------------ */
/* purpose: capitalize or initcap plain text, replace blanks by &nbsp;
(non-breaking space) by default */
/*          works with defaults: lowercase letters are FONT SIZE=-1
      */
/* capitalize, translate to html of **PLAIN TEXT** string, blanks will be
translated into &nbsp; ! */
:: ROUTINE SmartCap                     PUBLIC
  USE ARG string, type, bBlanks

  bBlanks = ( bBlanks <> .false ) /* default: translate blanks into &nbsp;     */

  IF TRANSLATE( LEFT( type, 1 ) ) = "I" THEN type = "I" /* InitCap desired     */
                                      ELSE type = "C" /* default: capitalize */

  IF bBlanks THEN string = CHANGESTR( " ", string, "00"x )

  IF type = "C" THEN string = CAPITALIZE( string )    /* capitalize string      */
  ELSE string = INITCAP( string )                     /* initcap string        */

  IF bBlanks THEN string = CHANGESTR( "00"x, string, "&nbsp;" ) /* to "&nbsp;"  */
  ELSE string = CHANGESTR( "00"x, string, " " )               /* leave blanks */

  RETURN string
```

6.8 Public Routine `X2Blank()`

Usage: `X2Blank( string )`

Returns the edited string in which all blanks are replace with the SGML entity for the "non breaking space" ("&nbsp;") character.

Source code of the public routine:

```
:: ROUTINE X2BLANK             PUBLIC
   PARSE ARG plainText

   RETURN CHANGESTR( " " , plainText,  "&nbsp;" )
```

6.9 Public Routine `X2Bare()`

Usage: `X2Bare( string )`

Returns the edited string in which all special SGML/HTML-characters, namely ampersand (&), smaller than (<), greater than (>) and the quote (") are replaced with their respective SGML-entities.

Source code of the public routine:

```
:: ROUTINE X2BARE              PUBLIC
   PARSE ARG plainText

   plainText = CHANGESTR( "&" , plainText,  "&amp;" ) /* translate ampersand */
   plainText = CHANGESTR( "<" , plainText,  "&lt;" )  /* translate smaller   */
   plainText = CHANGESTR( ">" , plainText,  "&gt;" )  /* translate larger    */
   plainText = CHANGESTR( '"' , plainText,  "&quot;" )/* translate to quote  */
   RETURN plainText
```

6.10 Public Routine `PlainText()`

Usage: `PlainText( string )`

Returns the edited string by calling *X2Bare()* and *BreakLines()* with *string* as an argument.

6.11 Public Routine `htmlComment()`

Usage: `htmlComment` *`( string )`*

Returns the string in form of an HTML/SGML-comment.

Source code of the public routine:

```
:: ROUTINE htmlComment
   RETURN "<!--" ARG( 1 ) "-->"
```

6.12 Public Routine `www_tag()`

Usage: `www_tag` *`( string [, tag1+attr1] [, tag2+attr2]... )`*

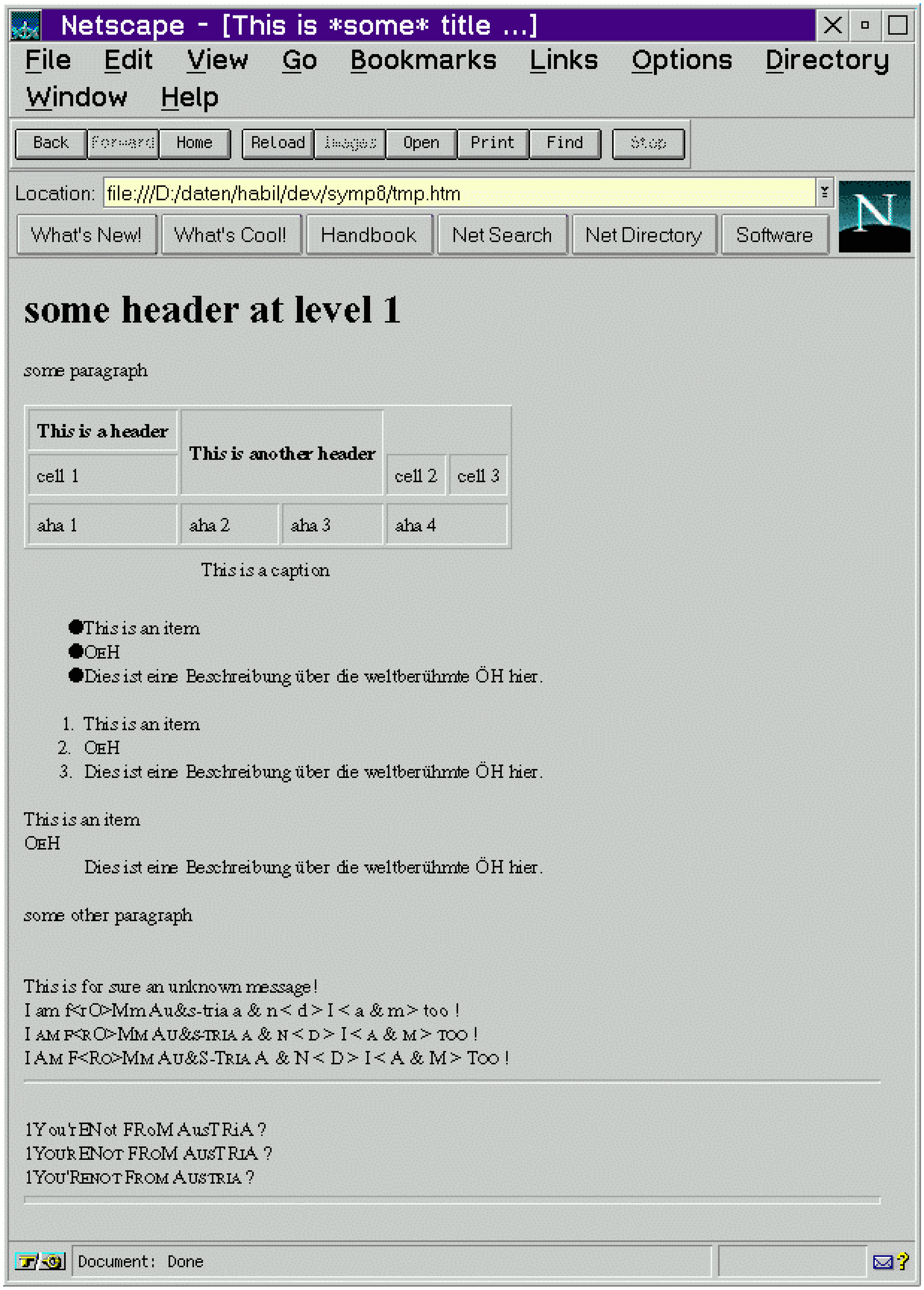
Returns the HTML-code for the *string* embedded in *tag1* with optional attributes *attr1*, *tag2* with optional attributes *attr2* and so forth. The routine makes sure that for the following tags no end-tags are produced as this is forbidden per the definitions of *HTML 3.2:* `"BR", "HR", "Image", "Input"` and `"IsIndex".`

6.13 Example

The following example shows code to produce an HTML-file which may be viewed with any WWW-browser:

```
/* */

list1 = .html_list ~ new                /* create a HTML_LIST instance  */
list1 ~ item("This is an item")
/* a term and a description supplied */
list1 ~ term( Capitalize( "OeH") )
list1 ~ description( "Dies ist eine Beschreibung über die weltberühmte ÖH
hier.")
list1 ~ close

aha1 = .html_table ~ new                /* create a HTML_TABLE instance */
aha1 ~ setcaption(" This is a caption " )
aha1 ~ putheader( "This is a header" )
aha1 ~ putheader( "This is another header", "ROWSPAN=2 COLSPAN=2" )
aha1 ~ newrow
aha1 ~ putColumn( "cell 1" )
aha1 ~ putColumn( "cell 2" )
aha1 ~ putColumn( "cell 3" )
aha1 ~ newrow
aha1 ~ newrow
aha1 ~ putColumn( "aha 1" )
aha1 ~ putColumn( "aha 2" )
aha1 ~ putColumn( "aha 3" )
aha1 ~ putColumn( "aha 4", "COLSPAN=2" )
aha1 ~ newrow

a = "I am f<rO>Mm Au&s-tria a & n < d > I < a & m > too ! "
a1 = capitalize( a )            /* show effects of Capitalize   */
a2 = InitCap( a )               /* show effects of InitCap      */

b = "1You'rENot FRoM AusTRiA ?"
b1 = capitalize( b )            /* show effects of Capitalize   */
b2 = InitCap( b )               /* show effects of InitCap      */

        /* create an instance of HTML_DOC       */
file = "tmp.htm"
html = .html_doc ~ new( file, "This is *some* title ...", .true )
html ~ h1( "some header at level 1" )
html ~ p( "some paragraph" )
html ~ lineout( aha1  ~  htmlText )
html ~ lineout( list1 ~  htmlText )
html ~ lineout( list1 ~~ setListType( "OL" ) ~ htmlText )
html ~ lineout( list1 ~~ setListType( "DL" ) ~ htmlText )
html ~ lineout( www_tag( "some other paragraph", "P" ) )
html ~ br( "This is for sure an unknown message!")
html ~ br( PlainText( a ) )
html ~ br( a1 )
html ~ br( a2 )
html ~ hr
html ~ br( PlainText( b ) )
html ~ br( b1 )
html ~ br( b2 )
html ~ hr( , "SIZE=5")
html ~ close
CALL CPFile2SGMLEntity file, 850   /* translate 850-chars to SGML-entities ! */

:: REQUIRES HTML_UTIL.CMD          /* load HTML-support    */
```

The resulting HTML-file is displayed on the next page.



some header at level 1

some paragraph

This is a header	**This is another header**			
cell 1			cell 2	cell 3
aha 1	aha 2	aha 3	aha 4	

This is a caption

- This is an item
- OEH
- Dies ist eine Beschreibung über die weltberühmte ÖH hier.

1. This is an item
2. OEH
3. Dies ist eine Beschreibung über die weltberühmte ÖH hier.

This is an item
OEH
Dies ist eine Beschreibung über die weltberühmte ÖH hier.

some other paragraph

This is for sure an unknown message!
I am f<rO>Mm Au&s-tria a & n< d > I < a & m> too !
I AM F<RO>MM AU&S-TRIA A & N < D > I< A & M > TOO !
I AM F<RO>MM AU&S-TRIA A & N < D > I< A & M > TOO !

1You'rENot FRoM AusTRiA ?
1YOU'RENOT FROM AUSTRIA ?
1YOU'RENOT FROM AUSTRIA ?

---

7 SUMMARY

This paper discussed and documented the public routines and public classes of the following modules in a brief manner: `class_rel.cmd`, `routine_USIfy.cmd`, `class_ref.cmd`, `sgmlEntity_util.cmd` and `html_util.cmd`.

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.

8 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", 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.