Dimac
websolutions
www.dimac.net

# w3 Utils 1.0

Manual

# Table of contents

**INTRODUCTION ..... 4**

Prerequisites ..... 4
System Requirements ..... 4
Free Support! ..... 4

**INSTALLATION ..... 5**

Running the installation program ..... 5
Uninstalling w3 Utils ..... 5

**GETTING STARTED ..... 6**

Installation ..... 6
Object usage: Review ..... 6
Object usage in Visual Basic ..... 7
Dates and ASP Javascript ..... 7
StringUtils ..... 8
CountryUtils ..... 9
GeneratorUtils ..... 9
ValidationUtils ..... 10
DateUtils ..... 10
NetUtils ..... 10

**REFERENCE ..... 11**

Object references used in the examples ..... 11
Complete Reference ..... 11
w3.stringUtils ..... 11
w3.CountryUtils ..... 22
w3.GeneratorUtils ..... 24
w3.ValidationUtils ..... 24
w3.DateUtils ..... 27
w3.NetUtils ..... 29

**SOFTWARE LICENSE AGREEMENT AND LIMITED WARRANTY ..... 31**

# Introduction

Welcome to Dimac w3 Utils - the solution to several of your programming chores are within your reach! This manual will help you get the most out of this product and allow it to boost your productivity.

The w3 Utils package is a bundle of software tools in the form of COM-objects each targeting one area where programmers often are faced with solving the same problems. The w3 Utils package has been developed with the needs of the network developer in mind. These are the components of the complete package and descriptions of the routines they include:

| | |
|---|---|
| w3.StringUtils | String manipulation: Conventional and net oriented |
| w3.CountryUtils | Lookup routines for countrycodes and countrynumbers |
| w3.GeneratorUtils | Specific string generation |
| w3.ValidationUtils | Various validation routines |
| w3.DateUtils | Date manipulation and conversion |
| w3.NetUtils | Network utilities |

## Prerequisites

This manual assumes programming proficiency and knowledge of using COM-objects within the language and development environment of your choice. Examples of some development environments supporting COM and their respective languages are:

- Microsoft Visual Studio for C++, Java and Visual Basic
- Microsoft ActiveX Scripting (ASP) using any environment/editor of your choice and any supported script-language of your choice (currently supported: JavaScript and VBScript)
- Inprise Borland Delphi for Object Pascal
- Inprise Borland C++ Builder for C++
- Inprise Borland JBuilder for Java

## System Requirements

w3 Utils is a package of COM objects and requires a computer supporting these. Specifically, the following points are required for the intended system:

(1) A computer running Microsoft Windows 95, 98 or Windows NT version 3.51 or 4.0. 32 MB memory is recommended for running. Winsock 2 is also required for w3 Utils to be fully functional. Winsock 2 is preinstalled on Windows 98 and Windows NT 4.0 installations. Win95 and WinNT 3.5x userss should download Winsock 2 from ftp.microsoft.com.
(2) The complete package requires 2 MB of harddisk space.

## Free Support!

Dimac gives free support on our products as long as it is conducted solely by email and you are a registered customer. In order to register you need to log in at our tech-support site. In addition, support is also given through this tech-support site at http://tech.dimac.net/. Dimac does not give any phone support and no such program has been planned. If you do need further assistance in your project please contact us by email.

# Installation

Installation of w3 Utils is straightforward as it is the same as for any COM-component.

## *Running the installation program*

If you obtained w3 Utils in CD form simply pop the CD in the computer where it is to be installed and run setup.

The same file is run if the package has been downloaded.

## *Uninstalling w3 Utils*

Currently there is no automatic uninstall utility for the w3 UpLoad component. The easy steps to uninstall are as follows:

Find the installed w3u_base.dll and then type “REGSVR32 \DIMAC\ w3u_base.dll /u” from your start menu (run).

After unregistering the component in step one you are now free to remove the files from your computer.

# Getting Started

The purpose of this chapter is to give a step by step guide to the fundamentals of using w3 Utils and it is partly written as a tutorial. It is aimed at helping the developer who, although familiar with programming and the environment of choice, is just beginning to use COM-objects (or perhaps never knowingly used such software previously in their development). Any which way – if you feel unsure about using the package, reading this might be time well spent. However, since COM objects can be used in virtually any language you may have to substitute the syntax in the examples with the specific syntax of your language.

For the purposes of this chapter we have chosen the scripting language Javascript as the syntax resembles Java, C++ and is well known for it's web usage in client script as well as in MS ASP code. If you use pascal or another language the constructs are still easily understood.

We have provided counterparts to most code where differences are essential in Visual Basic. Although there are differences in the different Visual Basic dialects (VB, VBA, VBScript) most of these constructs work the same way. The examples are tested as VBScript code.

## Installation

Please see the previous chapter.

## Object usage: Review

Even though it is assumed that you know how to use objects here is a short review of the major points. Should you feel unsure of any of this material, we highly recommend that you invest some time in reading about object oriented programming theory and the specifics of applying it to your chosen developing environment. Several of the most popular environments have such documentation in their online help system or even on their website.

In order to use an object you create an instance of it in the code. For example:

```
var StringUtils = Server.CreateObject( 'w3.StringUtils' );
```

After the object (or rather an instance of it) has been created in memory (on the webserver when it comes to this ASP application) you have gained access to all of the methods and functions it is capable of. Actually, the result of the statement is an instance of the object is created and the variable 'StringUtils' (the object pointer) is set to point to it.

In order to use the object you then send messages to it telling it what to do and the data which it needs to do it or upon which it will act. A message is basically a function or procedure which you write as you normally would but after the object pointer with a dot ('.') in between. For example to use the function 'HTMLEncode' of the previously created object you could write:

```
var htmlText = StringUtils.HTMLEncode( aText );
```

It returns a string with the proper HTML codes for special characters, such as 'åäö' and so on.

In summary, if we desired to encode a text into proper HTML we could use the following code:

```
var StringUtils, aText, htmlText;
StringUtils = Server.CreateObject( 'w3.StringUtils' );
aText = 'Hello and welcome to Skåne!';
htmlText = StringUtils.HTMLEncode( aText );
```

After running this code the variable 'htmlText' would contain 'Hello and welcome to Skåne!'.

## Object usage in Visual Basic

Here follows VB counterparts for the Javascript snippets in the previous section.

Object creation is achieved by the following:

```
SET StringUtils = Server.CreateObject( "w3.StringUtils" )
```

Usage of object – message sending is about the same as in Javascript:

```
htmlText = StringUtils.HTMLEncode ( aText )
```

In total, this piece of code could look like:

```
Dim StringUtils, aText, htmlText

SET StringUtils = Server.CreateObject( "w3.StringUtils" )
aText = "Hello and welcome to Skåne!"
htmlText = StringUtils.HTMLEncode ( aText )
```

It achieves the same as the Javascript counterpart. This code looks the same even if used in VBA for MS Office, VB development or VBScript for ASP.

## Dates and ASP Javascript

The way dates are stored and handled by Javascript is not compatible with how most other programs (in our case we are interested in interacting with COM and ActiveX objects) store and use dates. You are forced to use the getVarDate() method when sending a date to a COM object. For example, using the DateUtils IncDay method:

```
var DateUtils = Server.Createobject ( 'w3.DateUtils' );

var aDate = new Date();
var tomorrow = new Date ( DateUtils.IncDay( aDate.getVarDate(), 1 ) );
```

The getVarDate() method returns the VT_DATE value stored in the Date object which can be used by COM objects for example.

# Package Overview

This chapter summarizes the routines of all the tools included in the package and serves both as a software overview and a useful shorthand when you are familiar with the functions.

## StringUtils

| *Name* | *ResultType* | *Arguments* | *Description* |
|---|---|---|---|
| AdjustLineBreaks | String | szString | Adjusts all linbreaks so that all breaks are valid CRLF sequenses |
| BuffToHex | String | szString | Converts a datastring to a Hex sequence. Can be used for dumping memory, converting blobs into SQL usable strings. |
| ContentType | String | FileName | Returns the content-type given a filename |
| ConvertBase | String | dwNum, bBase | Converts the base of a numeric to a user defined base. |
| Currency2String | String | Cur | Formats a currency variable to string using the Locale of the computer. |
| CutText | String | szString, szFrom, szTo | Cuts text from a string to a string literal. |
| DemaskString | String | szString, Mask | Mask a string and removes all characters defined in MASK |
| ExtractQuotedStr | String | szString, [QuoteChar] | Unquotes a string that have been QuotedStr:ed |
| FormatCurrency | String | Cur, Pattern | |
| HashString | Integer | szString | Calculates a Hash value of a string |
| HasWildcards | Integer | szString | Returns the index of the first wildcard in a string. If none is present then zero is returned |
| Hex | String | dwNumber, [NumChars] | Converts a numeric variable to Hexadecimal. |
| HexDump | String | szString | Produces a Hexdump of a datastring |
| HTMLEncode | String | szString, [ProcessHTTPLinks] | HTMLEncodes text. This function takes care of internatinal characters with their respective . Also, optinally, links all http:// addresses found in the text. |
| HTMLFormatBreaks | String | szString | Reformats a string and inserts a <BR> before each CRLF sequence. |
| Indent | String | szString, Columns | Indents a string. |
| JscriptEncode | String | szString | JScriptEncode encodes a string so that it can be used within a JScript statement. It converts the cr, lf, tab characters to \r, \n and \t etc.. |
| Ltrim | String | szString | Trims all breakable characters of the left on a string |
| MakeISO8859_1 | String | szString | Encodes a string, if neccesary, in ISO8859-1 format. Can be used in SMTP Mail headers etc... |
| MaskString | String | szString, Mask | Masks a string to only contain cracters specified in MASK |
| Match | Boolean | szString, Pattern | Macthes two strings. Pattern can contain wildcards. |
| MimeDecode | String | szString | Decodes a string encoded with MimeEncode or any other base64 encoded string. |
| MimeEncode | String | szString | Base64 encodes a string of data |
| QPDecode | String | szString | Decodes a Quoted-Printable encoded string. |
| QPEncode | String | szString | Applies Quoted-Printable encoding to a string |
| QuotedStr | String | szString, [QuoteChar] | Creates a Quoted string ready for use with SQL statements etc. |
| Replace | String | szString, SearchFor, ReplaceWith | Replaces all substrings specified with another. |
| ReverseString | String | szString | Reverses a string |
| Rtrim | String | szString | Trims all breakable characters of the right on a string |
| Soundex | String | szString | Calculates the soundex value of a string |
| String2Currency | Currency | szString | Converts a Currency string to a Currency variable. |
| Trim | String | szString | Trims all breakable characters of the beginning and end of a string |
| TrimQuotes | String | szString | Trims quotes of an String |
| WrapText | String | szString, ColWidth, [JustifyMargins], [Columns], [ColumnDelimiter] | Wraps text into columns with options to justify margins, multicolumar text etc. |

## CountryUtils

| Name | ResultType | Arguments | Description |
|---|---|---|---|
| LookupCountryCodeByName | String | CountryCode | |
| LookupCountryCodeByNumber | String | CountryNumber | |
| LookupCountryNameByCode | String | CountryCode | |
| LookupCountryNameByNumber | String | CountryNumber | |
| LookupCountryNumberByCode | Integer | CountryCode | |
| LookupCountryNumberByName | Integer | CountryName | |

## GeneratorUtils

| Member | ResultType | Arguments | Description |
|---|---|---|---|
| GenerateGUID | String | [Stripped] | Returns a GUID, Optinally stipped from all special characters. |
| GeneratePassword | String | dwLength | Given a length, generates a close to unique string that can be used as password etc. |
| GenerateSemiUniqueString | String | dwLength, [stringClass] | Given a length, generates a close to unique string that can be used as password etc. |

## ValidationUtils

| Member | ResultType | Arguments | Description |
|---|---|---|---|
| CheckEmailAddress | Boolean | EmailAddress, [DeepScan] | Checks to see if an email address is valid. Optinally checks if the user is valid in the specified domain. |
| CheckSumMod10 | Byte | Number | |
| GetCreditcardType | String | CCNumber | Given a string, returns the type of creditcard. Supports: VISA, Mastercard, AmiEx. |
| SE_CheckBankgironr | Boolean | szBGNR | |
| SE_CheckOrgnr | Boolean | szOrgNr | |
| SE_CheckPostgironr | Boolean | szPGNr | |
| SE_CheckPsnr | Boolean | szPSNR | Checks the personal<BR>Locale specific: Sweden |

## DateUtils

| Member | ResultType | Arguments | Description |
|---|---|---|---|
| DateTimeToGMT | String | dDate | Converts a dateTime to a GMT string |
| FormatDateTime | String | Mask, dDate | Formats a dateTime using a user defined mask. |
| GetGMTOffset | Integer | | Returns the offset from GMT |
| GetTimeZone | String | | Returns the name of the current timezone |
| GMTToDateTime | DateTime | szGMTDateString | Converts a GMT string to DateTime |
| IncDay | DateTime | dtDate, dwCount | Increeses a date with <I>count</I> days |
| IncMonth | DateTime | dtDate, dwCount | Increeses a date with <I>count</I> months |
| IncWeek | DateTime | dtDate, dwCount | Increeses a date with <I>count</I> weeks |
| IncYear | DateTime | dtDate, dwCount | Increeses a date with <I>count</I> years |
| WeekByDate | Byte | dtDate | Returns the week number given a date. |

## NetUtils

| Member | ResultType | Arguments | Description |
|---|---|---|---|
| DebugWrite | | szString | Writes a string to NetDebug (tm). NetDebug is a free software avaiable from dimac. It mo |
| HTTPGet | String | szURL, [szAuthentication], [dwMaxSize] | Retreives an URL and returns the HTTP output from it. Optional authorization and Maximum allowed returned size. |
| Ping | Integer | szHostName | Returns teh number of miliseconds for a Ping reply of a specified host. Negative value indicates error. |

# Reference

Here follows expanded explanations of the functions in the package per object.

## Object references used in the examples

Several methods/functions have examples of usage. Where these are provided they assume the existence of an object instance previously in the code. Examples (in ASP/JavaScript) of such creation with names of the instances like the ones used in the examples:

```
var StringUtils = Server.CreateObject( 'w3.StringUtils' );
var CountryUtils = Server.CreateObject( 'w3.CountryUtils' );
var GeneratorUtils = = Server.CreateObject( 'w3.GeneratorUtils' );
var ValidationUtils = Server.CreateObject( 'w3.ValidationUtils' );
var DateUtils = Server.CreateObject( 'w3.DateUtils' );
var NetUtils = Server.CreateObject( 'w3.NetUtils' );
```

## Complete Reference

### w3.stringUtils

#### *AdjustLineBreaks*

Signature:

```
AdjustLineBreaks(szString) : String
```

Description:

Adjusts all linebreaks in a given string so that all breaks are valid CRLF sequences.

This is useful for ensuring that the text will have it's linebreaks interpreted correctly and look as it should in any environment.

Example:

```
var correctLineBreaks = StringUtils.AdjustLineBreaks( stringWithIncorrectBreaks );
```

#### *BuffToHex*

Signature:

```
BuffToHex(szString) : String
```

Description:

Converts a datastring to a Hex sequence.

Can be used for dumping memory, converting blobs into SQL usable strings.

Example:

```
SQL = "INSERT INTO BlobTable (BlobField)\r\n" +
```

```
"VALUES ( " + buffToHex( BinaryString )+ " )";
```

### *ContentType*

Signature:

```
ContentType(FileName) : String
```

Description:

Returns the content-type given a filename

Checks registry on computer for filetype information.

Example:

```
var contType = StringUtils.ContentType( 'afile.gif' );
```

### *ConvertBase*

Signature:

```
ConvertBase(dwNum, bBase) : String
```

Description:

Returns a number representing the given value (dwNum) in the base supplied (bBase). In essence converts the base of a number to a user defined base.

Example:

```
StringUtils.ConvertBase( 4642160200, 16 ); // Returns a Hexadecimal string
```

### *Currency2String*

Signature:

```
Currency2String(Cur) : String
```

Description:

Formats a currency variable to string using the locale of the computer.

Given a currency value in the format specified in the computer control panel, Currency2String returns the value as a plain number in a string.

See also the String2Currency function.

Example:

```
var aCurrency = new String ( '1,225,125.00' );
var stringOfValue = StringUtils.Currency2String( aCurrency );
```

### *CutText*

Signature:

```
CutText(szString, szFrom, szTo) : String
```

Description:

Cuts text from a string to a string literal.

Finds the string szFrom in the given string (szString) and cuts everything from there to szTo.

Example:

```
var aDocument = '\<HTML\>A Document of some insignificance.\<BR\>\<\/HTML\>';
var bDocument = StringUtils.CutText( aDocument, "\<HTML\>", "\<\/HTML\>" );

// aDocument: <HTML>A Document of some insignificance.<BR></HTML>
// bDocument: A Document of some insignificance.<BR>
```

### *DemaskString*

Signature:

```
DemaskString(szString, Mask) : String
```

Description:

Mask a string and removes all characters defined in MASK.

Useful for removing unwanted characters in a code in order to extract it's value, among other things.

Example:

```
var aCode = '123-4567-89912';
var valueOfaCode = StringUtils.DeMaskString( aCode, '-' );

// valueOfaCode now contains '123456789912'
```

### *ExtractQuotedStr*

Signature:

```
ExtractQuotedStr(szString, [QuoteChar]) : String
```

Description:

Unquotes a string that have been QuotedStr:ed

Basically is an undo/reversal of the StringUtils.QuotedStr method. See QuotedStr below for furter details.

Example:

```
var aString = '\'Here is a string with simple quotes.\'';
var fixedString = StringUtils.QuotedStr( aString );
var unFixedString = StringUtils.ExtractQuotedStr( fixedString, '\'' );
```

### *FormatCurrency*

Signature:

```
FormatCurrency(Cur, Pattern) : String
```

Description:

Formats a currency according to the pattern specified.

The pattern is composed according to the following:

0 Digit placeholder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.

# Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.

. Decimal point. The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator global variable. The default value of DecimalSeparator is specified in the Number Format of the International section in the Windows Control Panel.

, Thousand separator. If the format string contains one or more ',' characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of ',' characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator global variable. The default value of ThousandSeparator is specified in the Number Format of the International section in the Windows Control Panel.

E+ Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-'
E- are contained in the format string, the number is formatted using
e+ scientific notation. A group of up to four '0' characters can
e- immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.

'xx' Characters enclosed in single or double quotes are output as-is, and
"xx" do not affect formatting.

; Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section - The format string applies to all values.

Two sections - The first section applies to positive values and zeros, and

the second section applies to negative values.

Three sections - The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

Example:

```
var aNumber = new Number ( 1234569 );
var currencyOfNumber = StringUtils.FormatCurrency( aNumber, '### ### ###.##' );


// basically use # and whatever you like.
// Note however how the character you use for comma is dependent on your system
// settings. In this example '.' is used
```

### *HashString*

Signature:

```
HashString(szString) : Integer
```

Description:

Calculates a Hash value of a string

Example:

```
aString = 'Test string';
var hashValue = StringUtils.HashString ( aString );

// hashValue = 66450
```

### *HasWildcards*

Signature:

```
HasWildcards(szString) : Integer
```

Description:

Returns the index of the first wildcard in a string. If none is present then zero is returned.

Wildcards are for example the asterisk '*'.

Example:

```
var wildString = 'adads*dsad';
var firstWild = StringUtils.HasWildCards( wildString ); // firstWild = 6 after this
```

### *Hex*

Signature:

```
Hex(dwNumber, [NumChars]) : String
```

Description:

Converts a numeric variable to Hexadecimal.

NumChars is an optional parameter specifying the length of the string returned. It fills with zeroes to the specified length.

Example:

```
var aNumber = Number ( 770217 );
var aHexNumber = StringUtils.Hex( aNumber, 8 ); // aHexNumber = '000BC0A9'
```

### *HexDump*

Signature:

```
HexDump(szString) : String
```

Description:

Produces a Hexdump of a datastring.

Output is fixed formatted using spaces and not tab-delimited.

For HTML output use <PRE></PRE> tags and a fixed width font.

Example:

```
var dumpString = 'asdaasdaasdaasdaasdaasda';
var hexDump = StringUtils.HexDump( dumpString );

// produces the following:
// 00000000  61 73 64 61 61 73 64 61 61 73 64 61 61 73 64 61  asdaasdaasdaasda
// 00000010  61 73 64 61 61 73 64 61                          asdaasda
```

### *HTMLEncode*

Signature:

```
HTMLEncode(szString, [ProcessHTTPLinks]) : String
```

Description:

HTMLEncodes text.

This function takes care of substituting international characters (such as 'ÅÄÖÜ') with their respective HTML codes (such as 'Å').

If the optional parameter ProcessHTTPLinks is supplied as true the method also takes any text that is a HTTP URL and puts an anchor tag with the URL as HREF value in it.

Example:

```
var htmlText = 'Characters: åäö and a link: http://www.dimac.net';
var codeText = StringUtils.HTMLEncode ( htmlText, true );

// codeText gets the content:
// 'Characters: &aring;&auml;&ouml; and a link: <A HREF="http://
www.dimac.net">http://www.dimac.net</A>'
```

### *HTMLFormatBreaks*

Signature:

```
HTMLFormatBreaks(szString) : String
```

Description:

Reformats a string and inserts a <BR> before each CRLF sequence.

Good for HTML use when text data is retrieved from a database and published on the web.

Example:

```
var htmlText = 'A break: \r\n...and a continuing text.';
var codeText = StringUtils.HTMLFormatBreaks ( htmlText );

//  codeText contents becomes:
// A break: <BR>
// ...and a continuing text.
```

### *Indent*

Signature:

```
Indent(szString, Columns) : String
```

Description:

Indents a string.

Simply puts as many spaces as the number supplied ('Columns') in front of each line.

Use <PRE> tags to accomplished desired result on a HTML page.

Example:

```
var indentedText = StringUtils.Indent( unindentedText, 10 );
```

### *JscriptEncode*

Signature:

```
JscriptEncode(szString) : String
```

Description:

JScriptEncode encodes a string so that it can be used within a JScript/JavaScript statement.

It converts the cr, lf, tab characters to \r, \n and \t etc.

Example:

```
var encodedText = StringUtils.JScriptEncode( aText );
```

### *Ltrim*

Signature:

```
Ltrim(szString) : String
```

Description:

Trims all breakable characters on the left side of a string.

Example:

```
var unTrimmed = '      asdasd asdasd adsd';
var leftTrimmed = StringUtils.LTrim ( unTrimmed );

// leftTrimmed contains: 'asdasd asdasd adsd'
```

### *MakeISO8859_1*

Signature:

```
MakeISO8859_1(szString) : String
```

Description:

Encodes a string in ISO8859-1 format.

Can be useful in SMTP Mail headers and so on.

Example:

```
var encodedString = StringUtils.MakeISO8859_1( aString );
```

### *MaskString*

Signature:

```
MaskString(szString, Mask) : String
```

Description:

Masks a string to only contain characters specified in MASK

Useful to extract only essential characters from some code or other string.

Example:

```
var unMasked = '82 13 45 - 6712 - 787';
var maskResult = StringUtils.MaskString( unMasked, '0123456789' );

// maskResult contains: '8213456712787'
```

### *Match*

Signature:

```
Match(szString, Pattern) : Boolean
```

Description:

Matches two strings. Pattern can contain wildcards.

Example:

```
var eMail = 'john.doe@domain.doe';
var isMatch = StringUtils.Match( eMail, "*@*.*" );

// isMatch will be true
```

### *MimeDecode*

Signature:

```
MimeDecode(szString) : String
```

Description:

Decodes a string encoded with MimeEncode or any other base64 encoded string.

Example:

```
var decodedText = StringUtils.MimeDecode( encodedText );
```

### *MimeEncode*

Signature:

```
MimeEncode(szString) : String
```

Description:

Base64 encodes a string of data

Example:

```
var encodedText = StringUtils.MimeEncode( aText );
```

### *QPDecode*

Signature:

```
QPDecode(szString) : String
```

Description:

Decodes a Quoted-Printable encoded string.

Example:

```
var decodedText = StringUtils.QPDecode( encodedText );
```

### *QPEncode*

Signature:

```
QPEncode(szString) : String
```

Description:

Applies Quoted-Printable encoding to a string

Example:

```
var encodedText = StringUtils.QPEncode( aText );
```

### *QuotedStr*

Signature:

```
QuotedStr(szString, [QuoteChar]) : String
```

Description:

Creates a Quoted string ready for use with SQL statements etc.

This function does two things: Puts quotes around the string and codes any characters inside of the string correctly, such as quotes and so on.

It is useful when putting data from a form into a database, for example.

Example:

```
// retrieving userdata from a HTML form

var name = StringUtils.QuotedStr( Request.Form( 'name' ).item );
var email = StringUtils.QuotedStr( Request.Form( 'email' ).item );

sql = 'SELECT ID, Address FROM users WHERE name = ' + name + ' AND email = ' +
email;
```

### *Replace*

Signature:

```
Replace(szString, SearchFor, ReplaceWith) : String
```

Description:

Replaces all substrings specified with another.

Searches the szString for any occurences of the SearchFor string and replaces them with the ReplaceWith string.

Example:

```
var aLongString = 'John A Johnsson. John has been employed since 1991. John is a
good guy.';
var anotherString = StringUtils.Replace( aLongString, 'John', 'Eric' );

// aLongString and anotherString contain:
// John A Johnsson. John has been employed since 1991. John is a good guy.
// Eric A Ericsson. Eric has been employed since 1991. Eric is a good guy.
```

### *ReverseString*

Signature:

```
ReverseString(szString) : String
```

Description:

Reverses a string.

Example:

```
var aLongString = 'John A Johnsson. John has been employed since 1991. John is a
good guy.';
var reversedString = StringUtils.ReverseString( aLongString );


// reversedString contains:
// '.yug doog a si nhoJ .1991 ecnis deyolpme neeb sah nhoJ .nossnhoJ A nhoJ'
```

### *Rtrim*

Signature:

```
Rtrim(szString) : String
```

Description:

Trims all breakable characters to the right of a string.

See also LTrim.

Example:

```
var unTrimmed = '      asdasd asdasd adsd       ';
var rightTrimmed = StringUtils.RTrim ( unTrimmed );

// rightTrimmed contains: '     asdasd asdasd adsd'
```

### *Soundex*

Signature:

```
Soundex(szString) : String
```

Description:

Calculates the soundex value of a string

Example:

```
var aLongString = 'John A Johnsson. John has been employed since 1991. John is a
good guy.';
var sValue = StringUtils.Soundex( aLongString );

// sValue contains: J525
```

### *String2Currency*

Signature:

```
String2Currency(szString) : Currency
```

Description:

Converts a Currency string to a Currency variable.

See also the Currency2String function.

Example:

```
var aCurrency = ( '$12251.25' );
var aCurrency = StringUtils.String2Currency( aCurrencyString );
```

### *Trim*

Signature:

```
Trim(szString) : String
```

Description:

Trims all breakable characters of the beginning and end of a string

A combination of LTrim and RTrim.

Example:

```
var unTrimmed = '      asdasd asdasd adsd      ';
var Trimmed = StringUtils.Trim ( unTrimmed );

// rightTrimmed contains: 'asdasd asdasd adsd'
```

### *TrimQuotes*

Signature:

```
TrimQuotes(szString) : String
```

Description:

Trims quotes of a String

Function addresses (“)-quotes.

Example:

```
var stringWithQuotes = '“Hello there George!”';
var stringWithout = StringUtils.TrimQuotes( stringWithQuotes );

// String values are:
// “Hello there George!”
// Hello there George!
```

### *WrapText*

Signature:

```
WrapText(szString, ColWidth, [JustifyMargins], [Columns], [ColumnDelimiter]) :
String
```

Description:

Wraps text into columns with options to justify margins, multicolumar text etc.

The text is formatted using spaces.

Example:

```
wrappedText = StringUtils.WrapText( szMyText, 50, true, 2, “   “ );
```

## w3.CountryUtils

### *LookupCountryCodeByName*

Signature:

```
LookupCountryCodeByName(CountryCode) : String
```

Description:

Looks up country code using the name supplied.

Example:

```
var aCountry = ‘Sweden’ ;
var countryCode = CountryUtils.LookupCountryCodeByName( aCountry );

// countryCode: ‘SE’
```

### *LookupCountryCodeByNumber*

Signature:

```
LookupCountryCodeByNumber(CountryNumber) : String
```

Description:

Returns the country code for the given country number

Example:

```
var aCountryNumber = 752 ;
var countryCode = CountryUtils.LookupCountryCodeByNumber( aCountryNumber );

// countryCode: ‘SE’
```

### *LookupCountryNameByCode*

Signature:

```
LookupCountryNameByCode(CountryCode) : String
```

Description:

Returns the country name for the given country code

Example:

```
var aCountryNumber = 752;
var aCountry = CountryUtils.LookupCountryCodeByName( aCountryNumber );

// aCountry: 'Sweden'
```

### *LookupCountryNameByNumber*

Signature:

```
LookupCountryNameByNumber(CountryNumber) : String
```

Description:

Returns the country name for the given country number

Example:

```
aCountryNumber = 752;
aCountry = CountryUtils.LookupCountryNameByNumber( aCountryNumber );

// aCountry: 'Sweden'
```

### *LookupCountryNumberByCode*

Signature:

```
LookupCountryNumberByCode(CountryCode) : Integer
```

Description:

Returns the country name for the given country code

Example:

```
aCountryCode = 'SE';
aCountryNumber = CountryUtils.LookupCountryNumberByCode( aCountryCode );

// aCountryNumber: 752
```

### *LookupCountryNumberByName*

Signature:

```
LookupCountryNumberByName(CountryName) : Integer
```

Description:

Returns the country number for the given country name

Example:

```
aCountry = 'Sweden';
aCountryNumber = CountryUtils.LookupCountryNumberByName( aCountry );

// aCountryNumber: 752
```

## w3.GeneratorUtils

### *GenerateGUID*

Signature:

```
GenerateGUID([Stripped]) : String
```

Description:

Returns a GUID, Optinally stripped from all special characters.

The function simply generates a GUID.

Example:

```
var uid = GeneratorUtils.GenerateGUID( true );
```

### *GeneratePassword*

Signature:

```
GeneratePassword(dwLength) : String
```

Description:

Given a length, generates a close-to-unique string that can be used as a password etc.

Example:

```
var pwd = GeneratorUtils.GeneratePassword( 8 );
```

### *GenerateSemiUniqueString*

Signature:

```
GenerateSemiUniqueString(dwLength, [stringClass]) : String
```

Description:

Given a length, generates a close to unique string that can be used as password etc.

Example:

```
var aString = GeneratorUtils.GenerateSemiUniqueString( 8 );
```

## w3.ValidationUtils

### *CheckEmailAddress*

Signature:

```
CheckEmailAddress(EmailAddress, [DeepScan]) : Boolean
```

Description:

Checks to see if an email address is valid.

If DeepScan is set to true, it actually checks if the user is valid in the specified domain.

Example:

```
mailOk = validationUtils.CheckEmailAddress( “support@dimac.net”, true );

// would return ‘true’
```

### *CheckSumMod10*

Signature:

```
CheckSumMod10(Number) : Byte
```

Description:

Method used by several of the other methods in the package.

Example:

```
var checkSum = ValidationUtils.CheckSumMod10( aNumber );
```

### *GetCreditcardType*

Signature:

```
GetCreditcardType(CCNumber) : String
```

Description:

Given a string, returns the type of creditcard.

Supports: VISA, Mastercard, AmEx.

Example:

```
var cardNo = Request.Form( ‘creditCard’ ).item;

var cardType = ValidationUtils.GetCreditcardType( cardNo );

if( cardType == ‘VISA’ )
{
  alert( ‘This is a VISA card’ );
}
```

### *SE_CheckBankgironr*

Signature:

```
SE_CheckBankgironr(szBGNR) : Boolean
```

Description:

Checks a swedish ‘bankgiro’ number

Example:

```
var accNo = Request.Form ( ‘accNumber’ ).item;

if ( !ValidationIUtils.SE_CheckBankGironr( accNo ) )
{
  errMsg = ‘The entered account number is not valid!’;
  ok = 0;
}
```

### *SE_CheckOrgnr*

Signature:

```
SE_CheckOrgnr(szOrgNr) : Boolean
```

Description:

Checks a swedish organisation number

Example:

```
var orgNo = Request.Form ( 'orgNumber' ).item;

if ( !ValidationIUtils.SE_CheckOrgnr( orgNo ) )
{
  errMsg = 'The entered organisation number is not valid!';
  ok = 0;
}
```

### *SE_CheckPostgironr*

Signature:

```
SE_CheckPostgironr(szPGNr) : Boolean
```

Description:

Checks a swedish postal giro account number

Example:

```
var accNo = Request.Form ( 'accNumber' ).item;

if ( !ValidationIUtils.SE_CheckPostGironr( accNo ) )
{
  errMsg = 'The entered account number is not valid!';
  ok = 0;
}
```

### *SE_CheckPsnr*

Signature:

```
SE_CheckPsnr(szPSNR) : Boolean
```

Description:

Checks the given swedish personal number

Is similar to the US Social Security number

Example:

```
var persNo = Request.Form ( 'personalNumber' ).item;

if ( !ValidationIUtils.SE_CheckPsnr( persNo ) )
{
  errMsg = 'The entered personal number is not valid!';
  ok = 0;
}
```

w3.DateUtils

### *DateTimeToGMT*

Signature:

```
DateTimeToGMT(dDate) : String
```

Description:

Converts a dateTime to a GMT string

Example:

```
var aDate = new Date();
var gmtDate = DateUtils.DateTimeToGMT( aDate.getVarDate() );

// gmtDate gets something like: 'Tue, 8 Dec 1998 09:59:16 +0100'
```

### *FormatDateTime*

Signature:

```
FormatDateTime(Mask, dDate) : String
```

Description:

Formats a dateTime using a user defined mask.

Example:

```
var dateFormatted = DateUtils.FormatDateTime( "yyyy-mm-dd hh:mm",
aDate.getVarDate() );

// note usage of getVarDate()
```

### *GetGMTOffset*

Signature:

```
GetGMTOffset( ) : Integer
```

Description:

Returns the offset from GMT

Example:

```
var gmtOffset = DateUtils.GetGMTOffset();
```

### *GetTimeZone*

Signature:

```
GetTimeZone( ) : String
```

Description:

Returns the 'name' of the current timezone

Example:

```
var timeZone = DateUtils.GetTimeZone();

// returns '+0100' for example
```

### *GMTToDateTime*

Signature:

```
GMTToDateTime(szGMTDateString) : DateTime
```

Description:

Converts a GMT string to DateTime

Example:

```
var aDateTime = DateUtils.GMTToDateTime( gmtDate );

// returns something like: '12/8/98 10:02:30 AM'
```

### *IncDay*

Signature:

```
IncDay(dtDate, dwCount) : DateTime
```

Description:

Increases the given date (dtDate) with dwCount number of days

Example:

```
var aDate = new Date();
var inTwoDays = new Date( DateUtils.IncDay( aDate.getVarDate(), 2 ) );

// note the usage of getVarDate()
```

### *IncMonth*

Signature:

```
IncMonth(dtDate, dwCount) : DateTime
```

Description:

Increases the given date (dtDate) with dwCount number of months

Example:

```
var aDate = new Date();
var inTwoMonths = new Date( DateUtils.IncMonth( aDate.getVarDate(), 2 ) );

// note the usage of getVarDate()
```

### *IncWeek*

Signature:

```
IncWeek(dtDate, dwCount) : DateTime
```

Description:

Increases the given date (dtDate) with dwCount number of weeks

Example:

```
var aDate = new Date();
```

```
var inTwoWeeks = new Date( DateUtils.IncWeek( aDate.getVarDate(), 2 ) );

// note the usage of getVarDate()
```

### *IncYear*

Signature:

```
IncYear(dtDate, dwCount) : DateTime
```

Description:

Increases the given date (dtDate) with dwCount number of years

Example:

```
var aDate = new Date();
var inTwoYears = new Date( DateUtils.IncYear( aDate.getVarDate(), 2 ) );

// note the usage of getVarDate()
```

### *WeekByDate*

Signature:

```
WeekByDate(dtDate) : Byte
```

Description:

Returns the week number given a date.

Using the week number is very common for planning purposes in for example Sweden, whereas it is virtually a nonexistant practice in the US. This can be seen in the absence of week numbering in US calendars and the usage of such nunbering in Swedish calendars.

Example:

```
var aDate = new Date();
var theWeek = DateUtils.WeekByDate( aDate.getVarDate() );

// note the usage of getVarDate()
```

## w3.NetUtils

### *DebugWrite*

Signature:

```
DebugWrite(szString) :
```

Description:

Writes a string to NetDebug (tm).

Please see separate chapter on NetDebug for further informatioon.

Example:

```
var pageName = 'Examples Page';
NetUtils.DebugWrite( 'Now executing: ' + pageName );

// writes 'Now executing: Examples page' in the NetDebug window
```

### *HTTPGet*

Signature:

```
HTTPGet(szURL, [szAuthentication], [dwMaxSize]) : String
```

Description:

Retreives an URL and returns the HTTP output from it.

You can optionally provide authorization information and maximum allowed return size.

Example:

```
var aPage = NetUtils.HTTPGet( 'http://www.news.com/', 'userName:password' , 1024 *
16 );


// aPage would contain the whole HTTP output including actual page
```

### *Ping*

Signature:

```
Ping(szHostName) : Integer
```

Description:

Returns the number of miliseconds for a Ping reply of a specified host.

Negative value indicates error.

Example:

```
var pingValue = NetUtils.Ping( "duplo.org" );
```

# Software License Agreement and Limited Warranty

**Notice to user:**
Please read this License Carefully. This is a legal agreement between the end user (”Licensee”) and DIMAC AB. The enclosed software and documentation are licensed by DIMAC AB to the original individual customer for use only on the terms described in this License Agreement (this ”License”). Opening the enclosed CD envelope and / or using the Software indicates that the end user accepts and agrees to comply with these terms.

**1. GRANT OF LICENSES.**
(a) DIMAC AB hereby grants to Licensee a non-exclusive, nontransferable, license (without the ability to sublicense) to use this product and make one copy of the Software in machine-readable form for backup purposes.

(b) DIMAC AB retains title to the Software in all forms whatsoever.

(c) All rights not expressly granted herein are reserved by DIMAC AB.

**2. LICENSE FEES.**
This license shall have no force or effect unless and until Licensee shall have submitted to DIMAC AB all applicable license fees in full. All such fees are exclusive of any taxes, duties, licenses, fees, excises or tariffs now or hereafter imposed on Licensee’s production, licensing, sale, transportation, import, export or use of the Software or Licensee Programs, all of which shall be the responsibility of Licensee.

**3. LIMITED WARRANTY.**
(a) DIMAC AB warrants that for one (1) year following delivery of the Software to Licensee, the Software, unless modified in any way by Licensee, will perform substantially the functions described in any associated product documentation provided by DIMAC AB. DIMAC AB does not warrant that the Software will meet Licensee’s specific requirements or that operation of the Software will be uninterrupted or error-free. DIMAC AB is not responsible for any problem, including any problem which would otherwise be a breach of warranty, caused by :

* Changes in the operating characteristics of computer hardware or computer operating systems.
* Interaction of the Software with software not supplied or approved by DIMAC AB.
* Accident, abuse, or misapplication.

(b) DIMAC AB’s entire liability and Licensee’s sole remedy under the foregoing warranty during the warranty period is that DIMAC AB shall, at its sole and exclusive option, either use reasonable efforts to correct any reported deviation from the relevant product documentation, replace the Software with a functionally equivalent program, or refund all license fees paid, in which case, this License shall immediately terminate. Any repaired or replaced Software will be rewarranted for an additional ninety (90) day period, unless subsequently modified by Licensee.

(c) The Above warranties are exclusive and no other warranties are made by DIMAC AB or its licensors, Whether expressed or implied, including the implied warranties of merchantability, fitness for a Particular purpose, or noninfringement.

**4. LIMITATION OF LIABILITY.**
Under no circumstances shall DIMAC AB be liable for any incidental, special or consequential Damages, even if DIMAC AB has been advised of the possibility of such damages.

In no event shall DIMAC AB's total liability to Licensee for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by Licensee for the Software.

**5. BREACH AND TERMINATION.**
(a) This License is effective until terminated. This License may be terminated by the non-defaulting party if either party materially fails to perform or comply with this License or any provision hereof.

(b) Termination due to a breach of Section 6 shall be effective upon notice. In all other cases termination shall be effective thirty (30) days after notice of termination to the defaulting party if the defaults have not been cured within such thirty (30) day period. The rights and remedies of the parties provided herein shall not be exclusive and are in addition to any other rights and remedies provided by law or this Agreement.

(c) Upon termination of this Agreement, all rights and licenses granted hereunder shall immediately terminate and all Software and other Proprietary Information of DIMAC AB in the possession of Licensee or under its control, shall be immediately returned to DIMAC AB. End user licenses properly granted pursuant to this Agreement and prior to termination of this Agreement shall not be diminished or abridged by the termination of this Agreement.

**6. GOVERNING LAW.**
This Agreement is governed by Swedish law and you submit to the jurisdiction of the Swedish court in relation to any matter or dispute arising hereunder.

**7. NOTICE TO UNITED STATES GOVERNMENT END USERS.**
The Software and Documentation:

* Was developed with no government funds.

* Is a trade secret of DIMAC AB for all purposes of the Freedom of Information Act.

* Are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation" as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §227.7202-1 through §227.7202-4, as applicable, The "Commercial Computer Software" and "Commercial Computer Software Documentation" are being licensed to U.S. Government end users (i) only as "Commercial Items" and (ii) with only those rights granted to all other end users pursuant to the terms and conditions herein.

For units of the Department of Defense (DOD), this Software is sold only with "Restricted Rights" as that term is defined in the DOD Supplement to the Federal Acquisition Regulations ("DFARS") 52.227-7013 (c)(1)(ii) and use, duplication or disclosure is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DF ARS 52.227-7013. Manufacturer: DIMAC AB.


Dimac AB, Box 22228, S-250 24
Helsingborg, Sweden.