

# ObjMem Library

#### 1. Introduction

**ObjAsm** components and projects take advantage from optimized assembler routines. This document describes those functions and procedures located in the open-source static library called **ObjMem.lib**. This precompiled code can be used in an OOP or plain assembler environment. All necessary files are provided in the **ObjAsm** package.

Implementation details can be checked at the source code level by reading the comments on each file.

Source code directories are divided in **32** bit, **64** bit and **Common** code. This last directory contains the bitness, platform, and/or string encoding independent code. The filename suffix of those files indicates their purpose.

The **ObjMem** library comes with an include file called **ObjMem.inc**, that conditional assembly to ensure that for each assembly target only the intended functions are visible. It also selects accordingly the matching equates and string encoding aliases.

## 2. Contents

| 1. | Introduction     | 2 |
|----|------------------|---|
| 2. | Contents         | 3 |
|    | Acknowledgements |   |
|    | Nomenclature     |   |
| 5. | Abbreviations    | 5 |
| 6. | Library Build    | 5 |
| 7. | Notes            | 5 |
| 8. | 32 Bit Code      | 6 |
|    | 64 Bit Code      |   |

# 3. Acknowledgements

I would like to express my very great appreciation to all whose valuable and constructive contributions made this work possible. Thank you!

Corrections, comments, suggestions, contributions, etc. may be sent to the MASM32 Forum, or directly mailed to:

ObjAsm@gmx.net

G. Friedrich,

August, 2022

#### 4. Nomenclature

The following list describes the rules used to create the library:

- 1. X prefixes are used to denote a variable or register that can change according to the bitness assembly target. Example: xax means rax in 64 bits, while eax in 32 bits.
- 2. T file suffixes are used to denote a neutral string encoding.
- 3. X file suffixes are used to denote bitness-neutral code.
- 4. P file suffixes denote platform-independent code, usually leaf procedures.
- 5. Other file suffixes were used to identify the purpose of the code.

#### 5. Abbreviations

BNC: Bitness Neutral Code

COM: Component Object Model

DLL: Dynamic Link Library

GUID: Globally Unique Identifier

HLL: High Level Language

ID: Identifier

IID: Interface Identifier

ZTC: Zero Terminating Character

→: Pointer to

### 6. Library Build

There are some .cmd files in the main directory of ObjMem that make the build easier. In most cases, **MakeObjMem3264.cmd** will do the job.

The **ObjMem.api** file contains the API definitions for the **RadASM** 2.x autocomplete feature. It must be copied to the ...\RadASM\Masm folder and RadASM restarted.

**Note:** The BuildTest.cmd file is for procedure testing purposes only. Edit the file with the filename you want to check.

#### 7. Notes

The links in Sections 8 and 9 assume that this document is on the same drive as the **ObjAsm** installation and that it was done at the root level of the drive, e.g. D:\ObjAsm\...

#### 8. 32 Bit Code

Procedure: aCRC32C

\ObjAsm\Code\ObjMem\32\aCRC32C.asm Files:

\langle ObjAsm\Code\ObjMem\Common\aCRC32C\_XP.inc Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an aligned Purpose:

memory block.

Arguments: Arg1: → Aligned memory block.

Arg2: Memory block size in BYTEs.

Return: eax = CRC32C.

Procedure: ActivatePrevInstanceA

eax = TRUE if activated, otherwise FALSE. Return:

Procedure: ActivatePrevInstanceW

\ObjAsm\Code\ObjMem\32\ActivatePrevInstanceW.asm

Purpose: Activate a previously existing instance of an application. Arguments: Arg1: → WIDE application name.

Arg2: → WIDE class name. eax = TRUE if activated, otherwise FALSE. Return:

Procedure: AreVisualStylesEnabled
File: \( \ObjAsm\Code\ObjMem\32\AreVisualStylesEnabled.asm \)

Determine if there is an activated theme for the running application Purpose:

Arguments: None.

Return: eax = TRUE if the application is themed, otherwise FALSE.

Procedure: bin2dwordA

\ObjAsm\Code\ObjMem\32\bin2dwordA.asm

Purpose: Purpose: Load an ANSI string binary representation of a DWORD.

Arguments: Arg1: → ANSI binary string.

eax = DWORD.Return:

Procedure: bin2dwordW

File:

\ObjAsm\Code\ObjMem\32\bin2dwordW.asm
Load an WIDE string binary representation of a DWORD. Purpose:

Arguments: Arg1: → Wide binary string.

eax = DWORD.Return:

Procedure: bin2gwordA

\ObjAsm\Code\ObjMem\32\bin2gwordA.asm File:

Purpose: Load an ANSI string binary representation of a QWORD.

Arguments: Arg1: → ANSI binary string.

Return: edx::eax = QWORD.

Procedure: bin2qwordW

File: \ObjAsm\Code\ObjMem\32\bin2qwordW.asm

Purpose: Compute a WIDE string binary representation of a QWORD. Arguments: Arg1:  $\rightarrow$  WIDE binary string.

edx::eax = QWORD.Return:

Procedure: Bmp2Rqn

\ObjAsm\Code\ObjMem\32\Bmp2Rqn.asm Create a GDI region based on a device dependant or independent bitmap (DDB or DIB). Purpose:

This region is defined by the non transparent area delimited by the transparent color.

Arguments: Arg1: Bitmap handle.

Arg2: RGB transparet color.

eax = Region handle or zero if failed. Return:

Procedure: BStrAlloc

\ObjAsm\Code\ObjMem\32\BStrAlloc.asm File:

Allocate space for a BStr with n characters. The length field is set to zero. Purpose:

Arguments: Arg1: Character count.

Return:  $eax \rightarrow New$  allocated BStr or NULL if failed. Procedure: BStrCat

\ObjAsm\Code\ObjMem\32\BStrCat.asm

Purpose: Concatenate 2 BStrs. Arguments: Arg1: Destrination BStr. Arg2: Source BStr.

Return: Nothina.

Procedure: BStrCatChar

\ObjAsm\Code\ObjMem\32\BStrCatChar.asm Append a character to the end of a BStr. File: Purpose:

Arguments: Arg1: Destrination BStr.

Arg2: Wide character.

Return: Nothing.

Procedure: BStrCCatChar

File: \ObjAsm\Code\ObjMem\32\BStrCCatChar.asm

Purnose: Append a WIDE character to a BStr with length limitation.

Arguments: Arg1:  $\rightarrow$  Destination BStr. Arg2:  $\rightarrow$  Wide character.

Return: Nothing.

Procedure: BStrCECat

File: \ObjAsm\Code\ObjMem\32\BStrCECat.asm

Concatenate 2 BStrs with length limitation and return the ending zero character Purpose:

address. The destination string buffer should have at least enough room for the

maximum number of characters + 1.

Arg1: → Destination BStr buffer. Arg2: → Source BStr. Arguments:

Arg3: Maximal number of charachters that the destination string can hold including

the zero terminating character.

Return: eax  $\rightarrow$  ZTC.

Procedure: BStrCECopy

File:

\ObjAsm\Code\ObjMem\32\BStrCECopy.asm
Copy the the source BStr with length limitation and return the ZTC address.
The destination buffer should hold the maximum number of characters + 1. Purpose:

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source BStr.

Arg3: Maximal number of charachters.

 $eax \rightarrow ZTC$ . Return:

Procedure: BStrCNew

\ObjAsm\Code\ObjMem\32\BStrCNew.asm File:

Purpose:

Allocate a new copy of the source BStr with length limitation.

If the pointer to the source string is NULL or points to an empty string, BStrCNew returns NULL and doesn't allocate any heap space. Otherwise, StrCNew makes a duplicate of the source string. The maximal size of the new string is limited to the second

parameter.

Arguments: Arg1: → Source BStr.

Arg2: Maximal character count.

Return:  $eax \rightarrow New BStr copy.$ 

Procedure: BStrCopy

\ObjAsm\Code\ObjMem\32\BStrCopy.asm File: Purpose: Copy a BStr to a destination buffer. Arguments: Arg1: Destrination BStr buffer.

Arg2: Source BStr.

Return: Nothing.

Procedure: BStrCScan

\ObjAsm\Code\ObjMem\32\BStrCScan.asm

Scan from the beginning of a BStr for a character with length limitation. Purpose:

Arg1: → Source WIDE string. Arguments:

Arg2: Maximal character count.
Arg3: Wide character to search for.

eax → Character address or NULL if not found. Return:

BStrDispose Procedure:

\ObjAsm\Code\ObjMem\32\BStrDispose.asm File:

Purpose: Free the memory allocated for the string using BStrNew, BStrCNew, BStrLENew or

BStrAlloc.

If the pointer to the string is NULL, BStrDispose does nothing.

Arguments: Arg1:  $\rightarrow$  BStr.

Return: Nothing.

Procedure: BStrECat

File: \ObjAsm\Code\ObjMem\32\BStrECat.asm

Append a BStr to another and return the address of the ending zero character.
BStrCat does not perform any length checking. The destination buffer must have room Purnose:

for at least BStrLength(Destination) + BStrLength(Source) + 1 characters.

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source BStr.

 $eax \rightarrow ZTC$ . Return:

Procedure: BStrECatChar

\ObjAsm\Code\ObjMem\32\BStrECatChar.asm

Append a WIDE character to a BStr and return the address of the ZTC. Purpose:

Append a WIDE Character to a BStr and return the address of the ZTC.

BStrECatChar does not perform any length checking. The destination buffer must have enough room for at least BStrLength(Destination) + 1 + 1 characters.

Arguments: Arg1: \( \rightarrow Destination BStr buffer.

Arg2: \( \rightarrow WIDE character.

Return: eax \( \rightarrow ZTC.

Procedure: BStrECopy

File: \ObjAsm\Code\ObjMem\32\BStrECopy.asm

Purpose: Copy a BStr to a buffer and return the address of the ZTC.

Source and destination strings may overlap.

Arguments: Arg1: → Destination BStr buffer. Arg2: → Source BStr buffer.

Return:  $eax \rightarrow ZTC$ .

Procedure: BStrEnd

File:

\ObjAsm\Code\ObjMem\32\BStrEnd.asm
Get the address of the ZTC that terminates the string. Purpose:

Arguments: Arg1: → Source BStr.

 $eax \rightarrow ZTC.$ Return:

Procedure: BStrEndsWith

\ObjAsm\Code\ObjMem\32\BStrEndsWith.asm Files:

\ObjAsm\Code\ObjMem\Common\BStrEndsWith\_X.inc

Purpose: Compare the ending of a BSTR.

Arguments: Arg1:  $\rightarrow$  Analized BSTR.

Arg2: → Suffix BSTR.

eax = TRUE of the ending matches, otherwise FALSE. Return:

Procedure: BStrFillChr

Files: \ObjAsm\Code\ObjMem\32\BStrFillChr.asm

\ObjAsm\Code\ObjMem\Common\BStrFillChr\_TX.inc

Fill a preallocated BSTR with a character.

Arguments: Arg1: → String. Arg2: Character.

Arg3: Character Count.

Return: Nothing.

Procedure: BStrLeft

File:

\ObjAsm\Code\ObjMem\32\BStrLeft.asm
Extract the left n characters of the source BStr. Purpose:

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source BStr

eax = Number of copied characters, not including the ZTC. Return:

Procedure: BStrLength

\ObjAsm\Code\ObjMem\32\BStrLength.asm

Determine the length of a BStr not including the ZTC. Purpose:

Arguments: Arg1: → Source BStr.

eax = Length of the string in characters. Return:

Procedure: BStrLRTrim

File:

\ObjAsm\Code\ObjMem\32\BStrLRTrim.asm
Trim blank characters from the beginning and the end of a BStr. Purpose:

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source BStr.

Nothing. Return:

Procedure: BStrLTrim

\ObjAsm\Code\ObjMem\32\BStrLTrim.asm

Trim blank characters from the beginning of a BStr. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination BStr buffer. Arg2:  $\rightarrow$  Source BStr.

Return: Nothina.

Procedure: BStrMid

\ObjAsm\Code\ObjMem\32\BStrMid.asm Extract a substring from a BStr string. File: Purpose:

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source Bstr.

Arg3: Start character index. Index ranges [1 .. String length].

Arg3: Character count.

Return: Nothing.

Procedure: BStrMove

\ObjAsm\Code\ObjMem\32\BStrMove.asm File:

Move part of a BStr. The ZTC is not appended automatically. Purpose:

Source and destination strings may overlap.

Arguments:

Arg1: → Destination buffer. Arg2: → Source BStr. Arg3: Character count.

Nothing. Return:

Procedure: BStrNew

\ObjAsm\Code\ObjMem\32\BStrNew.asm Allocate an new copy of the source BStr. File: Purpose:

If the pointer to the source string is NULL or points to an empty string, BStrNew returns NULL and doesn't allocate any heap space. Otherwise, BStrNew makes a duplicate

of the source string.
The allocated space is Length(String) + 1 character.

Arguments: Arg1: → Source BStr. Return:  $eax \rightarrow New BStr copy.$ 

Procedure: BStrRepChr

\ObjAsm\Code\ObjMem\32\BStrRepChr.asm Files:

\ObjAsm\Code\ObjMem\Common\BStrRepChr\_X.inc Purpose: Create a new BSTR filled with a given char.

Arguments: Arg1: Used character.

Arg2: Repetition count. xax → New BSTR or NULL if failed. Return:

Procedure: BStrRight

File: \ObjAsm\Code\ObjMem\32\BStrRight.asm

Purpose: Copy the right n characters from the source string into the destination buffer.

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source BStr.

Arg3: Character count.

Return: Nothing.

Procedure: BStrRTrim

File:

ObjAsm\Code\ObjMem\32\BStrRTrim.asm
Trim blank characters from the end of a BStr. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination BStr buffer. Arg2:  $\rightarrow$  Source BStr.

Return: Nothing.

Procedure: BStrSize

\ObjAsm\Code\ObjMem\32\BStrSize.asm

Determine the size of a BStr including the zero terminating character + leading DWORD. Purpose:

Arguments: Arg1: → Source BStr.

eax = String size including the length field and zero terminator in BYTEs. Return:

Procedure: BStrStartsWith

\ObjAsm\Code\ObjMem\32\BStrStartsWith.asm Files:

\ObjAsm\Code\ObjMem\Common\BStrStartsWith\_X.inc

Purpose: Compare the beginning of a BSTR. Arguments: Arg1:  $\rightarrow$  Analized BSTR.

Arg2: → Prefix BSTR.

eax = TRUE of the beginning matches, otherwise FALSE.

Procedure: byte2hexA

\ObjAsm\Code\ObjMem\32\byte2hexA.asm

Convert a BYTE to its hexadecimal ANSI string representation. Purpose:

Arg1:  $\rightarrow$  Destination ANSI string buffer. Arg2: BYTE value. Arguments:

Nothing. Return:

The destination buffer must be at least 3 BYTEs large to allocate the output string Notes:

(2 character BYTES + ZTC = 3 BYTES).

Procedure: byte2hexW

File: \ObjAsm\Code\ObjMem\32\byte2hexw.asm

Purpose: Convert a BYTE to its hexadecimal WIDE string representation.

Arg1: → Destination WIDE string buffer. Arguments:

Arg2: BYTE value.

Return: Nothing.

The destination buffer must be at least 5 BYTEs large to allocate the output string Notes:

(2 character WORDS + ZTC = 5 BYTES).

Procedure: CalcVarianceDW

\ObjAsm\Code\ObjMem\32\CalcVarianceDW.asm Files: <u>\ObjAsm\Code\ObjMem\Common\CalcVariance\_XP.inc</u>

Purpose: Calculate the MSE of an array of DWORDs.

 $Arg1: \rightarrow Array of DWORDs.$ Arguments:

Arg2: DWORD Array count.

Arg3: → Variance.

Return: eax = TRUE is succeeded, otherwise FALSE.

Links:

https://www.mun.ca/biology/scarr/Simplified\_calculation\_of\_variance.html#:~:text=A%

20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202

https://www.mun.ca/biology/scarr/Mean\_&\_variance.html#:~:text=easily%20calculated%20as  $Var = Y2/N-(Y/N)^2$  or  $(Y2*N-Y^2)/N^2$  where Y: Sum(y), Y2:  $Sum(y^2)$ , N:Population count = Array size.

Formulas:

Procedure: CalcVarianceQW

Files: \ObjAsm\Code\ObjMem\32\CalcVarianceQW.asm

\ObjAsm\Code\ObjMem\Common\CalcVariance\_XP.inc

Calculate the MSE of an array of QWORDs. Purpose:

Arg1:  $\rightarrow$  Array of QWORDs. Arguments: Arg2: DWORD Array count.

Arg3: → Variance.

eax = TRUE is succeeded, otherwise FALSE. Return:

Links:

https://www.mun.ca/biology/scarr/Simplified\_calculation\_of\_variance.html#:~:text=A%

20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202

 $\label{lem:https://www.mun.ca/biology/scarr/Mean_&_variance.html#:~:text=easily%20calculated%20as $$ var = Y2/N-(Y/N)^2 or $(Y2*N-Y^2)/N^2 $$$ 

Formulas:

where Y: Sum(y), Y2:  $Sum(y^2)$ , N:Population count = Array size.

Procedure: CalcVarianceR4

\ObjAsm\Code\ObjMem\32\CalcvarianceR4.asm Files:

\ObjAsm\Code\ObjMem\Common\Calcvariance\_XP.inc Purpose: Calculate the MSE of an array of REAL4s.

Arg1: → Array of REAL4s. Arg2: DWORD Array count. Arguments:

Arg3: → Variance.

Return: eax = TRUE is succeeded, otherwise FALSE.

Links:

https://www.mun.ca/biology/scarr/Simplified\_calculation\_of\_variance.html#:~:text=A%

20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202

https://www.mun.ca/biology/scarr/Mean\_&\_Variance.html#:~:text=easily%20calculated%20as

Formulas:

Procedure: CalcVarianceR8

\ObjAsm\Code\ObjMem\32\CalcvarianceR8.asm Files:

\ObjAsm\Code\ObjMem\Common\CalcVariance\_XP.inc Calculate the MSE of an array of REAL8s.

Purnose:

 $Arg1: \rightarrow Array of REAL8s.$ Arguments: Arg2: DWORD Array count.

Arg3: → Variance.

Return: eax = TRUE is succeeded, otherwise FALSE.

Links:

https://www.mun.ca/biology/scarr/Simplified\_calculation\_of\_variance.html#:~:text=A%

20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202

https://www.mun.ca/biology/scarr/Mean\_&\_Variance.html#:~:text=easily%20calculated%20as

Formulas:

Var =  $Y^2/N-(Y/N)^2$  or  $(Y^2*N-Y^2)/N^2$ where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.

Procedure: CenterForm

File: \ObjAsm\Code\ObjMem\32\CenterForm.asm
Purpose: Calculate the starting coordinate of a window based on the screen and the window size.
Arguments: Arg1: Window size in pixel.

Arg2: Screen size in pixel.

Return: eax = Starting point in pixel.

Procedure: ComEventsAdvice

\ObjAsm\Code\ObjMem\32\ComEventsAdvice.asm

\ObjAsm\Code\ObjMem\Common\ComEventsAdvice\_X.inc

Notificate the Event source that pISink will recieve Events. Purpose:

Arguments: Arg1: → Any Source Object Interface.

Arg2: → Sink IUnknown Interface.

Arg3: → IID of the outgoing interface whose connection point object is being requested (defined by the Source to communicate and implemented by the Sink). Arg4: → ConnectionPoint interface pointer. Arg5: → DWORD Cookie.

Return: eax = HRESULT.

Procedure: ComEventsUnadvice

Files: \ObjAsm\Code\ObjMem\32\ComEventsUnadvice.asm

\ObjAsm\Code\ObjMem\Common\ComEventsUnadvice\_X.inc

Notificate the Event source that pISource will NOT recieve Events any more. Purpose:

Arguments: Arg1: → Previous ConnectionPoint interface.
Arg2: DWORD Cookie received from previous ComEventsAdvice call.

eax = HRESULT.Return:

Procedure: ComGetErrStrA

\ObjAsm\Code\ObjMem\32\ComGetErrStrA.asm Files:

\ObjAsm\Code\ObjMem\Common\ComGetErrStr\_TX.inc

Purpose: Return a description ANSI string from a COM error code.

Arguments: Arg1: COM error code. Return: eax → Error string.

Procedure: ComGetErrStrW

Files: \ObjAsm\Code\ObjMem\32\ComGetErrStrW.asm

\ObjAsm\Code\ObjMem\Common\ComGetErrStr\_TX.inc

Return a description WIDE string from a COM error code. Purpose:

Arguments: Arg1: COM error code.  $eax \rightarrow Error string.$ Return:

Procedure: ComPtrAssign

Files: \ObjAsm\Code\ObjMem\32\ComPtrAssign.asm

\ObjAsm\Code\ObjMem\Common\ComPtrAssign\_X.inc

First increment the reference count of the new interface and then release any Purpose:

existing interface pointer.

Arguments: Arg1: → Old Interface pointer.

Arg2: New Interface pointer.

Procedure: CreatePathA

\ObjAsm\Code\ObjMem\32\CreatePathA.asm Create a path on the destination drive. File: Purpose:

Arguments: Arg1: → ANSI path string.

Return: Nothing.

Procedure: CreatePathW

\ObjAsm\Code\ObjMem\32\CreatePathW.asm Purpose: Create a path on the destination drive.

Arguments: Arg1:  $\rightarrow$  Wide path string.

Return: Nothing.

Procedure: DbqClose

\ObjAsm\Code\ObjMem\32\DbgClose.asm File: Purpose: Close the connection to the output device.

Arguments: None. Return: Nothing. Procedure: DbgConOpen

File: \ObjAsm\Code\ObjMem\32\DbgConOpen.asm

Open a new console for the calling process. Purpose:

Arguments: None.

Return: eax = TRUE if it was opened, otherwise FALSE.

Procedure: DbgLogOpen

\ObjAsm\Code\ObjMem\32\DbgLogOpen.asm File:

Purpose: Open a Log-File.

Arguments: None.

eax = TRUE if it was opened, otherwise FALSE. Return:

Procedure: DbgOutApiErr

\ObjAsm\Code\ObjMem\32\DbgOutApiErr.asm

\ObjAsm\Code\ObjMem\Common\DbgOutApiErr\_X.inc

Identify a API error with a string. Purpose:

Arguments: Arg1: Api error code obtained with GetLastError.
Arg2: → Destination Window WIDE name.

Return: Nothina.

Procedure: DbgOutBitmap

\ObjAsm\Code\ObjMem\32\DbgOutBitmap.asm File: Purpose: Send a bitmap to the Debug Center Window.

Arg1: Bitamp HANDLE. Arguments:

Arg2: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutCmd

\ObjAsm\Code\ObjMem\32\DbgOutCmd.asm File: Purpose: Send a command to a specific Debug window.
Arguments: Arg1: Command ID [BYTE].
Arg2: Target Debug Window WIDE name.

Return: Nothina.

Procedure: DbgOutComErr

Files: \ObjAsm\Code\ObjMem\32\DbgOutComErr.asm

\ObjAsm\Code\ObjMem\Common\DbgOutComErr\_X.inc

Identify a COM error with a string. Purpose:

Arguments: Arg1: COM error ID.
Arg2: → Destination Window WIDE name.

Nothing. Return:

Procedure: DbgOutComponent

\ObjAsm\Code\ObjMem\32\DbgOutComponent.asm File:

Purpose: Identify a COM-Component. Arguments: Arg1: → CSLID.

Arg2: Foreground color.

Arg2: → Destination Window WIDE name.

Procedure: DbgOutComponentName

File: \ObjAsm\code\ObjMem\32\DbgOutComponentName.asm

Purpose: Identify a COM-Component.

Arguments: Arg1: → CSLID.

Arg2: Foreground color.

Arg2: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutFPU

\ObjAsm\Code\ObjMem\32\DbgOutFPU.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutFPU\_XP.inc

Purpose: Display the content of the FPU.

Arguments: Arg1: → Destination Window WIDE name.
Arg2: Text RGB color.

Nothing. Return:

Procedure: DbgOutFPU\_UEFI

\ObjAsm\Code\ObjMem\32\DbgOutFPU\_UEFI.asm Files: \ObjAsm\Code\ObjMem\Common\DbgOutFPU\_XP.inc

Purpose:

Display the content of the FPU. Arg1: → Destination Window WIDE name. Arg2: Text RGB color. Arguments:

Nothing.

Procedure: DbgOutInterface

\ObjAsm\Code\ObjMem\32\DbgOutInterface.asm

Identify a COM-Interface. Purpose:

Arguments: Arg1: → CSLID.

Arg2: Foreground color.

Arg2: → Destination Window WIDE name.

Procedure: DbgOutInterfaceName

\ObjAsm\Code\ObjMem\32\DbgOutInterfaceName.asm File:

Identify a COM-Interface.
Arg1: → CSLID. Purpose:

Arguments:

Arg2: Foreground color. Arg2: → Destination window WIDE name.

Nothing. Return:

Procedure: DbgOutMem

Files:

ObjAsm\Code\ObjMem\32\DbgOutMem.asm
\ObjAsm\Code\ObjMem\Common\DbgOutMem\_XP.inc
Output the content of a memory block.

Purpose:

Arguments: Arg1: → Memory block.

Arg2: Memory block size. Arg3: Representation format. Arg4: Memory output color. Arg5: Representation output color.

Arg6: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutMem UEFI

\Objasm\Code\ObjMem\32\DbqOutMem\_UEFI.asm\Objasm\Code\ObjMem\Common\DbqOutMem\_XP.inc\Objasm\Content of a memory block.

Arg1: \to Memory block. Files:

Purpose:

Arguments: Arg2: Memory block size.

Arg3: Representation format. Arg4: Memory output color. Arg5: Representation output color.

Arg6: → Destination Window WIDE name.

Nothing. Return:

Procedure: DbgOutMsg

Files:

\ObjAsm\Code\ObjMem\32\DbgOutMsg.asm \ObjAsm\Code\ObjMem\Common\DbgOutMsg\_X.inc

Purpose: Identifies a windows message with a string.

Arguments: Arg1: Windows message ID. Arg2: Foreground color.

Arg3: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutTextA

\ObjAsm\Code\ObjMem\32\DbgOutTextA.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutText\_TX.inc Sends an ANSI string to the debug output device. Arg1: → Zero terminated ANSI string. Arg2: Color value. Purpose:

Arguments:

Arg3: Effect value (DBG\_EFFECT\_XXX). Arg4: → Destination window WIDE name.

Nothing. Return:

Procedure: DbgOutTextA\_UEFI

\ObjAsm\Code\ObjMem\32\DbgOutTextA\_UEFI.asm

\ObjAsm\Code\ObjMem\Common\DbgOutText\_TX\_UEFI.inc Sends an ANSI string to the debug output device. Purpose:

Arg1: → Zero terminated ANSI string. Arguments:

Arg2: Color value.

Arg3: Effect value (DBG\_EFFECT\_XXX). Arg4: → Destination window WIDE name.

Nothing. Return:

Procedure: DbgOutTextCA

File:

\ObjAsm\Code\ObjMem\32\DbgOutTextCA.asm
Send a counted ANSI string to the debug output device Purpose:

Arguments: Arg1: → Null terminated ANSI string.

Arg2: Character count.

Arg3: Color value.
Arg4: Effect value (DBG\_EFFECT\_XXX).

Arg5: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutTextCW

File: \ObjAsm\Code\ObjMem\32\DbgOutTextCW.asm

Send a counted WIDE string to the debug output device Purpose:

Arg1: → Null terminated WIDE string. Arguments:

Arg2: Character count.

Arg3: Color value.

Arg4: Effect value (DBG\_EFFECT\_XXX). Arg5: → Destination Window WIDE name.

Return:

Procedure: DbgOutTextW

\ObjAsm\Code\ObjMem\32\DbgOutTextW.asm Files:

Purpose:

Arguments: Arg1:  $\rightarrow$  Zero terminated WIDE string. Arg2: Color value.

Arg3: Effect value (DBG\_EFFECT\_XXX) Arg4: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutTextW\_UEFI

\ObjAsm\Code\ObjMem\32\DbgOutTextW\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutText\_TX\_UEFI.inc

Send a WIDE string to the debug output device. Purpose: Arg1: → Zero terminated WIDE string. Arg2: Color value. Arguments:

Arg3: Effect value (DBG\_EFFECT\_XXX).

Arg4: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgShowObjectHeader

\ObjAsm\Code\ObjMem\32\DbgShowObjectHeader.asm

\ObjAsm\Code\ObjMem\Common\DbgShowObjectHeader\_XP.inc

Output heading object information. Purpose:

Arg1: → Object Name. Arg2: → Instance. Arguments:

Arg3: Text RGB color. Arg3: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgShowObjectHeader\_UEFI

Files: \ObjAsm\Code\ObjMem\32\DbgShowObjectHeader\_UEFI.asm

\ObjAsm\Code\ObjMem\Common\DbgShowObjectHeader\_XP.inc

Purpose: Output heading object information.
Arguments: Arg1: → Object Name.
Arg2: → Instance.

Arg3: Text RGB color. Arg3: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgShowTraceMethod

\ObjAsm\Code\ObjMem\32\DbqShowTraceMethod.asm
Output trace information about a method. File:

Purpose:

Arguments: Arg1: → Method Name.

Arg2: Method count. Arg3: Method ticks.

Arg4: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgWndOpen

\ObjAsm\Code\ObjMem\32\DbgWndOpen.asm File:

Purpose: Open Debug Center instance.

Arguments: None.

Return: eax = TRUE if it was opened, otherwise FALSE.

Procedure: dec2dwordA

\ObjAsm\Code\ObjMem\32\dec2dwordA.asm

Purpose: Convert a decimal ANSI string to a DWORD.

Arg1:  $\rightarrow$  Source ANSI string. Possible leading characters are " ", tab, "+" and "-", followed by a sequence of chars between "0".."9" and finalized by a ZTC. Arguments:

Other characters terminate the convertion returning zero.

eax = Converted DWORD. Return:

ecx = Conversion result. Zero if succeeded, otherwise failed.

Procedure: dec2dwordW

\ObjAsm\Code\ObjMem\32\dec2dwordW.asm Convert a decimal WIDE string to a DWORD. File: Purpose:

Arg1: → Source WIDE string. Possible leading characters are " ", tab, "+" and "-", followed by a sequence of chars between "0".."9" and finalized by a ZTC. Arguments:

Other characters terminate the convertion returning zero.

Return: eax = Converted DWORD.

ecx = Conversion result. Zero if succeeded, otherwise failed.

Procedure: DisableCPUSerialNumber

\ObjAsm\Code\ObjMem\32\DisableCPUSerialNumber.asm File: Disable the reading of the CPU serial number. Purpose:

Arguments: None. Nothing. Return:

Procedure: DllErr2StrA

\ObjAsm\Code\ObjMem\32\DllErr2StrA.asm File:

Purpose: Translate an error code to an ANSI string stored in a DLL.

Arguments: Arg1: Error code.

Arg2: → ANSI character buffer.

Arg3: Buffer size in characters, inclusive ZTC.

Arg4: → DLL ANSI name.

Return: Nothing.

Procedure: DllErr2StrW

\ObjAsm\Code\ObjMem\32\DllErr2StrW.asm File:

Purpose: Translate an error code to a WIDE string stored in a DLL.

Arguments: Arg1: Error code.

Arg2: → WIDE character buffer.

Arg3: Buffer size in characters, inclusive ZTC.

Arg4: → DLL WIDE name.

Nothing. Return:

Procedure: DrawTransparentBitmap

\ObjAsm\Code\ObjMem\32\DrawTransparentBitmap.asm
Draw a bitmap with transparency on a device context. File: Purpose:

Arguments: Arg1: DC handle.
Arg2: Bitmap handle to draw.

Arg3; X start position on DC. Arg4: Y start position on DC.

Arg5: RGB transparent color. Use TBM\_FIRSTPIXEL to indicate that the pixel in the

upper left corner contains the transparent color.

Nothing. Return:

Original source by Microsoft. Note:

"HOWTO: Drawing Transparent Bitmaps (Q79212)"

(http://support.microsoft.com/default.aspx?scid=kb;EN-US;q79212)

Transcribed by Ernest Murphy.

Procedure: dword2binA

\ObjAsm\Code\ObjMem\32\dword2binA.asm File:

Convert a DWORD to its binary ANSI string representation. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: DWORD value.

Nothing. Return:

The destination buffer must be at least 33 BYTEs large to allocate the output string Note:

(32 character BYTES + ZTC = 33 BYTEs).

Procedure: dword2binW

File:

\ObjAsm\Code\ObjMem\32\dword2binW.asm
Convert a DWORD to its binary WIDE string representation. Purnose:

Arguments: Arg1: → Destination buffer.
Arg2: DWORD value.

Return: Nothing.

The destination buffer must be at least 66 BYTEs large to allocate the output string Note:

(32 character WORDS + ZTC = 66 BYTES).

Procedure: dword2hexA

\ObjAsm\Code\ObjMem\32\dword2hexA.asm File:

Convert a DWORD to its hexadecimal ANSI string representation.

Arguments: Arg1: → Destination buffer.

Arg2: DWORD value.

Nothing. Return:

The destination buffer must be at least 9 BYTEs large to allocate the output string Note:

(8 character BYTEs + ZTC = 9 BYTEs).

Procedure: dword2hexW

File:

\ObjAsm\Code\ObjMem\32\dword2hexw.asm
Convert a DWORD to its hexadecimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: DWORD value.

Nothing. Return:

The destination buffer must be at least 18 BYTEs large to allocate the output string Note:

(8 character WORDS + ZTC = 18 BYTES).

Procedure: Err2StrA

\ObjAsm\Code\ObjMem\32\Err2StrA.asm File:

Translate a system error code to an ANSI string. Purpose:

Arg1: Error code. Arguments:

Arg2: → ANSI string buffer.

Arg3: Buffer size in characters, inclusive ZTC.

Return: Nothing.

Procedure: Err2StrW

File: \ObjAsm\Code\ObjMem\32\Err2StrW.asm

Translate a system error code to a WIDE string. Purpose:

Arg1: Error code. Arguments:

Arg2:  $\rightarrow$  WIDE string buffer. Arg3: Buffer size in characters, inclusive ZTC.

Return: Nothing.

Procedure: ErrorMessageBoxA

File:

\ObjAsm\Code\ObjMem\32\ErrorMessageBoxA.asm
Show a Messagebox containing an error string in the locale language and an user str. Purpose:

Arg1: Messagebox parent window HANDLE. Arguments: Arg2: → User ANSI string.

Arg3: Locale ID.

Arg4: API error code.

Return: Nothing.

Procedure: ErrorMessageBoxW

\ObjAsm\Code\ObjMem\32\ErrorMessageBoxW.asm File:

Purpose: Show a messagebox containing an error string in the locale lenguage and an user str.

Arguments: Arg1: Messagebox parent window HANDLE.

Arg2: → User WIDE string. Arg3: Locale ID. Arg4: API error code.

Nothing. Return:

Procedure: FileExistA

\ObjAsm\Code\ObjMem\32\FileExistA.asm Check the existence of a file. File:

Purpose:

Arguments: Arg1:  $\rightarrow$  ANSI file name. Return: eax = TRUE if the file exists, otherwise FALSE.

Procedure: FileExistW

\ObjAsm\Code\ObjMem\32\FileExistW.asm File:

Check the existence of a file.

Arguments: Arg1: → WIDE file name.

Return: eax = TRUE if the file exists, otherwise FALSE. Return:

Procedure: FindFileA

\ObjAsm\Code\ObjMem\32\FindFileA.asm Files:

\ObjAsm\Code\ObjMem\Common\FindFile\_TX.inc

Purpose: Search for a file in a list of paths.

 $Arg1: \rightarrow File name.$ Arguments:

 $Arg2: \rightarrow List$  of path strings. The end of the list is indicated with a ZTC. Arg3:  $\rightarrow$  Buffer containing the full path and file name in which the file was found.

Buffer length = MAX\_PATH.

Return: eax = Number of chars copied to the destination buffer. O if the file was not found. Example: invoke FindFile, \$0fsCStr("free.inc"), \$0fsCStr("\Here\*",0), addr cBuf Search free inc in all \Here and suddirectories.

Procedure: FindFileW

Files: \ObjAsm\Code\ObjMem\32\FindFileW.asm

\ObjAsm\Code\ObjMem\Common\FindFile\_TX.inc

Purpose: Search for a file in a list of paths. Arguments: Arg1: → File name.

Arg2: → List of path strings. The end of the list is indicated with a ZTC.

Arg3: → Buffer containing the full path and file name in which the file was found.

Buffer length = MAX\_PATH.

eax = Number of chars copied to the destination buffer. 0 if the file was not found.

Example: invoke FindFile, \$0fsCStr("free.inc"), \$0fsCStr("\Here\*",0), addr cBuf Return:

Search free.inc in all \Here and suddirectories.

Procedure: FindModuleByAddrA

File:

\ObjAsm\Code\ObjMem\32\FindModuleByAddrA.asm
Find the module name from an address on a WinNT system. Purpose:

Arguments: Arg1: Address.

 $Arg2: \rightarrow ANSI module name buffer.$ 

eax = Number of characters copied into the buffer. Return:

Procedure: FindModuleByAddrW

\ObjAsm\Code\ObjMem\32\FindModuleByAddrw.asm File:

Purpose: Find the module name from an address on a WinNT system.

Arguments: Arg1: Address.

Arg2: → WIDE module name buffer.

eax = Number of characters copied into the buffer. Return:

Procedure: GetAncestorID

Files: \ObjAsm\Code\ObjMem\32\GetAncestorID.asm

\ObjAsm\Code\ObjMem\Common\GetAncestorID\_XP.inc

Retrieve the ancestor type ID of an object type ID. Purnose: Arguments: Arg1: → Object class ID.

Return: eax = Ancestor type ID or zero if not found.

Procedure: GetBottomWindow

\ObjAsm\Code\ObjMem\32\GetBottomWindow.asm Purpose: Get the Z order bottom child window HANDLE.

Arguments: Arg1: Parenat HWND.

Return: eax = Z order bottom child window HANDLE.

Procedure: GetDlgBaseUnits

\ObjAsm\Code\ObjMem\32\GetDlgBaseUnits.asm Files:

\ObjAsm\Code\ObjMem\Common\GetDlgBaseUnits\_X.inc

Purpose: Returns the Dialog Base Units.

Arguments: Arg1: Dialog DC. Return: eax = X DBU.ecx = Y DBU.

Procedure: GetExceptionStrA

\ObjAsm\Code\ObjMem\32\GetExceptionStrA.asm
Translate an exception code to an ANSI string. File: Purpose:

Arguments: Arg1: Exception code.  $eax \rightarrow ANSI string.$ Return:

Procedure: GetExceptionStrW

\ObjAsm\Code\ObjMem\32\GetExceptionStrW.asm Translate an exception code to a WIDE string. File: Purpose:

Arguments: Arg1: Exception code.  $eax \rightarrow WIDE$  string. Return:

Procedure: GetFileHashA

File:

\ObjAsm\Code\ObjMem\32\GetFileHashA.asm Compute the hash value from the content of a file. Purnose:

Arguments:  $Arg1: \rightarrow Hash return value$ Arg2: → ANSI file name.

Arg3: Hash type. eax = 0 if succeeded. Return:

http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297 Links:

Original translation from MSDN library by Edgar Hansen Notes:

It requires a fully qualified path to a file to generate a hash for and a pointer

to a WIDE string buffer to hold the resulting hash in HEX (16 BYTEs for MDx, 20 BYTEs

for SHAx) and an algorithm ID, for MD5 set dHashType to GFH\_MD5.

See ObjMem.inc GFH\_xxx.

Procedure: GetFileHashW

File:

\ObjAsm\Code\ObjMem\32\GetFileHashW.asm Compute the hash value from the content of a file. Purnose:

Arguments: Arg1: → Hash return value Arg2: → WIDE file name.

Arg3: Hash type.

eax = 0 if succeeded. Return:

http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297 Links:

Notes: Original translation from MSDN library by Edgar Hansen

It requires a fully qualified path to a file to generate a hash for and a pointer to a WIDE string buffer to hold the resulting hash in HEX (16 BYTES for MDx, 20 BYTES

for SHAx) and an algorithm ID, for MD5 set dHashType to GFH\_MD5.

See ObjMem.inc GFH\_xxx.

Procedure: GetFileLinesA

Files: \ObjAsm\Code\ObjMem\32\GetFileLinesA.asm

\\\\ObjAsm\\Code\\ObjMem\\Common\\GetFileLines\_AX.inc\\\\
Return an array of line ending offsets of an ANSI text file. Purpose:

Arguments: Arg1: File HANDLE.

Return:

eax = Number of lines.  $ecx \rightarrow Mem block containing an array of DWORD offsets.$ 

The user must dispose it using MemFree.

Notes: - Lines must be terminated with the ANSI char sequence 13, 10 (CRLF).

- The last line may not terminate with a CRLF.

Procedure: GetLogProcCount

\ObjAsm\Code\ObjMem\32\GetLogProcCount.asm File:

Return the number of logical CPUs on the current system. Purpose:

Arguments: None

Return: eax = Number of logical processors.

Procedure: GetObjectID

\ObjAsm\Code\ObjMem\32\GetObjectID.asm Retrieve the type ID of an object instance. Purpose:

Arguments: Arg1: → Object instance. Return: eax = Object class ID.

Procedure: GetObjectTemplate

Files:

\ObjAsm\Code\ObjMem\32\GetObjectTemplate.asm \ObjAsm\Code\ObjMem\Common\GetObjectTemplate\_XP.inc

Purpose: Get the template address of an object type ID.

Arguments: Arg1: Object type ID.

eax → Object template or NULL if not found. Return:

ecx = Object template size or zero if not found.

Procedure: GetPrevInstanceA

File: \ObjAsm\Code\ObjMem\32\GetPrevInstanceA.asm

Return a HANDLE to a previously running instance of an application. Purpose:

Arguments: Arg1:  $\rightarrow$  ANSI application name. Arg2:  $\rightarrow$  ANSI class name.

Return: eax = Window HANDLE of the application instance or zero if failed.

Procedure: GetPrevInstanceW

File: \ObjAsm\Code\ObjMem\32\GetPrevInstanceW.asm

Purpose: Return a HANDLE to a previously running instance of an application.

Arguments: Arg1: → WIDE application name. Arg2: → WIDE class name.

eax = Window HANDLE of the application instance or zero if failed. Return:

Procedure: GetRawClientRect

\ObjAsm\Code\ObjMem\32\GetRawClientRect.asm Files:

\ObjAsm\Code\ObjMem\Common\GetRawClientRect\_X.inc

Purpose: Calculate the window client RECT including scrollbars, but without the room needed

for the menubar.

Arguments: Arg1: Window HANDLE Arg2:  $\rightarrow$  RECT.

Nothing. Return:

Procedure: GUID2BStr

\ObjAsm\Code\ObjMem\32\GUID2BStr.asm File:

Convert a GUID to a BStr. Purpose:

Arguments: Arg1: → Destination BStr Buffer. It must hold at least

36 characters plus a ZTC.

Arg2:  $\rightarrow$  GUID.

Return: Nothing.

Procedure: GUID2StrA

Files:

\ObjAsm\Code\ObjMem\32\GUID2StrA.asm \ObjAsm\Code\ObjMem\Common\GUID2Str\_AXP.inc

Purpose: Convert a GUID to an ANSI string.

Arguments: Arg1: → Destination ANSI string buffer.

It must hold at least 36 characters plus a ZTC (= 37 BYTEs).

Arg2: → GUID.

Return: Nothing.

Procedure: GUID2StrW

Files: \ObjAsm\Code\ObjMem\32\GUID2Strw.asm

\ObjAsm\Code\ObjMem\Common\GUID2Str\_WXP.inc

Purpose: Convert a GUID to a WIDE string.

Arguments: Arg1:  $\rightarrow$  Destination WIDE string Buffer.

It must hold at least 36 characters plus a ZTC (= 74 BYTEs).

Arg2: → GUID.

Return: Nothing.

Procedure: hex2dwordA

File:

Purpose:

Arguments: Arg1: → ANSI hexadecimal string. Return: eax = DWORD.

Return:

Procedure: hex2dwordW

File: \ObjAsm\Code\ObjMem\32\hex2dwordW.asm

Purpose: Load a WIDE string hexadecimal representation of a DWORD.

Arguments: Arg1: → WIDE hex string.

eax = DWORD.Return:

Procedure: IsAdmin

File: \ObjAsm\Code\ObjMem\32\IsAdmin.asm

Purpose: Check if the current user has administrator rights.

Arguments: None.

eax = TRUE or FALSE.Return:

Procedure: IsGUIDEqual

File: \ObjAsm\Code\ObjMem\32\IsGUIDEqual.asm

Purpose: Compare 2 GUIDs. Arguments:  $Arg1: \rightarrow GUID1$ Arg2: → GUID2.

Return: eax = TRUE if they are equal, otherwise FALSE.

Procedure: IsHardwareFeaturePresent

\ObjAsm\Code\ObjMem\32\IsHardwareFeaturePresent.asm File: Purpose: Check if a CPU hardware feature is present on the system.

Notes: Check IHFP\_xxx equates in ObjMem.inc file.

Arguments: Arg1: CPUID feature ID. Return: eax = TRUE or FALSE.

Procedure: IsPntInRect

\ObjAsm\Code\ObjMem\32\IsPntInRect.asm Check if a point is within a rect. Purnose:

If rect.left = rect.right = 0, the point.x is considdered inside. Idem for y coord.

Arguments: Arg1:  $\rightarrow$  POINT.

ARg2: → RECT

eax = TRUE or FALSE. Return:

Procedure: IsProcessElevated

Files: \ObjAsm\Code\ObjMem\32\IsProcessElevated.asm

\ObjAsm\Code\ObjMem\Common\IsProcessElevated\_X.inc Check if the current process has elevated privileges.

Arguments: Arg: Process HANDLE.

Return: eax = TRUE or FALSE.

Example: invoke GetCurrentProcess invoke IsProcessElevated, xax

Procedure: IsScrollBarVisible

File: \ObjAsm\Code\ObjMem\32\IsscrollBarVisible.asm
Purpose: Determine if a Scrollbar is currently visible.
Arguments: Arg1: Main window handle that the scrollbar belongs to.

Arg2: Scrollbar type [SB\_HORZ or SB\_VERT].

Return: eax = TRUE if the scrollbar is visible, otherwise FALSE.

Procedure: IsWinNT

File: \ObjAsm\Code\ObjMem\32\IsWinNT.asm Purpose: Detect if the OS is Windows NT based.

Arguments: None.

Return: eax = TRUE if OS is Windows NT based, otherwise FALSE.

Procedure: LoadCommonControls

\ObjAsm\Code\ObjMem\32\LoadCommonControls.asm

Invoke InitCommonControls with a correctly filled input structure. Purpose:

Arguments: Arg1: ICC\_COOL\_CLASSES, ICC\_BAR\_CLASSES, ICC\_LISTVIEW\_CLASSES, ICC\_TAB\_CLASSES,

ICC\_USEREX\_CLASSES, etc.

Return: Nothing.

Procedure: Mem2HexA

\ObjAsm\Code\ObjMem\32\Mem2HexA.asm

Convert the memory content into a hex ANSI string representation. Purpose:

Arguments: Arg1: → ANSI character buffer. Arg2: → Source memory. Arg3: Byte count.

Return: Nothina.

Procedure: Mem2HexW

File: \ObjAsm\Code\ObjMem\32\Mem2HexW.asm

Purpose: Convert the memory content into a hex WIDE string representation.

Arguments: Arg1: → WIDE character buffer.

Arg2: → Source memory.

Arg3: Byte count.

Return: Nothing.

Procedure: MemAlloc\_UEFI

Files:

\ObjAsm\Code\ObjMem\32\MemAlloc\_UEFI.asm \ObjAsm\Code\ObjMem\Common\MemAlloc\_X\_UEFI.inc

Purpose: Allocate a memory block.

Arguments: Arg1: Memory block attributes [0, MEM\_INIT\_ZERO].
Arg2: Memory block size in BYTEs.

eax → Memory block or NULL if failed. Return:

Procedure: MemClone

\ObjAsm\Code\ObjMem\32\MemClone.asm File:

Purpose: Copy a memory block from a source to a destination buffer.

Source and destination must NOT overlap.

Destination buffer must be at least as large as number of BYTEs to copy, otherwise a

fault may be triggered. Arguments: Arg1:  $\rightarrow$  Destination buffer.

Arg2: → Source buffer. Arg3: Number of BYTEs to copy.

Return: Nothing.

Procedure: MemComp

\ObjAsm\Code\ObjMem\32\MemComp.asm

Compare 2 memory blocks. Purpose:

Both memory blocks must be at least as large as the maximal number of BYTEs to

Arguments:

compare, otherwise a fault may be triggered.

Arg1: → Memory block 1.

Arg2: → Memory block 2.

Arg3: Maximal number of BYTEs to compare.

If MemBlock1 = MemBlock2, then eax <> 0. If MemBlock1 = MemBlock2, then eax = 0.

Procedure: MemFillB

Return:

\ObjAsm\Code\ObjMem\32\MemFillB.asm

Purpose: Fill a memory block with a given byte value.

Destination buffer must be at least as large as number of BYTEs to fill, otherwise a

fault may be triggered.

Arguments: Arg1: → Destination memory block.

Arg2: Memory block size in BYTEs.

Arg3: Byte value to fill.

Nothing. Return:

Procedure: MemFillW

\ObjAsm\Code\ObjMem\32\MemFillW.asm File:

Fill a memory block with a given word value. Purpose:

Destination buffer must be at least as large as number of BYTEs to fill, otherwise a

fault may be triggered.

Arguments: Arg1: → Destination memory block.

Arg2: Memory block size in BYTEs. Arg3: Word value to fill with.

Return: Nothing.

Procedure: MemFree\_UEFI

\ObjAsm\Code\ObjMem\32\MemFree\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\MemFree\_X\_UEFI.inc

Purpose: Dispose a memory block. Arguments: Arg1: → Memory block.

eax = EFI\_SUCCESS or an UEFI error code. Return:

Procedure: MemReAlloc\_UEFI

\ObjAsm\Code\ObjMem\32\MemReAlloc\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\MemReAlloc\_X\_UEFI.inc

Shrink or expand a memory block. Purpose:

Arguments: Arg1: → Memory block

Arg2: Memory block size in BYTEs.

Arg3: New memory block size in BYTEs.

Arg4: Memory block attributes [0, MEM\_INIT\_ZERO]. eax  $\rightarrow$  New memory block.

Return:

Procedure: MemShift

File: \ObjAsm\Code\ObjMem\32\MemShift.asm

Copy a memory block from a source to a destination buffer. Purpose:

Source and destination may overlap.

Destination buffer must be at least as large as number of BYTEs to shift, otherwise a

fault may be triggered.

Arguments: Arg1: → Destination buffer.
Arg2: → Source buffer.
Arg3: Number of BYTEs to shift. eax = Number of BYTEs shifted. Return:

Procedure: MemSwap

File: \ObjAsm\Code\ObjMem\32\MemSwap.asm

Purpose: Exchange the memory content from a memory buffer to another.

They must NOT overlap.

Both buffers must be at least as large as number of BYTEs to exchange, otherwise a

fault may be triggered. Arguments:  $Arg1: \rightarrow Memory \ buffer \ 1.$   $Arg2: \rightarrow Memory \ buffer \ 2.$   $Arg3: \ Number \ of \ BYTEs \ to \ exchange.$ 

Return: Nothing.

Procedure: MemZero

\ObjAsm\Code\ObjMem\32\MemZero.asm File:

Fill a memory block with zeros. A bit faster than MemFillB. Purpose:

The memory buffer must be at least as large as number of BYTEs to zero, otherwise a

fault may be triggered.

Arguments: Arg1: → Memory buffer.
Arg2: Number of BYTEs to zero.

Return: Nothing.

Procedure: MoveWindowVisible

\ObjAsm\Code\ObjMem\32\MoveWindowVisible.asm File:

On a multimonitor system, move a window but remain always in the visible region. Purpose:

Arguments: Arg1: HANDLE of the Window to move.

Arg2: Target X position in pixel. Arg3: Target Y position in pixel.

Return: Nothing. Procedure:

\ObjAsm\Code\ObjMem\32\MsqBoxA.asm

\ObjAsm\Code\ObjMem\Common\MsgBox\_TX.inc

Purpose: Show a customized MessageBox.

Arguments: Arg1: Parent HANDLE. Arg2: → Markup text. Arg3: → Caption text.

Arg4: Flags.

eax = Zero if failed, otherwise pressed button ID. Return:

Note: Caption, text etc. are transferred via a caption string which contains a header and

the address of a MsgBoxInfo structure in text form.

Procedure:

\ObjAsm\Code\ObjMem\32\MsgBoxW.asm

\ObjAsm\Code\ObjMem\Common\MsgBox\_TX.inc

Purpose: Show a customized MessageBox.

Arg1: Parent HANDLE. Arg2: → Markup text. Arg3: → Caption text. Arguments:

Arg4: Flags.

eax = Zero if failed, otherwise pressed button ID. Return:

Caption, text etc. are transferred via a caption string which contains a header and Note:

the address of a MsgBoxInfo structure in text form.

Procedure: NetErr2StrA

\ObjAsm\Code\ObjMem\32\NetErr2StrA.asm

Purpose: Translate a network error code to an ANSI string.

Arguments: Arg1: Error code.

Arg2: → ANSI character buffer. Arg3: Buffer size in characters, inclusive ZTC.

Return: Nothing.

Procedure: NetErr2StrW

\ObjAsm\Code\ObjMem\32\NetErr2StrW.asm File:

Purpose: Translate a network error code to a WIDE string.

Arguments: Arg1: Error code.

Arg2: → WIDE string buffer.

Arg3: Buffer size in characters, inclusive ZTC.

Nothing. Return:

Procedure: NewObjInst

\ObjAsm\Code\ObjMem\32\NewObjInst.asm Files:

\ObjAsm\Code\ObjMem\Common\NewObjInst\_XP.inc Purpose: Create an object instance from an object ID.

Arguments: Arg1: Object ID.

Return:  $eax \rightarrow New$  object instance or NULL if failed.

Procedure: NewObjInst\_UEFI

\ObjAsm\Code\ObjMem\32\NewObjInst\_UEFI.asm \ObjAsm\Code\ObjMem\Common\NewObjInst\_XP.inc

Purpose: Create an object instance from an object ID.

Arguments: Arg1: Object ID.

eax → New object instance or NULL if failed. Return:

Procedure: ParseA

\ObjAsm\Code\ObjMem\32\ParseA.asm File:

Purpose: Extract a comma separated substring from a source string.

 $Arg1: \rightarrow Destination buffer.$  Must be large enough to hold the ANSI substring. Arguments:

Arg2: → Source ANSI string.

Arg3: Zero based index of the requested ANSI substring.

eax = 1: success. Return:

2: insufficient number of components. 3: non matching quotation marks.

4: empty quote.

Procedure: ParseW

\ObjAsm\Code\ObjMem\32\ParseW.asm File:

Extract a comma separated substring from a source string. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination buffer. Must be large enough to hold the WIDE substring.

Arg2:  $\rightarrow$  Source WIDE string. Arg3: Zero based index of the requested WIDE substring.

eax = 1: success. Return:

2: insufficient number of components.

3: non matching quotation marks.

4: empty quote.

Procedure: PdfViewA

\ObjAsm\Code\ObjMem\32\PdfViewA.asm Files:

\ObjAsm\Code\ObjMem\Common\PdfView\_TX.inc Display a PDF document on a named destination. Purnose:

Arguments: Arg1: Parent HANDLE.

Arg2: → PDF document. Arg3: → Destination.

eax = HINSTANCE. See ShellExecute return values. Return:

A value greater than 32 indicates success.

Procedure: PdfViewW

\ObjAsm\Code\ObjMem\32\PdfViewW.asm Files:

\ObjAsm\Code\ObjMem\Common\PdfView\_TX.inc

Display a PDF document on a named destination. Purpose:

Arguments: Arg1: Parent HANDLE. Arg2: → PDF document. Arg3: → Destination.

eax = HINSTANCE. See ShellExecute return values. Return:

A value greater than 32 indicates success.

Procedure: qword2binA

File: \ObjAsm\Code\ObjMem\32\qword2binA.asm

Purpose: Convert a QWORD to its binary ANSI string representation.

Arguments: Arg1: → Destination buffer. Arg2: QWORD value.

Nothing. Return:

The destination buffer must be at least 65 BYTEs large to allocate the output string Note:

(64 character BYTES + ZTC = 65 BYTES).

Procedure: qword2binW

File:

\ObjAsm\Code\ObjMem\32\qword2binW.asm Convert a QWORD to its binary WIDE string representation. Purpose:

Arguments: Arg1: → Destination buffer. Arg2: QWORD value.

Nothing. Return:

The destination buffer must be at least 130 BYTEs large to allocate the output string Note:

(64 character WORDS + ZTC = 130 BYTES).

Procedure: qword2hexA

File: \ObjAsm\Code\ObjMem\32\qword2hexA.asm

Purpose: Convert a QWORD to its hexadecimal ANSI string representation.

Arg1: → Destination buffer. Arg2: QWORD value. Arguments:

Return: Nothing.

The destination buffer must be at least 17 BYTEs large to allocate the output string Note:

(16 character BYTEs + ZTC = 17 BYTEs).

Procedure: qword2hexW

File: \ObjAsm\Code\ObjMem\32\gword2hexw.asm

Convert a QWORD to its hexadecimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination buffer.
Arg2: QWORD value.

Nothing. Return:

Note: The destination buffer must be at least 34 BYTEs large to allocate the output string

(16 character WORDS + ZTC = 34 BYTES).

Procedure: RadixSortF32

\ObjAsm\Code\ObjMem\32\RadixSortF32.asm Files: \ObjAsm\Code\ObjMem\Common\RadixSort32.inc

Ascending sort of an array of single precision floats (REAL4) using a modified "4 passes radix sort" algorithm. Purpose:

Arg1: → Array of single precision floats. Arguments:

Arg2: Number of single precision floats contained in the array.

Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

eax = TRUE if succeeded, otherwise FALSE. Return:

- For short arrays, the shadow array can be placed onto the stack, saving the Notes:

expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included.

Links: http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortF64

\ObjAsm\Code\ObjMem\32\RadixSortF64.asm Files:

\ObjAsm\Code\ObjMem\Common\RadixSort32.inc

Ascending sort of an array of double precision floats (REAL8) using a modified "8 passes radix sort" algorithm. Purnose:

Arguments:

Arg1: → Array of double precision floats (REAL8).
Arg2: Number of double precision floats contained in the array.

Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

Return:

eax = TRUE if succeeded, otherwise FALSE.
- For short arrays, the shadow array can be placed onto the stack, saving the
expensive memory allocation/deallocation API calls. To achieve this, the proc Notes:

must be modified and stack probing must be included. - http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortI32

Links:

Links:

Links:

Links:

\ObjAsm\Code\ObjMem\32\RadixSortI32.asm \ObjAsm\Code\ObjMem\Common\RadixSort32.inc Files:

Ascending sort of an array of SDWORDs using a modified "4 passes radix sort" Purpose:

algorithm.

 $Arg1: \rightarrow Array of SDWORDs.$ Arguments:

Arg2: Number of SDWORDs contained in the array.

Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least

the size of the input array. If NULL, a memory chunk is allocated automatically. eax = TRUE if succeeded, otherwise FALSE.

Return:

Original code from r22. Notes:

http://www.asmcommunity.net/board/index.php?topic=24563.0 - For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc

must be modified and stack probing must be included.
- http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrF32

Files: \ObjAsm\Code\ObjMem\32\RadixSortPtrF32.asm \ObjAsm\Code\ObjMem\Common\RadixSort32.inc

Purpose:

Ascending sort of an array of POINTERS to structures containing a single precision float (REAL4) key using a modified "4 passes radix sort" algorithm.

 $Arg1: \rightarrow Array of POINTERs.$ Arguments:

Arg2: Number of POINTERs contained in the array. Arg3: offset of the REAL4 key within the hosting structure.

Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

eax = TRUE if succeeded, otherwise FALSE. Return:

- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included. Notes:

- http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrF64

Files: \ObjAsm\Code\ObjMem\32\RadixSortPtrF64.asm

Purnose:

\langle ObjAsm\Code\ObjMem\Common\RadixSort32.inc Ascending sort of an array of POINTERs to structures containing a double precision float (REAL8) key using a modified "8 passes radix sort" algorithm.

Arguments:

Arg1: → Array of POINTERS.

Arg2: Number of POINTERS contained in the array.

Arg3: offset of the REAL8 key within the hosting structure. Arg4:  $\rightarrow$  Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

Return: eax = TRUE if succeeded, otherwise FALSE.

- For short arrays, the shadow array can be placed onto the stack, saving the Notes:

expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included.

http://www.codercorner.com/RadixSortRevisited.htmhttp://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrI32

\ObjAsm\Code\ObjMem\32\RadixSortPtrI32.asm Files: \ObjAsm\Code\ObjMem\Common\RadixSort32.inc

Ascending sort of an array of POINTERs to structures containing a SDWORD key using a modified "4 passes radix sort" algorithm. Purpose:

Arg1:  $\rightarrow$  Array of POINTERs. Arguments:

Arg2: Number of POINTERs contained in the array.

Arg3: offset of the SDWORD key within the hosting structure.

 $Arg4: \rightarrow Memory$  used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

eax = TRUE if succeeded, otherwise FALSE.

- Original code from r22. Return:

Notes:

http://www.asmcommunity.net/board/index.php?topic=24563.0
- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included.

- http://www.codercorner.com/RadixSortRevisited.htm - http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrUI32

Links:

Links:

Tinks:

Files: \ObjAsm\Code\ObjMem\32\RadixSortPtrUI32.asm \ObjAsm\Code\ObjMem\Common\RadixSort32.inc

Ascending sort of a POINTER array to structures containing a DWORD key using the "4 passes radix sort" algorithm. Purpose:

Arguments:

Arg1: → Array of POINTERS.

Arg2: Number of POINTERS contained in the array.

Arg3: offset of the DWORD key within the hosting structure.

Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

eax = TRUE if succeeded, otherwise FALSE. Return:

Notes: - Original code from r22.

http://www.asmcommunity.net/board/index.php?topic=24563.0

- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc

must be modified and stack probing must be included. - http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortUI32

Files: \ObjAsm\Code\ObjMem\32\RadixSortUI32.asm

\langle ObjAsm\Code\ObjMem\Common\RadixSort32.inc Ascending sort of an array of DWORDs using the "4 passes radix sort" algorithm. Purnose:

 $Arg1: \rightarrow Array of DWORDs.$ Arguments:

Arg2: Number of DWORDs contained in the array.

Arg3:  $\rightarrow$  Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically. eax = TRUE if succeeded, otherwise FALSE.

Return:

- Original code from r22. Notes:

http://www.asmcommunity.net/board/index.php?topic=24563.0 - For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc

must be modified and stack probing must be included. - http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: Random32

Files: \ObjAsm\Code\ObjMem\32\Random32.asm

Purpose:

\ObjAsm\Code\ObjMem\Common\Random32 X.inc
Generate a random 32 bit number in a given range [0..Limit-1].
Park Miller random number algorithm. Written by Jaymeson Trudgen (NaN) and optimized

by Rickey Bowers Jr. (bitRAKE).

Arguments: Arg1: Range limit (max. = 07FFFFFFFh).
Return: eax = Random number in the range [0..Limit-1].

Procedure: Real4ToHalf

\ObjAsm\Code\ObjMem\32\Real4ToHalf.asm Files:

<u>\ObjAsm\Code\ObjMem\Common\Real4ToHalf\_XP.inc</u>

Purpose: Convert a REAL4 to an HALF.

Arguments: Arg1: REAL4 value. Return: ax = HALF.

alternative code using VCVTPS2PH: Note:

movss xmm0, r4value VCVTPS2PH xmm1, xmm0, 0

movd eax, xmm1

Procedure: RGB24To16ColorIndex

\ObjAsm\Code\ObjMem\32\RGB24To16ColorIndex.asm Files:

\ObjAsm\Code\ObjMem\Common\RGB24To16ColorIndex\_XP.inc Purpose: Map a 24 bit RGB color to a 16 color palette index.

Arguments: Arg1: RGB color. eax = Palette index.Return:

Procedure: sdword2decA

\ObjAsm\Code\ObjMem\32\sdword2decA.asm File:

Purpose: Convert a signed DWORD to its decimal ANSI string representation.

Arguments: Arg1: → Destination ANSI string buffer.

Arg2: SDWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 12 BYTES large to allocate the output string (Sign + 10 ANSI characters + ZTC = 12 BYTES). Note:

Procedure: sdword2decW

File:

\ObjAsm\Code\ObjMem\32\sdword2decW.asm
Convert a signed DWORD to its decimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination WIDE string buffer.

Arg2: SDWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 24 BYTEs large to allocate the output string Note:

(Sign + 10 WIDE characters + ZTC = 24 BYTES).

Procedure: SendChildrenMessage

Files: \ObjAsm\Code\ObjMem\32\SendChildrenMessage.asm

\ObjAsm\Code\ObjMem\Common\SendChildrenMessage\_X.inc

Purpose: Callback procedure for EnumChildWindows that sends a message to a child window.

Arguments: Arg1: Child window HANDLE.

Arg2: → CHILD\_MSG structure. Return: eax = Always TRUE (continue the enumeration).

Procedure: SetClientSize

Files: \ObjAsm\Code\ObjMem\32\SetClientSize.asm

\ObjAsm\Code\ObjMem\Common\SetClientSize\_X.inc

Set the client window size. Purpose: Arguments: Arg1: Target window handle.
Arg2: Client area width in pixel.

Arg3: Client area height in pixel.

Return: Nothina.

Procedure: SetExceptionMessageA

File: \ObjAsm\Code\ObjMem\32\SetExceptionMessageA.asm

Install a final exception handler that displays a messagebox showing detailed Purpose:

exception

information and a user text.

Arg1: → User ANSI message string. Arg2: → Messagebox ANSI title string. Arguments:

Arg3: → Callback procedure fired when an exception reaches the final handler. If the callback returns zero, the messagebox is displayed, otherwise

EXCEPTION\_EXECUTE\_HANDLER is passed to the OS without showing the messagebox.

If this parameter is NULL, the messgebox is always displayed.

Return: Nothing.

Procedure: SetExceptionMessageW

\ObjAsm\Code\ObjMem\32\SetExceptionMessageW.asm

Install a final exception handler that displays a messagebox showing detailed Purpose:

exception

information and a user text. Arguments:

Arg1: → User wide message string.

Arg2: → Messagebox WIDE title string.

Arg3: → Callback procedure fired when an exception reaches the final handler.

If the callback returns zero, the messagebox is displayed, otherwise

EXCEPTION\_EXECUTE\_HANDLER is passed to the OS without showing the messagebox.

If this parameter is NULL, the messgebox is always displayed.

Return: Nothing.

Procedure: SetPrivilegeTokenA

\ObjAsm\Code\ObjMem\32\SetPrivilegeTokenA.asm

Enable privilege tokens. Purnose: Arguments: Arg1: Process handle.
Arg2: → Privilege name (ANSI string).

Arg3: Eanble = TRUE, disable = FALSE eax = Zero if failed.

Return:

Procedure: SetPrivilegeTokenW

\ObjAsm\Code\ObjMem\32\SetPrivilegeTokenw.asm File:

Purpose: Enable privilege tokens. Arguments: Arg1: Process handle.

Arg2: → Privilege name (ANSI string).

Arg3: Eanble = TRUE, disable = FALSE

Return: eax = Zero if failed.

Procedure: SetShellAssociationA

\ObjAsm\Code\ObjMem\32\SetShellAssociationA.asm Files:

\ObjAsm\Code\ObjMem\Common\SetShellAssociation\_TX.inc

Purpose:

Set association for a file extension.

Arg1: TRUE = system wide association, FALSE = user account only. Arguments:

Arg2: → File extension (without dot).

Arg3: → Verb ("open", "print", "play", "edit", etc.). This verb is displayed in the explorer context menu of a file with this extension.

Arg4: → Application to associate with (full path)  $Arg5: \rightarrow Application arguments, usually $0fsCStr("%1").$ 

Return: eax = HRESULT.

dGlobal = TRUE requires adminitrative rights. Note:

Procedure: SetShellAssociationW

\ObjAsm\Code\ObjMem\32\SetShellAssociationW.asm Files:

\ObjAsm\Code\ObjMem\Common\SetShellAssociation\_TX.inc Set association for a file extension.

Purnose:

Arguments: Arg1: TRUE = system wide association, FALSE = user account only.

 $Arg2: \rightarrow File$  extension (without dot).  $Arg3: \rightarrow Verb$  ("open", "print", "play", "edit", etc.). This verb is displayed in the explorer context menu of a file with this extension.

Arg4: → Application to associate with (full path). Arg5: → Application arguments, usually \$ofscstr("%1").

eax = HRESULT.Return:

dGlobal = TRUE requires adminitrative rights. Note:

Procedure: SetShellPerceivedTypeA

Files:

\ObjAsm\Code\objMem\32\SetShellPerceivedTypeA.asm \ObjAsm\Code\objMem\Common\SetShellPerceivedType\_TX.inc Set shell perception of a file type.

Purnose:

Arg1: TRUE = system wide perseption, FALSE = user account only. Arguments:

Arg2: → File extension (without dot).
Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,

Application, Gamemedia, Contacts)

Return: eax = HRESULT.

To retrieve the perceived type use the AssocGetPerceivedType API. Note:

dGlobal = TRUE requires adminitrative rights.

SetShellPerceivedTypeW Procedure:

Files:

\ObjAsm\Code\ObjMem\32\SetShellPerceivedTypeW.asm \ObjAsm\Code\ObjMem\Common\SetShellPerceivedType\_TX.inc

Set shell perception of a file type. Purpose:

Arguments: Arg1: TRUE = system wide perseption, FALSE = user account only.

Arg2: → File extension (without dot). Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,

Application, Gamemedia, Contacts)

Return: eax = HRESULT.

To retrieve the perceived type use the AssocGetPerceivedType API. Note:

dGlobal = TRUE requires adminitrative rights.

Procedure: ShortToLongPathNameA

File:

\ObjAsm\Code\ObjMem\32\ShortToLongPathNameA.asm
Allocate a new ANSI string containing the long path of a short path string. Purpose:

Arguments: Arg1: → Short path ANSI string.

Return: eax → Long path ANSI string or NULL if failed.

Procedure: SLR\_Calc\_AB\_DW

\ObjAsm\Code\ObjMem\32\SLR\_Calc\_AB\_DW.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_XP.inc Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + B that minimize mean squared error (MSE) of a DWORD array. Purpose:

Arguments: Arg1: → SLR\_DATA structure.

eax = TRUE is succeeded, otherwise FALSE. Return:

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_KE2%8B%AFhttps://mathschallenge.net/library/number/sum\_of\_squares Links:

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas:

B = (Y-A\*X)/N

Procedure: SLR\_Calc\_AB\_MSE\_DW

\ObjAsm\Code\ObjMem\32\SLR\_Calc\_AB\_MSE\_DW.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_MSE\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + B that minimize mean squared error (MSE) and the MSE value of a DWORD array. Purpose:

Arguments: Arg1: → SLR\_DATA structure.

eax = TRUE is succeeded, otherwise FALSE. Return:

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_KE2%8B%AFhttps://mathschallenge.net/library/number/sum\_of\_squares links:

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)QFormulas:

B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_MSE\_QW

\ObjAsm\Code\ObjMem\32\SLR\_Calc\_AB\_MSE\_OW.asm \ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_MSE\_XP.inc Files:

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + BPurpose:

that minimize mean squared error (MSE) and the MSE value of a QWORD array.

Arguments: Arg1: → SLR\_DATA structure.

Return: eax = TRUE is succeeded, otherwise FALSE.

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Links:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Since X ranges from [0. N-1], the known formulas have to be adjusted accordingly by Note:

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

 $A = (XY*N-X*Y)\dot{/}Q$ Formulas:

B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_MSE\_R4

\ObjAsm\Code\ObjMem\32\SLR\_Calc\_AB\_MSE\_R4.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_MSE\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + BPurpose:

that minimize mean squared error (MSE) and the MSE value of a REAL4 array.

Arg1: → SLR\_DATA structure. Arguments:

Return:

eax = TRUE is succeeded, otherwise FALSE. https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Tinks:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas:

B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_MSE\_R8

Files:

Purpose:

that minimize mean squared error (MSE) and the MSE value of a REAL8 array.

Arguments: Arg1: → SLR\_DATA structure

Return: eax = TRUE is succeeded, otherwise FALSE.

Links: https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas:

B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_QW

\ObjAsm\Code\ObjMem\32\SLR\_Calc\_AB\_QW.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + BPurpose:

that minimize mean squared error (MSE) of a QWORD array.

Arg1: → SLR\_DATA structure. Arguments:

eax = TRUE is succeeded, otherwise FALSE. Return:

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Links:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by Note:

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)'/QFormulas: B = (Y-A\*X)/N

Procedure: SLR\_Calc\_AB\_R4

\ObjAsm\Code\ObjMem\32\SLR\_Calc\_AB\_R4.asm Files: \ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + B that minimize mean squared error (MSE) of a REAL4 array. Purpose:

Arguments:

Arg1: → SLR\_DATA structure. eax = TRUE is succeeded, otherwise FALSE. Return:

 $https://en.wikipedia.org/wiki/1\_\%2B\_2\_\%2B\_3\_\%2B\_4\_\%2B\_\%E2\%8B\%AF$ Tinks:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas: B = (Y-A\*X)/N

Procedure: SLR\_Calc\_AB\_R8

Files:

Purnose:

that minimize mean squared error (MSE) of a REAL8 array.

Arguments: Arg1: → SLR\_DATA structure

Return: eax = TRUE is succeeded, otherwise FALSE.

Links: https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-linec7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

Formulas: A = (XY\*N-X\*Y)/Q

B = (Y-A\*X)/N

Procedure: SLR\_Init

\ObjAsm\Code\ObjMem\32\SLR\_Init.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Init\_XP.inc

Calculate in advance the invariant coefficients of a Simple Linear Regression (X, X2, Purpose:

Arguments: Arg1: → SLR\_DATA structure.

Return:

eax = TRUE is succeeded, otherwise FALSE.
https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Links:

https://mathschallenge.net/library/number/sum\_of\_squares https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

Formulas: X = N\*(N-1)/2

X2 = X\*(2\*N - 1)/3 $Q = N^2*(N^2-1)/12$ 

Procedure: sqword2decA

Files: \ObjAsm\Code\ObjMem\32\sqword2decA.asm

\ObjAsm\Code\ObjMem\Common\sqword2decT32.inc

Convert a signed QWORD to its decimal ANSI string representation.

Arg1: → Destination ANSI string buffer.

Arg2: SQWORD value. Purpose:

Arguments:

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

Note: The destination buffer must be at least 21 BYTEs large to allocate the output string

(Sign + 19 ANSI characters + ZTC = 21 BYTEs).

Procedure: sqword2decW

Files: \ObjAsm\Code\ObjMem\32\sqword2decW.asm

\ObjAsm\Code\ObjMem\Common\sqword2decT32.inc

Convert a signed SQWORD to its decimal WIDE string representation.

Arg1: → Destination WIDE string buffer. Arguments:

Arg2: SQWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 42 BYTEs large to allocate the output string (Sign + 19 WIDE characters + ZTC = 42 BYTEs). Note:

Procedure: StOToStrA

\ObjAsm\Code\ObjMem\32\StOTOStrA.asm \ObjAsm\Code\ObjMem\Common\StOTOStr\_AXP.inc Files:

Create an ANSI string representation of the content of the st(0) FPU register. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: Minimal number of places from the start of string up to the decimal point.

(f\_NOR only)

Arg3: Number of decimal places after the decimal point.

Return:

Notes:

Arg3: Number of decimal places after the decimal point.

Arg4: Format flag (f\_NOR, f\_SCI, f\_TRIM, f\_ALIGNED) defined in fMath.inc eax = Result code f\_OK, f\_ERROR, f\_NAN, ...

- Based on the work of Raymond Filiatreault (FpuLib).

- st4, st5, st6 and st7 must be empty.

- f\_NOR: regular output format

- f\_SCI: Scientific output format - f\_TRIM: Trim zeros on the right

- f\_ALIGN: Add a heading space to align the output with other negative numbers

- f\_PLUS: like f\_ALIGN, but using a + character.

Procedure: StOToStrW

\ObjAsm\Code\ObjMem\32\StOToStrW.asm Files:

\ObjAsm\Code\ObjMem\Common\StOToStr\_wXP.inc

Create a WIDE string representation of the content of the st(0) FPU register. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: Minimal number of places from the start of string up to the decimal point.

(f NOR only)

Arg3: Number of decimal places after the decimal point.

Arg4: Format flag (f\_NOR, f\_SCI, f\_TRIM, f\_ALIGNED) defined in fMath.inc eax = Result code f\_OK, f\_ERROR, f\_NAN, ...

- Based on the work of Raymond Filiatreault (FpuLib).

Return:

Notes:

- st4, st5, st6 and st7 must be empty. - f\_NOR: regular output format - f\_SCI: Scientific output format - f\_TRIM: Trim zeros on the right

- f\_ALIGN: Add a heading space to align the output with other negative numbers - f\_PLUS: like f\_ALIGN, but using a + character.

Procedure: StkGrdCallback

\ObjAsm\Code\ObjMem\32\StkGrdCallback.asm File: Purpose: StackGuard notification callback procedure.

It is called when StackGuard is active and a stack overrun was detected. It displays a MessageBox asking to abort. If yes, then Exitprocess is called

immediately.

Arguments:

Returns: ZERO flag set if NO was pressed

Procedure: Str2BStrA

File: \( \lambda \text{ObjAsm\Code\ObjMem\32\Str2BStrA.asm} \)

Purpose: Convert a ANSI string into a BStr.

Arguments: Arg1: \( \rightarrow \text{Destination BStr buffer} = \text{Buffer address} + \text{sizeof DWORD.} \)

Arg2: \( \rightarrow \text{Source ANSI string.} \)

Return: Nothing.

Procedure: Str2BStrW

\ObjAsm\Code\ObjMem\32\Str2BStrW.asm Convert a ANSI string into a BStr.

Arguments: Arg1: → Destination BStr buffer = Buffer address + sizeof DWORD.

Arg2: → Source WIDE string.

Return: Nothing.

Procedure: StrA2StrW

\ObjAsm\Code\ObjMem\32\StrA2StrW.asm File: Purpose: Convert a ANSI string into a WIDE string. Arguments: Arg1:  $\rightarrow$  Destination WIDE string buffer. Arg2:  $\rightarrow$  Source ANSI string.

Nothing. Return:

Procedure: StrAllocA

\ObjAsm\Code\ObjMem\32\StrAllocA.asm

Purpose: Allocate space for an ANSI string with n characters. Arguments: Arg1: Character count without the ZTC. Return:  $eax \rightarrow New$  allocated ANSI string or NULL if failed.

Procedure: StrAllocA\_UEFI

\ObjAsm\Code\ObjMem\32\StrAllocA\_UEFI.asm \ObjAsm\Code\ObjMem\Common\StrAlloc\_TX\_UEFI.inc Files:

Purnose: Allocate space for a string with n characters.

Arguments: Arg1: Character count without the ZTC.

 $eax \rightarrow New$  allocated string or NULL if failed. Return:

Procedure: StrAllocW

\ObiAsm\Code\ObiMem\32\StrAllocw.asm

Allocate space for a WIDE string with n characters. Purpose:

Arguments: Arg1: Character count without the ZTC.

Return:  $eax \rightarrow New$  allocated WIDE string or NULL if failed.

Procedure: StrAllocW\_UEFI

\ObjAsm\Code\ObjMem\32\StrAllocW\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\StrAlloc\_TX\_UEFI.inc Purpose: Allocate space for a string with n characters.

Arguments: Arg1: Character count without the ZTC.

 $eax \rightarrow New$  allocated string or NULL if failed.

Procedure: StrCatA

File: \ObjAsm\Code\ObjMem\32\StrCatA.asm Purpose: Concatenate 2 ANSI strings.
Arguments: Arg1: Destrination ANSI buffer.

Arg2: Source ANSI string. eax = Number of added BYTEs. Return:

Procedure: StrCatCharA

\ObjAsm\Code\ObjMem\32\StrCatCharA.asm File:

Purpose: Append a character to the end of an ANSI string.

Arguments: Arg1: Destrination ANSI buffer. Arg2: ANSI character.

Return: Nothing.

Procedure: StrCatCharW

File:

\ObjAsm\Code\ObjMem\32\StrCatCharw.asm
Append a character to the end of a WIDE string. Purpose:

Arguments: Arg1: Destrination WIDE buffer. Arg2: WIDE character.

Return: Nothing.

Procedure: StrCatW

\ObjAsm\Code\ObjMem\32\StrCatW.asm Purpose: Concatenate 2 WIDE strings. Arguments: Arg1: Destrination WIDE string.
Arg2: Source WIDE string.

Return: Nothing.

Procedure: StrCCatA

\ObjAsm\Code\ObjMem\32\StrCCatA.asm File:

Concatenate 2 ANSI strings with length limitation. Purpose:

The destination string buffer should have at least enough room for the maximum number

of characters + 1.

Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string. Arguments:

Arg3: Maximal number of charachters that the destination string can hold including the

eax = Number of added BYTEs. Return:

Procedure: StrCCatCharA

File: \ObjAsm\Code\ObjMem\32\StrCCatCharA.asm

Purpose: Append a character to the end of an ANSI string with length limitation.

Arguments: Arg1: → Destination ANSI character buffer.

Arg2: → ANSI character.

Arg3: Maximal number of characters that fit into the destination buffer.

Return: Nothing.

Procedure: StrCCatCharW

\ObjAsm\Code\ObjMem\32\StrCCatCharw.asm

Append a character to the end of a WIDE string with length limitation. Purpose:

Arg1: → Destination WIDE character buffer. Arguments:

Arg2: → Wide character.

Arg3: Maximal number of characters that fit into the destination buffer.

Return: Nothina.

Procedure: StrCCatW

File: \ObjAsm\Code\ObjMem\32\StrCCatW.asm

Purpose:

Concatenate 2 WIDE strings with length limitation. The destination string buffer should have at least enough room for the maximum number

of characters + 1.

Arguments:

Arg1: → Destination WIDE character buffer.
Arg2: → Source WIDE string.
Arg3: Maximal number of charachters that the destination string can hold including the

ZTC.

eax = Number of added BYTEs. Return:

Procedure: StrCCompA

\ObjAsm\Code\ObjMem\32\StrCCompA.asm

Purpose: Compare 2 ANSI strings with case sensitivity up to a maximal number of characters.

Arguments:  $Arg1: \rightarrow ANSI string 1.$ Arg2: → ANSI string 2.

Arg3: Maximal number of characters to compare.

Return:

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0.

Procedure: StrCCompW

\ObjAsm\Code\ObjMem\32\StrCCompW.asm File:

Compare 2 WIDE strings with case sensitivity up to a maximal number of characters. Purpose:

Arg1:  $\rightarrow$  WIDE string 1. Arguments:

 $Arg2: \rightarrow WIDE string 2.$ 

Arg3: Maximal number of characters to compare.

Return:

If string 1 < string 2, then eax < 0.
If string 1 = string 2, then eax = 0.
If string 1 > string 2, then eax > 0.

Procedure: StrCCopyA

\ObjAsm\Code\ObjMem\32\StrCCopyA.asm File:

Copy the the source ANSI string  $\overline{\text{with}}$  length limitation. Purpose:

The destination buffer should be big enough to hold the maximum number of

characters + 1.

Arguments: Arg1: → Destination buffer.

Arg2: → Source ANSI string.

Arg3: Maximal number of charachters to copy, excluding the ZTC.

eax = Number of copied BYTEs, including the ZTC. Return:

Procedure: StrCCopyW

File:

Purnose:

\ObjAsm\Code\ObjMem\32\StrCCopyW.asm
Copy the the source WIDE string with length limitation.
The destination buffer should be big enough to hold the maximum number of

characters + 1.

Arguments: Arg1: → Destination buffer.

Arg2: → Source WIDE string.

Arg3: Maximal number of charachters to copy, excluding the ZTC.

eax = Number of copied BYTEs, including the ZTC. Return:

Procedure: **StrCECatA** 

File:

\ObjAsm\Code\ObjMem\32\StrCECatA.asm
Concatenate 2 ANSI strings with length limitation and return the ending zero character Purpose:

address. The destination string buffer should have at least enough room for the

maximum

number of characters + 1.

 $Arg1: \rightarrow Destination ANSI character buffer.$ Arguments:

Arg2: → Source ANSI string.

Arg̃3: Maximal number of charachters that the destination string can hold including the

ZTC.

eax  $\rightarrow$  ZTC. Return:

Procedure: StrCECatW

\ObjAsm\Code\ObjMem\32\StrCECatW.asm File:

Concatenate 2 WIDE strings with length limitation and return the ending zero character Purpose:

address. The destination string buffer should have at least enough room for the

maximum number of characters + 1.

Arg1:  $\rightarrow$  Destination WIDE character buffer. Arg2:  $\rightarrow$  Source WIDE string. Arguments:

Arg3: Maximal number of charachters that the destination string can hold including the

ZTC.

Return: eax  $\rightarrow$  ZTC.

StrCECopyA Procedure:

\ObjAsm\Code\ObjMem\32\StrCECopyA.asm File:

Purpose: Copy the the source ANSI string with length limitation and return the ending zero

character address.

The destination buffer should hold the maximum number of characters + 1.

Source and destination strings may overlap. Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string. Arguments:

Arg3: Maximal number of characters not including the ZTC.

eax  $\rightarrow$  ZTC. Return:

Procedure: StrCECopyW

\ObjAsm\Code\ObjMem\32\StrCECopyW.asm File:

Purpose: Copy the the source WIDE string with length limitation and return the last zero

character address.

The destination buffer should hold the maximum number of characters + 1.

Source and destination strings may overlap.

Arguments: Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string.
Arg3: Maximal number of characters not including the ZTC.

eax  $\rightarrow$  ZTC. Return:

Procedure: StrCICompA

File: \ObjAsm\Code\ObjMem\32\StrCICompA.asm

Purpose: Compare 2 ANSI strings without case sensitivity and length limitation.

Arg1:  $\rightarrow$  ANSI string 1. Arg2:  $\rightarrow$  ANSI string 2. Arguments:

Return:

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0.

Procedure: StrCICompW

File: \ObjAsm\Code\ObjMem\32\StrCICompW.asm

Compare 2 WIDE strings without case sensitivity and length limitation. Purpose:

Arg1:  $\rightarrow$  Wide string 1. Arguments:

Arg2: → Wide string 2. Return:

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0.

Procedure: StrCLengthA

File: \ObjAsm\Code\ObjMem\32\StrCLengthA.asm

Get the character count of the source ANSI string with length limitation. Purpose:

Arguments: Arg1: → Source ANSI string.
Arg3: Maximal character count. Return: eax = Limited character count.

Procedure: StrCLengthW

File:

\ObjAsm\Code\ObjMem\32\StrCLengthW.asm
Get the character count of the source WIDE string with length limitation. Purpose:

Arguments: Arg1: → Source WIDE string. Arg3: Maximal character count.

eax = Limited character count. Return:

Procedure: StrCNewA

\ObjAsm\Code\ObjMem\32\StrCNewA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrCNewT32P.inc

Purpose: Allocate a new copy of the source ANSI string with length limitation.

If the pointer to the source string is NULL or points to an empty string, StrCNewA returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter

Arguments: Arg1: → Source ANSI string.

Arg2: Maximal character count.

Return:  $eax \rightarrow New ANSI string copy.$ 

Procedure: StrCNewA\_UEFI

\ObjAsm\Code\ObjMem\32\StrCNewA\_UEFI.asm Files:

Purnose:

\ObjAsm\Code\ObjMem\Common\StrCNewT32P.inc
Allocate a new copy of the source ANSI string with length limitation.
If the pointer to the source string is NULL or points to an empty string, StrCNewA returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter.

Arguments: Arg1: → Source ANSI string.

Arg2: Maximal character count. Return:  $eax \rightarrow New ANSI string copy.$ 

Procedure: StrCNewW

\ObjAsm\Code\ObjMem\32\StrCNewW.asm Files:

\ObjAsm\Code\ObjMem\Common\StrCNewT32P.inc

Allocate a new copy of the source WIDE string with length limitation. Purnose:

If the pointer to the source string is NULL or points to an empty string, StrCNewW returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter.

Arguments: Arg1: → Source WIDE string. Arg2: Maximal character count. Return:  $eax \rightarrow New WIDE string copy.$ 

Procedure: StrCNewW\_UEFI

Files: \ObjAsm\Code\ObjMem\32\StrCNewW\_UEFI.asm \ObjAsm\Code\ObjMem\Common\StrCNewT32P.inc

Purpose:

Allocate a new copy of the source WIDE string with length limitation.

If the pointer to the source string is NULL or points to an empty string, StrCNewW returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter.

Arguments: Arg1: → Source WIDE string.

Arg2: Maximal character count.  $eax \rightarrow New WIDE string copy.$ 

Procedure: StrCompA

Return:

File: \ObjAsm\Code\ObjMem\32\StrCompA.asm

Purpose: Compare 2 ANSI strings with case sensitivity.

Arguments: Arg1:  $\rightarrow$  ANSI string 1. Arg2:  $\rightarrow$  ANSI string 2.

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0. Return:

Procedure:

\ObjAsm\Code\ObjMem\32\StrCompW.asm

Compare 2 WIDE strings with case sensitivity. Purpose:

Arguments: Arg1:  $\rightarrow$  WIDE string 1. Arg2:  $\rightarrow$  WIDE string 2.

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0. Return:

Procedure: StrCopyA

\ObjAsm\Code\ObjMem\32\StrCopyA.asm File:

Purpose: Copy an ANSI string to a destination buffer.

Arguments: Arg1: Destrination ANSI string buffer.

Arg2: Source ANSI string.

eax = Number of BYTEs copied, including the ZTC. Return:

Procedure: StrCopyW

File: \\_\ObjAsm\Code\\ObjMem\\32\\StrCopyW.asm}

Purpose: Copy a WIDE string to a destination buffer.

Arguments: Arg1: Destrination WIDE string buffer.

eax = Number of BYTEs copied, including the ZTC. Return:

Return: Nothing.

Procedure: StrCPosA

\ObjAsm\Code\ObjMem\32\StrCPosA.asm

Purpose: Scan for ANSI string2 into ANSI string1 with length limitation.

Arguments: Arg1: → Source ANSI string.
Arg2: → ANSI string to search for.
Arg3: Maximal character count.

eax → String position or NULL if not found. Return:

Procedure: StrCPosW

\ObjAsm\Code\ObjMem\32\StrCPosW.asm File:

Scan from the beginning of a WIDE string for a character. Purpose:

Arguments: Arg1: → Source WIDE string. Arg2: Character to search for.

Return:  $eax \rightarrow Character position or NULL if not found.$ 

Procedure: StrCScanA

\ObjAsm\Code\ObjMem\32\StrCScanA.asm

Purpose: Scan from the beginning of ANSI string for a character with length limitation.

Arguments: Arg1: → Source ANSI string.
Arg2: Maximal character count. Arg3: ANSI character to search for.

 $eax \rightarrow Character address or NULL if not found.$ Return:

Procedure: StrCScanW

\ObjAsm\Code\ObjMem\32\StrCScanW.asm

Purpose: Scan from the beginning of a WIDE string for a character with length limitation.

Arguments: Arg1: → Source WIDE string. Arg2: Maximal character count. Arg3: Wide character to search for.

 $eax \rightarrow Character address or NULL if not found.$ Return:

Procedure: StrDispose

\ObjAsm\Code\ObjMem\32\StrDispose.asm File:

Free the memory allocated for the string using StrNew, StrCNew, StrLENew or Purpose:

StrAlloc

If the pointer to the string is NULL, StrDispose does nothing.

Arguments: Arg1: → String.

Nothing. Return:

Procedure: StrDispose\_UEFI

\ObjAsm\Code\ObjMem\32\StrDispose\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\StrDispose\_TX\_UEFI.inc

Free the memory allocated for the string using StrNew\_UEFI, StrCNew\_UEFI, Purpose:

StrLENew\_UEFI or StrAlloc\_UEFI.

If the pointer to the string is NULL, StrDispose\_UEFI does nothing.

Arguments: Arg1: → String.

Return: Nothina.

Procedure: StrECatA

File: \ObjAsm\Code\ObjMem\32\StrECatA.asm

Append an ANSI string to another and return the address of the ending zero character. Purpose:

StrCatA does not perform any length checking. The destination buffer must have room for at least StrLengthA(Destination) + StrLengthA(Source) + 1 characters.

Arguments: Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string.

 $eax \rightarrow ZTC.$ Return:

Procedure: StrECatCharA

File: \ObjAsm\Code\ObjMem\32\StrECatCharA.asm

Append a character to an ANSI string and return the address of the ending zero. Purpose:

StrECatCharA does not perform any length checking. The destination buffer must have

enough room for at least StrLengthA(Destination) + 1 + 1 characters.

Arguments: Arg1:  $\rightarrow$  Destination ANSI string buffer. Arg2:  $\rightarrow$  ANSI character.

 $eax \rightarrow ZTC$ . Return:

Procedure: StrECatCharW

\ObjAsm\Code\ObjMem\32\StrECatCharW.asm File:

Append a character to a WIDE string and return the address of the ending zero. Purpose:

StrECatCharW does not perform any length checking. The destination buffer must have enough room for at least StrLengthw(Destination)  $+\ 1\ +\ 1$  characters.

Arguments: Arg1:  $\rightarrow$  Destination WIDE string buffer. Arg2:  $\rightarrow$  Wide character.

 $eax \rightarrow ZTC$ . Return:

Procedure:

File:

\ObjAsm\Code\ObjMem\32\StrECatw.asm
Append a WIDE string to another and return the address of the ending zero character.
StrCatw\_does not perform any length checking. The destination buffer must have room Purpose:

for at least StrLengthW(Destination) + StrLengthW(Source) + 1 characters.

Arg1: → Destination WIDE character buffer. Arg2: → Source WIDE string. Arguments:

 $eax \rightarrow ZTC.$ Return:

Procedure: StrECopyA

\ObjAsm\Code\ObjMem\32\StrECopyA.asm File:

Copy an ANSI to a buffer and return the address of the ending zero character. Purpose:

Source and destination strings may overlap. Arg1:  $\rightarrow$  Destination ANSI character buffer.

Arguments:

Arg2: → Source ANSI string.

Return:  $eax \rightarrow ZTC$ .

Procedure: StrECopyW

\ObjAsm\Code\ObjMem\32\StrECopyW.asm File:

Copy a WIDE to a buffer and return the address of the ZTC. Purpose:

Source and destination strings may overlap. Arg1:  $\rightarrow$  Destination WIDE character buffer. Arg2:  $\rightarrow$  Source WIDE string. Arguments:

 $eax \rightarrow ZTC$ . Return:

Procedure: StrEndA

\ObjAsm\Code\ObjMem\32\StrEndA.asm File:

Purpose: Get the address of the zero character that terminates the string.

Arguments: Arg1: → Source ANSI string.

 $eax \rightarrow ZTC$ . Return:

Procedure: StrEndsWithA

Files: \ObjAsm\Code\ObjMem\32\StrEndsWithA.asm

\ObjAsm\Code\ObjMem\Common\StrEndsWith\_TXP.inc

Purpose: Compare the ending of a string.

Arguments: Arg1: → Analized string.

Arg2: → Suffix string.

eax = TRUE of the ending matches, otherwise FALSE.

Procedure: StrEndsWithW

\ObjAsm\Code\ObjMem\32\StrEndsWithW.asm \ObjAsm\Code\ObjMem\Common\StrEndsWith\_TXP.inc Files:

Purpose: Compare the ending of a string.

Arguments: Arg1: → Analized string.

Arg2: → Suffix string.

Return: eax = TRUE of the ending matches, otherwise FALSE.

Procedure: StrEndW

\ObjAsm\Code\ObjMem\32\StrEndW.asm

Get the address of the zero character that terminates the string. Purpose:

Arguments: Arg1:  $\rightarrow$  Source WIDE string. Return: eax  $\rightarrow$  ZTC.

Procedure: StrFillChrA

Files: \ObjAsm\Code\ObjMem\32\StrFillChrA.asm

\ObjAsm\Code\ObjMem\Common\StrFillChr\_TXP.inc Purpose: Fill a preallocated String with a character.

Arguments: Arg1: → String.

Arg2: Character. Arg3: Character Count.

Nothing. Return:

Procedure: StrFillChrW

Files:

\ObjAsm\Code\ObjMem\32\StrFillChrw.asm \ObjAsm\Code\ObjMem\Common\StrFillChr\_TXP.inc Fill a preallocated String with a character. Purpose:

Arg1:  $\rightarrow$  String. Arguments:

Arg2: Character. Arg3: Character Count.

Return: Nothing. Procedure: StrFilterA

\ObjAsm\Code\ObjMem\32\StrFilterA.asm File:

Perform a case sensitive string match test using wildcards (\* and ?).

Arguments: Arg1: → Source ANSI string.

Arg2: → Filter ANSI string.

Return: eax = TRUE if strings match, otherwise FALSE.

Procedure: StrFilterW

\ObjAsm\Code\ObjMem\32\StrFilterW.asm File:

Purpose: Perform a case sensitive string match test using wildcards (\* and ?).

Arguments: Arg1: → Source WIDE string. Arg2: → Filter WIDE string.

Return: eax = TRUE if strings match, otherwise FALSE.

Procedure: StrICompA

\ObjAsm\Code\ObjMem\32\StrICompA.asm

Compare 2 ANSI strings without case sensitivity. Purpose:

Arguments: Arg1:  $\rightarrow$  ANSI string 1. Arg2:  $\rightarrow$  ANSI string 2.

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0. Return:

Procedure: StrICompW

\ObjAsm\Code\ObjMem\32\StrICompW.asm

Purpose: Compare 2 WIDE strings without case sensitivity.

Arguments: Arg1:  $\rightarrow$  Wide string 1. Arg2:  $\rightarrow$  Wide string 2.

Return:

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0.

Procedure: StrTFilterA

File: \ObjAsm\Code\ObjMem\32\StrIFilterA.asm

Purpose: Perform a case insensitive string match test using wildcards (\* and ?).

Arguments: Arg1:  $\rightarrow$  Source ANSI string. Arg2:  $\rightarrow$  Filter ANSI string.

eax = TRUE if strings match, otherwise FALSE.

Procedure: StrIFilterW

File: \ObjAsm\Code\ObjMem\32\StrIFilterw.asm

Purpose: Perform a case insensitive string match test using wildcards (\* and ?). Arguments: Arg1:  $\rightarrow$  Source WIDE string.

Arg2: → Filter WIDE string.

Return: eax = TRUE if strings match, otherwise FALSE.

Procedure: StrLeftA

\ObjAsm\Code\ObjMem\32\StrLeftA.asm

Extract the left n characters of the source ANSI string.

Arguments: Arg1: → Destination character buffer. Arg2: → Source ANSI string.

Return: eax = Number of copied characters, not including the ZTC.

Procedure: StrLeftW

File:

\ObjAsm\Code\ObjMem\32\StrLeftw.asm
Extract the left n characters of the source WIDE string. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: → Source WIDE string. eax = Number of copied characters, not including the ZTC. Return:

Procedure: StrLengthA

\ObjAsm\Code\ObjMem\32\StrLengthA.asm

Purpose: Determine the length of an ANSI string not including the zero terminating character. Arguments: Arg1: → Source ANSI string.

eax = Length of the string in characters. Return:

Procedure: StrLengthW

\ObjAsm\Code\ObjMem\32\StrLengthW.asm File:

Purpose: Determine the length of a WIDE string not including the zero terminating character.

Arguments: Arg1: → Wide string.

eax = Length of the string in characters. Return:

Procedure: StrLowerA

\ObjAsm\Code\ObjMem\32\StrLowerA.asm

Purpose: Convert all ANSI string characters into lowercase.

Arguments: Arg1: → Source ANSI string.

 $eax \rightarrow string.$ Return:

Procedure: StrLowerW

\ObjAsm\Code\ObjMem\32\StrLowerW.asm File:

Purpose: Convert all WIDE string characters into lowercase.

Arguments: Arg1: → Source WIDE string.

Return: eax  $\rightarrow$  string.

Procedure: StrLRTrimA

\ObjAsm\Code\ObjMem\32\StrLRTrimA.asm

Trim blank characters from the beginning and the end of an ANSI string. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination ANSI character buffer. Arg2:  $\rightarrow$  Source ANSI string.

Return: Nothina.

Procedure: StrLRTrimW

File:

\\\\ObjAsm\\Code\\ObjMem\\32\\StrLRTrimW.asm\\\
Trim blank characters from the beginning and the end of a WIDE string. Purpose:

Arguments: Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string.

Return: Nothing.

Procedure: StrLScanA

Frile: \ \ObjAsm\Code\ObjMem\32\StrLScanA.asm

Purpose: Scan for a character from the beginning of an ANSI string.

Arguments: Arg1: → Source ANSI string.

Arg2: Character to search.

Return: eax → Character address or NULL if not found.

Procedure: StrLScanW

File: \ObjAsm\Code\ObjMem\32\StrLScanw.asm

Purpose: Scan for a character from the beginning of a WIDE string.

Arguments: Arg1: → Source WIDE string. Arg2: Character to search for

Return:  $eax \rightarrow Character address or NULL if not found.$ 

Procedure: StrLTrimA

File:

\ObjAsm\Code\ObjMem\32\StrLTrimA.asm
Trim blank characters from the beginning of an ANSI string. Purpose:

Arguments: Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string.

Return: Nothing.

Procedure: StrLTrimW

\ObjAsm\Code\ObjMem\32\StrLTrimW.asm

Purpose: Trim blank characters from the beginning of a WIDE string.

Arguments: Arg1:  $\rightarrow$  Destination WIDE character buffer. Arg2:  $\rightarrow$  Source WIDE string.

Return: Nothing.

Procedure: StrMidA

\ObjAsm\Code\ObjMem\32\StrMidA.asm File:

Extract a substring from an ANSI source string. Purpose: Arguments: Arg1: → Destination ANSI character buffer.

Arg2: → Source ANSI string.

Arg3: Start character index. Index ranges [1 .. String length].

Arg3: Character count.

eax = Number of copied characters. Return:

Procedure: StrMidW

\ObjAsm\Code\ObjMem\32\StrMidw.asm File:

Extract a substring from a WIDE source string. Purpose: Arguments: Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string.

Arg3: Start character index. Index ranges [1 .. String length].

Arg3: Character count. eax = Number of copied characters. Return:

Procedure: StrMoveW

\ObjAsm\Code\ObjMem\32\StrMoveW.asm

Move part of a WIDE string. The ending zero charactrer is not appended automatically. Purpose:

Source and destination strings may overlap.

Arg1: → Destination buffer. Arg2: → Source WIDE string. Arg3: Character count. Arguments:

Nothing. Return:

Procedure: StrNewA

Files: \ObjAsm\Code\ObjMem\32\StrNewA.asm

\ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc

Allocate a new copy of the source string. Purpose:

If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate any memory space. Otherwise, StrNew makes a duplicate of the source string.

The allocated memory space is Length(String) + ZTC.

Arg1: → Source WIDE string.

Arguments:

 $eax \rightarrow New string copy.$ Return:

Procedure: StrNewA\_UEFI

\ObjAsm\Code\ObjMem\32\StrNewA\_UEFI.asm Files: \ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc Purpose: Allocate a new copy of the source string.

If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate

any memory space. Otherwise, StrNew makes a duplicate of the source string.
The allocated memory space is Length(String) + ZTC.

Arguments: Arg1: → Source WIDE string.

Return:  $eax \rightarrow New string copy.$ 

Procedure: StrNewW

\ObjAsm\Code\ObjMem\32\StrNewW.asm Files:

\ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc Purpose:

Allocate a new copy of the source string. If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate

any memory space. Otherwise, StrNew makes a duplicate of the source string. The allocated memory space is Length(String) + ZTC.
Arg1: → Source WIDE string.

Arguments:  $eax \rightarrow New string copy.$ Return:

Procedure: StrNewW UEFI

\ObjAsm\Code\ObjMem\32\StrNewW\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc

Purpose:

Allocate a new copy of the source string.
If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate

any memory space. Otherwise, StrNew makes a duplicate of the source string.

The allocated memory space is Length(String) + ZTC.
Arguments: Arg1: → Source WIDE string.

Return:  $eax \rightarrow New string copy.$ 

Procedure: StrPosA

\ObjAsm\Code\ObjMem\32\StrPosA.asm
Find the occurence of string 2 into string1. File:

Purpose:

Arguments: Arg1: → Source ANSI string.
Arg2: → Searched ANSI string.

 $eax \rightarrow string$  occurence or NULL if not found.

Procedure: StrPosW

Return:

File:

\ObjAsm\Code\ObjMem\32\StrPosW.asm
Find the occurence of string 2 into string1. Purpose:

Arguments: Arg1: → Source WIDE string.

Arg2: → Searched WIDE string.

 $eax \rightarrow string$  occurrence or NULL if not found. Return:

Procedure: StrRepChrA

\ObjAsm\Code\ObjMem\32\StrRepChrA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrRepChr\_TXP.inc Create a new string filled with a given char.

Purpose:

Arguments: Arg1: Used character.

Arg2: Repetition count

 $eax \rightarrow New string or NULL if failed.$ Return:

Procedure: StrRepChrW

\ObjAsm\Code\ObjMem\32\StrRepChrw.asm

\ObjAsm\Code\ObjMem\Common\StrRepChr\_TXP.inc Create a new string filled with a given char.

Purpose: Arguments: Arg1: Used character.

Arg2: Repetition count.

 $eax \rightarrow New string or NULL if failed.$ Return:

Procedure: StrRightA

\ObjAsm\Code\ObjMem\32\StrRightA.asm File:

Purpose: Copy the right n characters from the source string into the destination buffer. Arguments: Arg1:  $\rightarrow$  Destination ANSI character buffer.

Arg2: → Source ANSI string.

Arg3: Character count.

Return: Nothing.

Procedure: StrRightW

\ObiAsm\Code\ObjMem\32\StrRightW.asm File:

Purpose: Copy the right n characters from the source string into the destination buffer.

Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string.

Arg3: Character count.

Return: Nothing.

Procedure: StrRScanA

\ObjAsm\Code\ObjMem\32\StrRScanA.asm

Purpose: Scan from the end of an ANSI string for a character.

Arguments: Arg1: → Source ANSI string.

Arg2: Character to search for.

Return: eax → Character address or NULL if not found.

Procedure: StrRScanW

\ObjAsm\Code\ObjMem\32\StrRScanW.asm File:

Purpose: Scan from the end of a WIDE string for a character.

Arguments: Arg1: → Source WIDE string.

Arg2: Character to search for.

Return: eax → Character address or NULL if not found.

Procedure: StrRTrimA

File: \ObjAsm\Code\ObjMem\32\StrRTrimA.asm

Trim blank characters from the end of an ANSI string. Purpose:

Arguments: Arg1: → Destination ANSI character buffer.
Arg2: → Source ANSI string.

Return: Nothing.

Procedure: StrRTrimW

File:

\ObjAsm\Code\ObjMem\32\StrRTrimW.asm
Trim blank characters from the end of a WIDE string. Purpose:

Arguments: Arg1: → Destination WIDE character buffer. Arg2: → Source WIDE string.

Return: Nothing.

Procedure: StrSizeA

\ObjAsm\Code\ObjMem\32\StrSizeA.asm File:

Determine the size of an ANSI string including the zero terminating character (ZTC). Purpose:

Arguments: Arg1:  $\rightarrow$  ANSI string. Return: eax = Size of the string in BYTEs. Return:

Procedure: StrSizeW

\ObjAsm\Code\ObjMem\32\StrSizeW.asm

Determine the size of a WIDE string including the zero terminating character (ZTC). Purpose:

Arguments: Arg1: → Wide string.

eax = Size of the string in BYTEs. Return:

Procedure: StrStartsWithA

\ObjAsm\Code\ObjMem\32\StrStartsWithA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrStartsWith\_TXP.inc

Purpose: Compare the beginning of a string.
Arguments: Arg1: → Analized string.
Arg2: → Prefix string.

eax = TRUE of the beginning matches, otherwise FALSE. Return:

Procedure: StrStartsWithW

\ObjAsm\Code\ObjMem\32\StrStartsWithW.asm

\ObjAsm\Code\ObjMem\Common\StrStartsWith\_TXP.inc

Compare the beginning of a string. Purpose:

Arguments: Arg1: → Analized string.

Arg2: → Prefix string.

Return: eax = TRUE of the beginning matches, otherwise FALSE.

Procedure: StrToStOA

File:

\ObjAsm\Code\ObjMem\32\StrToStOA.asm Load an ANSI string representation of a floating point number into the st(0) Purpose:

FPU register.

Arguments: Arg1: → ANSI string floating point number.

Return:

eax = Result code f\_OK or f\_ERROR.
- Based on the work of Raymond Filiatreault (FpuLib).

- Source string should not be greater than 19 chars + zero terminator.

Procedure: StrToStOW

\ObjAsm\Code\ObjMem\32\StrToStOw.asm File:

Load a WIDE string representation of a floating point number into the st(0) Purpose:

FPU register.

Arguments: Arg1: → ANSI string floating point number.

Return:

eax = Result code f\_OK or f\_ERROR.
- Based on the work of Raymond Filiatreault (FpuLib). Note:

- Source string should not be greater than 19 chars + zero terminator.

Procedure: StrUpperA

\ObjAsm\Code\ObjMem\32\StrUpperA.asm File:

Convert all ANSI string characters into uppercase. Purpose:

Arguments: Arg1: → Source ANSI string.

 $eax \rightarrow String.$ Return:

Procedure: StrUpperW

\ObjAsm\Code\ObjMem\32\StrUpperW.asm File:

Purpose: Convert all WIDE string characters into uppercase.

Arguments: Arg1: → Source WIDE string.

 $eax \rightarrow String.$ Return:

Procedure: StrW2StrA

File: \ObjAsm\Code\ObjMem\32\Strw2StrA.asm

Convert a WIDE string into an ANSI string. Wide characters are converted to BYTEs by decimation of the high byte. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination ANSI character buffer. Arg2:  $\rightarrow$  Source WIDE string.

Return: Nothing.

Procedure: SysShutdown

\ObjAsm\Code\ObjMem\32\SysShutdown.asm

Shut down the system. Purnose: Arguments: Arg1: Shutdown type.

Arg2: Shutdown reason (see System Shutdown Reason Codes).

Nothing. Return:

Procedure: SysStandby

\ObjAsm\Code\ObjMem\32\SysStandby.asm File: Purpose: Set the system in standby modus.

Arguments: None. Return: Nothing.

Procedure: uCRC32C

\ObjAsm\Code\ObjMem\32\uCRC32C.asm Files:

\ObjAsm\Code\ObjMem\Common\uCRC32C\_XP.inc

Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an unaligned Purpose:

Purpose: Compute the CRC 325 (CHI 13) memory block.

Arguments: Arg1: → Unaligned memory block.

Arg2: Memory block size in BYTES.

Return: eax = CRC32C.

Procedure: udword2decA

\ObjAsm\Code\ObjMem\32\udword2decA.asm File:

Purpose: Convert an unsigned DWORD to its decimal ANSI string representation. Arguments: Arg1: → Destination ANSI string buffer.

Arg2: DWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 11 BYTEs large to allocate the output string Note:

(10 ANSI characters + ZTC = 11 BYTEs).

Procedure: udword2decW

File: \ObjAsm\Code\ObjMem\32\udword2decw.asm

Convert an unsigned DWORD to its decimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination WIDE string buffer.

Arg2: DWORD value.

Return: eax = Number of BYTEs copied to the destination buffer, including the ZTC.

The destination buffer must be at least 22 BYTEs large to allocate the output string Note:

(10 WIDE characters + ZTC = 22 BYTEs).

Procedure: UefiGetErrStrA

\ObjAsm\Code\ObjMem\32\UefiGetErrStrA.asm Files:

\ObjAsm\Code\ObjMem\Common\UefiGetErrStr\_TX.inc

Return a description ANSI string from an UEFI error code. Purnose:

Arguments: Arg1: UEFI error code.  $eax \rightarrow Error string.$ Return:

Procedure: UefiGetErrStrW

Files: \ObjAsm\Code\ObjMem\32\UefiGetErrStrw.asm

\ObjAsm\Code\ObjMem\Common\UefiGetErrStr\_TX.inc

Return a description WIDE string from an UEFI error code. Purpose:

Arguments: Arg1: UEFI error code.  $eax \rightarrow Error string.$ Return:

Procedure: uaword2decA

Files: \ObjAsm\Code\ObjMem\32\uqword2decA.asm

\ObjAsm\Code\ObjMem\Common\uqword2decT32.inc
Convert an unsigned QWORD to its decimal ANSI string representation. Purnose:

Arguments: Arg1: → Destination ANSI string buffer.

Arg2: QWORD value.

Return: eax = Number of BYTEs copied to the destination buffer, including the ZTC.

Note: The destination buffer must be at least 21 BYTEs large to allocate the output string

(20 ANSI characters + ZTC = 21 BYTEs).

Procedure: ugword2decW

\ObjAsm\Code\ObjMem\32\uqword2decW.asm Files:

\ObjAsm\Code\ObjMem\Common\uqword2decT32.inc
Convert an unsigned QWORD to its decimal WIDE string representation. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination WIDE string buffer. Arg2: QWORD value.

Return:

eax = Number of BYTEs copied to the destination buffer, including the ZTC. The destination buffer must be at least 42 BYTEs large to allocate the output string Note:

(20 WIDE characters + ZTC = 42 BYTEs).

Procedure: UTF8ToWide

\ObjAsm\Code\ObjMem\32\UTF8ToWide.asm Files:

\ObjAsm\Code\ObjMem\Common\UTF8ToWide\_XP.inc

Convert an UTF8 byte stream to a WIDE (UTF16) string. Purpose:

Arguments: Arg1: → Destination WIDE buffer.

Arg2: → Source UTF8 BYTE stream. Must be zero terminated.

Arg3: Destination buffer size in BYTEs.

eax = Number of BYTEs written. Return:

ecx = 0: succeeded 1: buffer full 2: conversion error

- The destination WIDE string is always terminated with a ZTC Notes:

(only if buffer size >= 2).

Procedure: WaitForProcess

\ObjAsm\Code\ObjMem\32\WaitForProcess.asm File:

Synchronisation procedure that waits until a process has finished. Purpose:

Arguments: Arg1: Process ID

Arg2: Timeout value in ms.

Return: eax = Wait result (WAIT\_ABANDONED, WAIT\_OBJECT\_0 or WAIT\_TIMEOUT).

Procedure: WideToUTF8

\ObjAsm\Code\ObjMem\32\WideToUTF8.asm Files:

\ObjAsm\Code\ObjMem\Common\WideToUTF8\_XP.inc

Purpose: Convert an WIDE string to an UTF8 encoded stream.

Arg1: → Destination buffer. Arg2: → Source WIDE string. Arguments:

Arg3: Destination buffer size in BYTEs.

eax = Number of BYTEs written. Return:

ecx = 0: succeeded 1: buffer full

- The destination stream is always zero terminated. Notes:

Procedure: WndFadeIn

\ObjAsm\Code\ObjMem\32\WndFadeIn.asm File:

Fade in a window when WS\_EX\_LAYERED is set. Purpose:

Arguments: Arg1: Window HANDLE.

Arg2: Transparency start value. Arg3: Transparency end value. Arg4: Transparency increment value. Arg5: Delay between steps.

Return: Nothing.

Procedure: WndFadeOut

\ObjAsm\Code\ObjMem\32\WndFadeOut.asm File:

Fade out a window when WS\_EX\_LAYERED is set. Purpose:

Arguments: Arg1: Window HANDLE.

Arg2: Transparency start value. Arg3: Transparency end value. Arg4: Transparency decrement value.

Arg5: Delay between steps.

Nothing. Return:

Procedure: word2hexA

File: \ObjAsm\Code\ObjMem\32\word2hexA.asm

Purpose: Convert a WORD to its hexadecimal ANSI string representation.

Arguments: Arg1: → Destination ANSI string buffer.

Arg2: WORD value.

Return: Nothing.

Notes: The destination buffer must be at least 5 BYTEs large to allocate the output string

(4 character BYTES + ZTC = 5 BYTES).

Procedure: word2hexW

\ObjAsm\Code\ObjMem\32\word2hexW.asm Convert a WORD to its hexadecimal WIDE string representation. Purpose:

Arg1: 

Destination WIDE string buffer.

Arg2: WORD value.

Nothing. Arguments:

Return:

The destination buffer must be at least 9 BYTEs large to allocate the output string Notes:

(4 character WORDS + ZTC = 9 BYTEs).

## 9.64 Bit Code

Procedure: aCRC32C

Files.

ObjAsm\Code\ObjMem\64\aCRC32C.asm \ObjAsm\Code\ObjMem\Common\aCRC32C\_XP.inc

Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an aligned Purpose:

memory block.

Arguments: Arg1: → Aligned memory block.

Arg2: Memory block size in BYTEs.

Return: eax = CRC32C.

Procedure: ActivatePrevInstanceA

File: \ObjAsm\Code\ObjMem\64\ActivatePrevInstanceA.asm
Purpose: Activate a previously existing instance of an application.
Arguments: Arg1: \to ANSI application name.

Arg2: → ANSI class name. rax = TRUE if activated, otherwise FALSE. Return:

Procedure: ActivatePrevInstanceW

\ObjAsm\Code\ObjMem\64\ActivatePrevInstanceW.asm

Activate a previously existing instance of an application.

Arguments: Arg1: → WIDE application name.

Arg2: → WIDE class name.

rax = TRUE if activated, otherwise FALSE. Return:

Procedure: AreVisualStylesEnabled

File:

\\\\ObjAsm\\Code\\ObjMem\\64\AreVisualStylesEnabled.asm\\\
Determine if there is an activated theme for the running application Purpose:

Arguments: None.

Return: rax = TRUE if the application is themed, otherwise FALSE.

Procedure: bin2dwordA

\ObjAsm\Code\ObjMem\64\bin2dwordA.asm

Load an ANSI string binary representation of a DWORD.

Arguments: Arg1: → ANSI binary string.

eax = DWORD.Return:

Procedure: bin2dwordW

\ObjAsm\Code\ObjMem\64\bin2dwordw.asm File:

Purpose: Load an WIDE string binary representation of a DWORD. Arguments: Arg1:  $\rightarrow$  WIDE binary string.

Return: eax = DWORD.

Procedure: bin2qwordA

File: \ObjAsm\Code\ObjMem\64\bin2qwordA.asm

Purpose: Purpose: Load an ANSI string binary representation of a QWORD. Arguments: Arg1: → ANSI binary string.

Return: rax = QWORD.

Procedure: bin2gwordW

File:

\(\ObjAsm\Code\ObjMem\64\bin2qwordW.asm\)
\(\Load\) an WIDE string binary representation of a QWORD. Purpose:

Arguments: Arg1: → Wide binary string.

Return: rax = QWORD.

Procedure: Bmp2Rgn

File: \ObjAsm\Code\ObjMem\64\Bmp2Rqn.asm

Purpose: Create a GDI region based on a device dependant or independent bitmap (DDB or DIB).
This region is defined by the non transparent area delimited by the transparent color.

Arguments: Arg1: Bitmap HANDLE.

Arg2: RGB transparet color.

rax = Region HANDLE or zero if failed. Return:

Procedure: BStrAlloc

\ObjAsm\Code\ObjMem\64\BStrAlloc.asm

Allocate space for a BStr with n characters. The length field is set to zero.

Arguments: Arg1: Character count.

 $rax \rightarrow New$  allocated BStr or NULL if failed.

Procedure: BStrCat

\ObjAsm\Code\ObjMem\64\BStrCat.asm File:

Concatenate 2 BStrs. Purpose: Arguments: Arg1: Destrination BStr. Arg2: Source BStr.

Return: Nothing.

Procedure: BStrCatChar

File: \ObjAsm\Code\ObjMem\64\BStrCatChar.asm Purpose: Append a character to the end of a BStr.

Arguments: Arg1: Destrination BStr.

Arg2: WIDE character.

Return: Nothing.

Procedure: BStrCCatChar

File:

\ObjAsm\Code\ObjMem\64\BStrCCatChar.asm
Append a WIDE character to a BStr with length limitation. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination BStr. Arg2:  $\rightarrow$  WIDE character.

 $rax \rightarrow BStr or NULL if failed.$ Return:

Procedure: BStrCECat

File:

\ObjAsm\Code\ObjMem\64\BStrCECat.asm
Concatenate 2 BStrs with length limitation and return the the address of the ZTC. Purpose:

The destination string buffer should have at least enough room for the maximum number

of characters + 1.

Arguments: Arg1: → Destination BStr. Arg2: → Source BStr.

Arg3: Maximal number of charachters the destination string can hold including the ZTC.

 $rax = NULL \text{ or } \rightarrow ZTC.$ Return:

Procedure: BStrCECopy

File: \ObjAsm\Code\ObjMem\64\BStrCECopy.asm

Copy the the source BStr with length limitation and return the address of the ZTC. Purpose:

The destination buffer should hold the maximum number of characters + 1.

Arguments: Arg1: → Destination BStr.

Arg2: → Source BStr.

Arg3: Maximal number of charachters the destination string can hold including the ZTC.

Return:  $rax = NULL or \rightarrow ZTC.$ 

Procedure: BStrCNew

File: \ObjAsm\Code\ObjMem\64\BStrCNew.asm

Allocate a new copy of the source BStr with length limitation. Purpose:

If the pointer to the source string is NULL, BStrCNew returns NULL and doesn't allocate any space. Otherwise, StrCNew makes a duplicate of the source string. The maximal size of the new string is limited to the second parameter.

Arguments: Arg1: → Source BStr.
Arg2: Maximal character count.

 $rax \rightarrow New BStr copy or NULL.$ Return:

Procedure: BStrCopy

File: \ObjAsm\Code\ObjMem\64\BStrCopy.asm Purpose: Copy a BStr to a destination buffer.

Arguments: Arg1: Destrination BStr buffer.

Arg2: Source BStr.

Return: Nothing.

Procedure: BStrCScan

\ObjAsm\Code\ObjMem\64\BStrCScan.asm File:

Purpose: Scan from the beginning of a BStr for a character with length limitation.

Arguments: Arg1: → Source WIDE string. Arg2: Maximal character count. Arg3: WIDE character to search for.

Return:  $rax \rightarrow Character address or NULL if not found.$ 

Procedure: BStrDispose

\ObjAsm\Code\ObjMem\64\BStrDispose.asm File:

Purpose: Free the memory allocated for the string using BStrNew, BStrCNew, BStrLENew or

If the pointer to the string is NULL, BStrDispose does nothing.

Arg1:  $\rightarrow$  BStr. Arguments: Nothing. Return:

Procedure: BStrECat

\ObjAsm\Code\ObjMem\64\BStrECat.asm

Append a BStr to another and return the address of the ZTC. Purpose:

BStrCat does not perform any length checking. The destination buffer must have room for at least BStrLength(Destination) + BStrLength(Source) + 1 characters.

Arguments: Arg1:  $\rightarrow$  Destination BStr buffer. Arg2:  $\rightarrow$  Added BStr.

 $rax \rightarrow ZTC.$ Return:

Procedure: BStrECatChar

\ObjAsm\Code\ObjMem\64\BStrECatChar.asm File:

Purpose:

Append a WIDE character to a BStr and return the address of the ZTC. BStrECatChar does not perform any length checking. The destination buffer must have

enough room for at least BStrLength(Destination) + 1 + 1 characters.

Arguments: Arg1:  $\rightarrow$  Destination BStr buffer. Arg2:  $\rightarrow$  WIDE character.

 $rax \rightarrow ZTC$ . Return:

Procedure: BStrECopy

\ObjAsm\Code\ObjMem\64\BStrECopy.asm File:

Purpose: Copy a BStr to a buffer and return the address of the ZTC.

Source and destination strings may overlap.

Arguments: Arg1: → Destination BStr buffer. Arg2: → Source BStr buffer.

Return:  $rax \rightarrow ZTC$ .

Procedure: BStrEnd

File: \ObjAsm\Code\ObjMem\64\BStrEnd.asm

Purpose: Get the address of the ZTC.
Arguments: Arg1: → Source BStr.
Return: rax → ZTC.

Procedure: BStrEndsWith

Files: \ObjAsm\Code\ObjMem\64\BStrEndsWith.asm

\ObjAsm\Code\ObjMem\Common\BStrEndsWith\_X.inc

Compare the ending of a BSTR. Purpose:

Arguments: Arg1: → Analized BSTR.

Arg2: → Suffix BSTR.

Return: eax = TRUE of the ending matches, otherwise FALSE.

Procedure: BStrFillChr

Files: \ObjAsm\Code\ObjMem\64\BStrFillChr.asm

\ObjAsm\Code\ObjMem\Common\BStrFillChr\_TX.inc

Fill a preallocated BSTR with a character. Purpose:

Arguments: Arg1: → String.

Arg2: Character.

Arg3: Character Count.

Nothing. Return:

Procedure: BStrLeft

File: \ObjAsm\Code\ObjMem\64\BStrLeft.asm

Purpose: Extract the left n characters of the source BStr.

Arguments: Arg1: → Destination BStr.

Arg2: → Source BStr.

Return: rax = Number of copied characters, not including the ZTC.

Procedure: BStrLength

\ObjAsm\Code\ObjMem\64\BStrLength.asm File:

Purpose: Determine the length of a BStr not including the ZTC.

Arguments: Arg1: → Source BStr.
Return: rax = Length of the string in characters.

Procedure: BStrLRTrim

File:

\ObjAsm\Code\ObjMem\64\BStrLRTrim.asm
Trim blank characters from the beginning and the end of a BStr. Purpose:

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source BStr.

Return: Nothing.

Procedure: BStrLTrim

File:

\ObjAsm\Code\ObjMem\64\BStrLTrim.asm
Trim blank characters from the beginning of a BStr. Purpose:

Arguments: Arg1: → Destination BStr buffer. Arg2: → Source BStr.

Return: Nothing.

Procedure: BStrMid

\ObjAsm\Code\ObjMem\64\BStrMid.asm File: Extract a substring from a BStr string. Purpose:

Arg1: → Destination BStr buffer. Arguments:

Arg2: → Source Bstr.

Arg3: Start character index. Index ranges [0 .. length-1].

Arg3: Character count.

Return: eax = String length.

Procedure: BStrMove

File:

\ObjAsm\Code\ObjMem\64\BStrMove.asm
Move part of a BStr. The ZTC is not appended automatically. Purpose:

Source and destination strings may overlap.

Arguments:

Arg1: → Destination BStr. Arg2: → Source BStr. Arg3: Character count.

Return: Nothing.

Procedure:

\ObjAsm\Code\ObjMem\64\BStrNew.asm Allocate an new copy of the source BStr. Purpose:

If the pointer to the source string is NULL or points to an empty string, BStrNew

returns NULL and doesn't allocate any heap space. Otherwise, BStrNew makes a duplicate

of the source string.

The allocated space is Length(String) + 1 character.

Arguments: Arg1: → Source BStr.

 $rax \rightarrow New BStr copy or NULL.$ Return:

Procedure: BStrRepChr

Files: \ObjAsm\Code\ObjMem\64\BStrRepChr.asm

\ObjAsm\Code\ObjMem\Common\BStrRepChr\_X.inc

Create a new BSTR filled with a given char. Purpose:

Arguments: Arg1: Used character.
Arg2: Repetition count.

Return:  $xax \rightarrow New BSTR or NULL if failed.$ 

Procedure: BStrRight

\ObjAsm\Code\ObjMem\64\BStrRight.asm File:

Copy the right n characters from the source string into the destination BStr, that Purpose:

must

have enought room for the new BStr.

Arguments: Arg1: → Destination BStr buffer.

Arg2: → Source BStr. Arg3: Character count. rax = Copied characters.

Procedure: BStrRTrim

Return:

File:

\ObjAsm\Code\ObjMem\64\BStrRTrim.asm
Trim blank characters from the end of a BStr. Purpose:

Arguments: Arg1: → Destination BStr buffer. Arg2: → Source BStr.

Return: Nothing.

Procedure: BStrSize

\ObjAsm\Code\ObjMem\64\BStrSize.asm

Determine the size of a BStr including the ZTC + leading DWORD. Purpose:

Arguments: Arg1: → Source BStr.

Return: rax = String size including the length field and ZTC in BYTEs.

Procedure: BStrStartsWith

\ObjAsm\Code\ObjMem\64\BStrStartsWith.asm Files:

\ObjAsm\Code\ObjMem\Common\BStrStartsWith\_X.inc

Purpose: Compare the beginning of a BSTR.

Arguments: Arg1: → Analized BSTR.

Arg2: → Prefix BSTR.

eax = TRUE of the beginning matches, otherwise FALSE. Return:

Procedure: byte2hexA

\ObjAsm\Code\ObjMem\64\byte2hexA.asm File:

Convert a BYTE to its hexadecimal ANSI string representation. Purpose:

Arg1: → Destination ANSI string buffer. Arg2: BYTE value. Arguments:

Nothing. Return:

Notes: The destination buffer must be at least 3 BYTEs large to allocate the output string

(2 character BYTES + ZTC = 3 BYTES).

Procedure: byte2hexW

\ObjAsm\Code\ObjMem\64\byte2hexW.asm File:

Purpose: Convert a BYTE to its hexadecimal WIDE string representation.

Arg1:  $\rightarrow$  Destination WIDE string buffer. Arg2: BYTE value. Arguments:

Return: Nothing.

Notes: The destination buffer must be at least 5 BYTEs large to allocate the output string

(2 character WORDS + ZTC = 5 BYTES).

Procedure: CalcVarianceDW

\ObjAsm\Code\ObjMem\64\CalcVarianceDW.asm Files:

\ObjAsm\Code\ObjMem\Common\CalcVariance\_XP.inc

Purpose: Calculate the MSE of an array of DWORDs.

Arguments:  $Arg1: \rightarrow Array of DWORDs.$ 

Arg2: QWORD Array count.

Arg3: → Variance.

eax = TRUE is succeeded, otherwise FALSE. Return:

Links:

https://www.mun.ca/biology/scarr/Simplified\_calculation\_of\_variance.html#:~:text=A%

20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202 https://www.mun.ca/biology/scarr/Mean\_&\_Variance.html#:~:text=easily%20calculated%20as Var = Y2/N-(Y/N)^2 or (Y2\*N-Y^2)/N^2 where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.

Formulas:

Procedure: calcvarianceQw

Files: \ObjAsm\Code\ObjMem\64\CalcVarianceQW.asm

\ObjAsm\Code\ObjMem\Common\CalcVariance\_XP.inc

Purpose: Calculate the MSE of an array of QWORDs.

 $Arg1: \rightarrow Array of QWORDs.$ Arguments:

Arg2: QWORD Array count.

Arg3: → Variance.

Return: eax = TRUE is succeeded, otherwise FALSE.

Links:

 $\label{lem:https://www.mun.ca/biology/scarr/Mean_&_variance.html#:~:text=easily%20calculated%20as $$ Var = Y2/N-(Y/N)^2 $ or $(Y2*N-Y^2)/N^2 $$ where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.$ 

Formulas:

Procedure: CalcvarianceR4

\ObjAsm\Code\ObjMem\64\CalcVarianceR4.asm Files:

\ObjAsm\Code\ObjMem\Common\CalcVariance\_XP.inc

Calculate the MSE of an array of REAL4s. Purpose:

Arguments:

Arg1: → Array of REAL4s. Arg2: QWORD Array count.

Arg3: → Variance.

Return: eax = TRUE is succeeded, otherwise FALSE.

Tinks:

https://www.mun.ca/biology/scarr/Simplified\_calculation\_of\_variance.html#:~:text=A%

https://www.mun.ca/biology/scarr/Mean\_&\_Variance.html#:~:text=easily%20calculated%20as

Formulas:  $Var = Y2/N-(Y/N)^2$  or  $(Y2*N-Y^2)/N^2$ 

where Y: Sum(y), Y2:  $Sum(y^2)$ , N:Population count = Array size.

Procedure: CalcVarianceR8

Files: \ObjAsm\Code\ObjMem\64\CalcvarianceR8.asm

\ObjAsm\Code\ObjMem\Common\CalcVariance\_XP.inc

Calculate the MSE of an array of REAL8s. Purpose: Arguments:

Arg1: → Array of REAL8s. Arg2: QWORD Array count.

Arg3: → Variance.

eax = TRUE is succeeded, otherwise FALSE. Return:

Links:

https://www.mun.ca/biology/scarr/Simplified\_calculation\_of\_variance.html#:~:text=A%

20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202

https://www.mun.ca/biology/scarr/Mean\_&\_variance.html#:~:text=easily%20calculated%20as

Formulas:  $Var = Y2/N-(Y/N)^2$  or  $(Y2*N-Y^2)/N^2$ 

where Y: Sum(y), Y2:  $Sum(y^2)$ , N:Population count = Array size.

Procedure: CenterForm

Return:

\ObjAsm\Code\ObjMem\64\CenterForm.asm File:

Purpose: Calculate the starting coordinate of a window based on the screen and the window size.

Arguments: Arg1: Window size in pixel.
Arg2: Screen size in pixel.

eax = Starting point in pixel.

Procedure: ComEventsAdvice

\ObjAsm\Code\ObjMem\64\ComEventsAdvice.asm Files:

\ObjAsm\Code\ObjMem\Common\ComEventsAdvice\_X.inc

Notificate the Event source that pISink will recieve Events. Purpose:

Arguments: Arg1: → Any Source Object Interface.

Arg2: → Sink IUnknown Interface.

Arg3: → STIN Tolknown Interface.

Arg3: → IID of the outgoing interface whose connection point object is being requested (defined by the Source to communicate and implemented by the Sink).

Arg4: → ConnectionPoint interface pointer.

Arg5: → DWORD Cookie.

Return: eax = HRESULT.

Procedure: ComEventsUnadvice

Files: \ObjAsm\Code\ObjMem\64\ComEventsUnadvice.asm

\ObjAsm\Code\ObjMem\Common\ComEventsUnadvice\_X.inc

Notificate the Event source that pISource will NOT recieve Events any more.

Arguments: Arg1: → Previous ConnectionPoint interface.

Arg2: DWORD Cookie received from previous ComEventsAdvice call.

eax = HRESULT.Return:

Procedure: ComGetErrStrA

\ObjAsm\Code\ObjMem\64\ComGetErrStrA.asm Files:

\ObjAsm\Code\ObjMem\Common\ComGetErrStr\_TX.inc

Purpose: Return a description ANSI string from a COM error code.

Arguments: Arg1: COM error code.  $rax \rightarrow Error string.$ Return:

Procedure: ComGetErrStrW

\ObjAsm\Code\ObjMem\64\ComGetErrStrW.asm Files:

\ObjAsm\Code\ObjMem\Common\ComGetErrStr\_TX.inc

Return a description WIDE string from a COM error code. Purpose:

Arguments: Arg1: COM error code. Return: rax → Error string.

Procedure: ComPtrAssign

Files: \ObjAsm\Code\ObjMem\64\ComPtrAssign.asm

\ObjAsm\Code\ObjMem\Common\ComPtrAssign\_X.inc

First increments the reference count of the new interface and then releases any

existing interface pointer.
Arguments: Arg1: → Old Interface pointer.

Arg2: New Interface pointer.

Procedure: CreatePathA

\ObjAsm\Code\ObjMem\64\CreatePathA.asm File: Purpose: Create a path on the destination drive.

Arguments: Arg1:  $\rightarrow$  ANSI path string.

Return: Nothing.

Procedure: CreatePathW

\ObjAsm\Code\ObjMem\64\CreatePathw.asm Purpose: Create a path on the destination drive.

Arguments: Arg1: → WIDE path string.

Nothing. Return:

Procedure: DbgClose

\ObjAsm\Code\ObjMem\64\DbgClose.asm File:

Purpose: Close the connection to the output device.

Arguments: None. Return: Nothing.

Procedure: DbgConOpen

File: \ObjAsm\Code\ObjMem\64\DbgConOpen.asm

Purpose: Open a new console for the calling process.

Arguments: None.

rax = TRUE if it was opened, otherwise FALSE.

Procedure: DbgLogOpen

\ObjAsm\Code\ObjMem\64\DbgLogOpen.asm File:

Purpose: Open a Log-File.

Arguments: None.

Return: rax = TRUE if it was opened, otherwise FALSE.

Procedure: DbgOutApiErr

Files: \ObjAsm\Code\ObjMem\64\DbgOutApiErr.asm

\ObjAsm\Code\ObjMem\Common\DbgOutApiErr\_

Identify a API error with a string. Purpose:

Arguments: Arg1: Api error code obtained with GetLastError.
Arg2: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutBitmap

\ObjAsm\Code\ObjMem\64\DbgOutBitmap.asm File: Purpose: Send a bitmap to the Debug Center Window.

Arguments: Arg1: Bitamp HANDLE.
Arg2: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutCmd

\ObjAsm\Code\ObjMem\64\DbgOutCmd.asm File: Send a command to a specific Debug window. Purpose:

Arguments: Arg1: Command ID [BYTE].
Arg2: → Destination Window WIDE name.

Nothing. Return:

Procedure: DbgOutComErr

Files: \ObjAsm\Code\ObjMem\64\DbgOutComErr.asm

\ObjAsm\Code\ObjMem\Common\DbgOutComErr\_X.inc

Identify a COM error with a string. Purpose:

Arguments: Arg1: COM error ID.

Arg2: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutComponent

\ObjAsm\Code\ObjMem\64\DbgOutComponent.asm File:

Purpose: Identify a COM-Component. Arguments: Arg1: → CSLID.

Arg2: Foreground color.

Arg2: → Destination Window WIDE name.

Procedure: DbgOutComponentName

\ObjAsm\Code\ObjMem\64\DbgOutComponentName.asm File:

Purpose:

Arguments:

Identify a COM-Component.

Arg1: → CSLID.

Arg2: Foreground color.

Arg2: → Destination Window WIDE name.

Return: Nothina.

Procedure: DbgOutFPU

Files: \ObjAsm\Code\ObjMem\64\DbgOutFPU.asm

\ObjAsm\Code\ObjMem\Common\DbgOutFPU\_XP.inc

Display the content of the FPU. Purpose:

Arguments: Arg1: → Destination Window WIDE name. Arg2: Text RGB color.

Return: Nothing.

Procedure: DbgOutFPU\_UEFI

\ObjAsm\Code\ObjMem\64\DbgOutFPU\_UEFI.asm Files: \ObjAsm\Code\ObjMem\Common\DbgOutFPU\_XP.inc

Purpose: Display the content of the FPU.

Arguments: Arg1:  $\rightarrow$  Destination Window WIDE name. Arg2: Text RGB color.

Nothing. Return:

Procedure: DbgOutInterface

\ObjAsm\Code\ObjMem\64\DbgOutInterface.asm File:

Purpose: Identify a COM-Interface.

Arguments: Arg1: → CSLID.

Arg2: Foreground color.

Arg2: → Destination Window WIDE name.

Procedure: DbgOutInterfaceName

\ObjAsm\Code\ObjMem\64\DbgOutInterfaceName.asm File:

Purpose: Identify a COM-Interface. Arguments: Arg1:  $\rightarrow$  CSLID.

Arg2: Foreground color.

Arg2: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutMem

\ObjAsm\Code\ObjMem\64\DbgOutMem.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutMem XP.inc
Output the content of a memory block.

Purpose:

Arguments: Arg1: → Memory block. Arg2: Memory block size.

Arg3: Representation format. Arg4: Memory output color.

Arg5: Representation output color. Arg6: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutMem\_UEFI

\ObjAsm\Code\ObjMem\64\DbgOutMem\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutMem\_XP.inc
Output the content of a memory block.

Purnose:

Arguments:

Arg1: → Memory block. Arg2: Memory block size. Arg3: Representation format. Arg4: Memory output color.

Arg5: Representation output color. Arg6: → Destination Window WIDE name.

Nothing. Return:

Procedure: DbgOutMsg

\ObjAsm\Code\ObjMem\64\DbgOutMsg.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutMsg\_X.inc

Purpose: Identify a windows message with a string.

Arg1: Windows message ID. Arguments:

Arg2: Foreground color. Arg3: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutTextA

\ObjAsm\Code\ObjMem\64\DbgOutTextA.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutText\_TX.inc Send an ANSI string to the debug output device.

Arg1: → Zero terminated ANSI string.

Arg2: Color value.

Arg3: Effect value (DBG\_EFFECT\_XXX). Purpose:

Arguments:

Arg4: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutTextA\_UEFI

\ObjAsm\Code\ObjMem\64\DbgOutTextA\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutText\_TX\_UEFI.inc Send an ANSI string to the debug output device. Purpose:

Arguments: Arg1: → Zero terminated ANSI string.

Arg2: Color value.

Arg3: Effect value (DBG\_EFFECT\_XXX). Arg4: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutTextCA

File: \ObjAsm\Code\ObjMem\64\DbgOutTextCA.asm

Purpose: Send a counted ANSI string to the debug output device.

Arguments: Arg1: → Null terminated WIDE string. Arg2: Maximal character count.

Arg3: Color value.

Arg4: Effect value (DBG\_EFFECT\_XXX). Arg5: → Destination Window WIDE name.

Nothing. Return:

Procedure: DbgOutTextCW

File:

\ObjAsm\Code\ObjMem\64\DbgOutTextCW.asm
Send a counted WIDE string to the debug output device. Purpose:

Arguments: Arg1: → Null terminated WIDE string. Arg2: Maximal character count.

Arg3: Color value.

Arg4: Effect value (DBG\_EFFECT\_XXX). Arg5: → Destination Window WIDE name.

Return: Nothing.

Procedure: DbgOutTextW

\ObjAsm\Code\ObjMem\64\DbgOutTextW.asm Files:

\ObjAsm\Code\ObjMem\Common\DbgOutText\_TX.inc Purnose: Send a WIDE string to the debug output device.

Arguments: Arg1: → Zero terminated WIDE string. Arg2: Color value.

Arg3: Effect value (DBG\_EFFECT\_XXX) Arg4: → Destination window WIDE name.

Return: Nothing.

Procedure: DbgOutTextW\_UEFI

\ObjAsm\Code\ObjMem\64\DbqOutTextW\_UEFI.asm \ObjAsm\Code\ObjMem\Common\DbqOutText\_TX\_UEFI.inc Files:

Purpose: Send a WIDE string to the debug output device.

Arguments: Arg1: → Zero terminated WIDE string.

Arg2: Color value.

Arg3: Effect value (DBG\_EFFECT\_XXX).

Arg4: → Destination window WIDE name.

Return: Nothina.

Procedure: DbgShowObjectHeader

<u>\ObjAsm\Code\ObjMem\64\DbgShowObjectHeader.asm</u> Files:

\ObjAsm\Code\ObjMem\Common\DbgShowObjectHeader\_XP.inc

Output heading object information. Purpose:

Arguments: Arg1: → Object Name. Arg2: → Instance.

Arg3: Text RGB color.

Arg3: → Destination Window WIDE name.

Return: Nothina.

Procedure: DbgShowObjectHeader\_UEFI

Files: \ObjAsm\Code\ObjMem\64\DbgShowObjectHeader\_UEFI.asm

\ObjAsm\Code\ObjMem\Common\DbgShowObjectHeader\_XP.inc

Purpose: Output heading object information.

Arg1: → Object Name. Arg2: → Instance. Arguments:

Arg3: Text RGB color.

Arg3: → Destination Window WIDE name.

Nothing. Return:

Procedure: DbgShowTraceMethod

\ObjAsm\Code\ObjMem\64\DbgShowTraceMethod.asm File:

Purpose: Output trace information about a method.

Arguments: Arg1:  $\rightarrow$  Method Name. Arg2: Method count. Arg3: Method ticks.

Arg4: → Destination Window WIDE name.

Nothing. Return:

Procedure: DbawndOpen

\ObjAsm\Code\ObjMem\64\DbqWndOpen.asm
Open a "Debug Center" instance. File:

Purnose:

Arguments: None.

Return: eax = TRUE if it was opened, otherwise FALSE.

Procedure: dec2dwordA

\ObjAsm\Code\ObjMem\64\dec2dwordA.asm File: Convert a decimal ANSI string to a DWORD. Arguments: Arg1:  $\rightarrow$  Source ANSI string. Possible leading characters are " ", tab, "+" and "-", followed by a sequence of chars between "0".."9" and finalized by a ZTC.

Other characters terminate the convertion returning zero.

eax = Converted DWORD. Return:

rcx = Conversion result. Zero if succeeded, otherwise failed.

Procedure: dec2dwordW

\ObjAsm\Code\ObjMem\64\dec2dwordw.asm File: Purpose: Convert a decimal WIDE string to a DWORD.

Arguments: Arg1: → Source WIDE string. Possible leading characters are " ", tab, "+" and "-", followed by a sequence of chars between "0".."9" and finalized by a ZTC.

Other characters terminate the convertion returning zero.

Return: rax = Converted DWORD.

rcx = Conversion result. Zero if succeeded, otherwise failed.

Procedure: DisableCPUSerialNumber

\ObjAsm\Code\ObjMem\64\DisableCPUSerialNumber.asm File: Purpose: Disable the reading of the CPU serial number.

Arguments: None. Nothing. Return:

Procedure: DllErr2StrA

\ObiAsm\Code\ObiMem\64\DllErr2StrA.asm File:

Purpose: Translate an error code to an ANSI string stored in a DLL.

Arguments: Arg1: Error code.

Arg2: → preallocated ANSI character buffer. Arg3: Buffer size in characters, inclusive ZTC.

Arg4: → DLL ANSI name.

Return: Nothing.

Procedure: DllErr2StrW

\ObjAsm\Code\ObjMem\64\DllErr2StrW.asm File:

Translate an error code to a WIDE string stored in a DLL. Purpose:

Arguments: Arg1: Error code.

Arg2: → WIDE character buffer.

Arg3: Buffer size in characters, inclusive ending terminator.

Arg4: → DLL WIDE name.

Return: Nothing.

Procedure: DrawTransparentBitmap

\ObjAsm\Code\ObjMem\64\DrawTransparentBitmap.asm File: Purpose: Draw a bitmap with transparency on a device context.

Arguments: Arg1: DC HANDLE.

Arg2: Bitmap HANDLE to draw. Arg3; X start position on DC. Arg4: Y start position on DC.

Arg5: RGB transparent color. Use TBM\_FIRSTPIXEL to indicate that the pixel in the

upper left corner contains the transparent color.

Return: Nothing.

Original source by Microsoft. Notes:

"HOWTO: Drawing Transparent Bitmaps (Q79212)"

(http://support.microsoft.com/default.aspx?scid=kb;EN-US;q79212)

Transcribed by Ernest Murphy.

Procedure: dword2binA

File:

\ObjAsm\Code\ObjMem\64\dword2binA.asm
Convert a DWORD to its binary ANSI string representation. Purpose:

Arguments: Arg1: → Destination buffer. Arg2: DWORD value.

Return: Nothing.

The destination buffer must be at least 33 BYTEs large to allocate the output string Notes:

(32 character BYTEs + ZTC = 33 BYTEs).

Procedure: dword2binW

\ObjAsm\Code\ObjMem\64\dword2binW.asm File:

Purpose: Convert a DWORD to its binary WIDE string representation.

Arguments: Arg1: → Destination buffer.

Arg2: DWORD value.

Return: Nothing.

Notes: The destination buffer must be at least 66 BYTEs large to allocate the output string

(32 character WORDS + ZTC = 66 BYTES).

Procedure: dword2hexA

File:

\ObjAsm\Code\ObjMem\64\dword2hexA.asm
Convert a DWORD to its hexadecimal ANSI string representation. Purpose:

Arguments: Arg1: → Destination ANSI string buffer. Arg2: DWORD value.

Nothing. Return:

Notes: The destination buffer must be at least 9 BYTEs large to allocate the output string

(8 character BYTES + ZTC = 9 BYTES).

Procedure: dword2hexW

File: \ObjAsm\Code\ObjMem\64\dword2hexw.asm

Purpose: Convert a DWORD to its hexadecimal WIDE string representation.

Arguments: Arg1:  $\rightarrow$  Destination WIDE string buffer.

Arg2: DWORD value.

Nothing.

The destination buffer must be at least 18 BYTEs large to allocate the output string Notes:

(8 character WORDS + ZTC = 18 BYTES).

Procedure: EHandler

\ObjAsm\Code\ObjMem\64\EHandler.asm
ASM exception handler File:

Purpose: Arguments: Arg1:  $\rightarrow$  Exception Record.  $Arg2: \rightarrow Establisher Frame.$ Arg3: → ContextRecord

Arg4: → DispatcherContext https://docs.microsoft.com/en-us/cpp/build/language-specific-handler Link:

http://www.nynaeve.net/?p=113

rax = ExceptionContinueSearch. Return:

Procedure: Err2StrA

File: \ObjAsm\Code\ObjMem\64\Err2StrA.asm

Purpose: Translate a system error code to an ANSI string. Arguments: Arg1: Error code.

Arg2: → ANSI string buffer. Arg3: Buffer size in characters, inclusive ZTC.

Return: Nothing.

Procedure: Err2StrW

\ObjAsm\Code\ObjMem\64\Err2StrW.asm

Translate a system error code to a WIDE string.

Arguments: Arg1: Error code.

Arg2: → WIDE string buffer. Arg3: Buffer size in characters, inclusive ZTC.

Return: Nothing.

Procedure: ErrorMessageBoxA

\ObjAsm\Code\ObjMem\64\ErrorMessageBoxA.asm File:

Show a messagebox containing an error string in the locale lenguage and an user Purpose:

string.

Arguments: Arg1: Messagebox parent window HANDLE. Arg2: → User ANSI string.

Arg3: Locale ID. Arg4: API error code.

Return: Nothing.

Procedure: ErrorMessageBoxW

\ObjAsm\Code\ObjMem\64\ErrorMessageBoxW.asm File:

Purpose: Show a messagebox containing an error string in the locale lenguage and an user

string.

Arguments: Arg1: Messagebox parent window HANDLE.

Arg2: → User WIDE string. Arg3: Locale ID. Arg4: API error code.

Nothing. Return:

Procedure: FileExistA

\ObjAsm\Code\ObjMem\64\FileExistA.asm File:

Check the existence of a file. Purnose:

Arguments: Arg1: → ANSI file name.

Return: rax = TRUE if the file exists, otherwise FALSE.

Procedure: FileExistW

\ObjAsm\Code\ObjMem\64\FileExistW.asm File:

Check the existence of a file.

Arguments: Arg1: → WIDE file name.

rax = TRUE if the file exists, otherwise FALSE. Return:

Procedure: FindFileA

\ObjAsm\Code\ObjMem\64\FindFileA.asm Files:

\ObjAsm\Code\ObjMem\Common\FindFile\_TX.inc Purpose: Search for a file in a list of paths. Arguments: Arg1: → File name.

Arg2: → List of path strings. The end of the list is indicated with a ZTC.

Arg3: → Buffer containing the full path and file name in which the file was found.

Buffer length = MAX\_PATH.
eax = Number of chars copied to the destination buffer. 0 if the file was not found.
Example: invoke FindFile, \$0fsCStr("free.inc"), \$0fsCStr("\Here\*",0), addr cBuf Return:

Search free.inc in all \Here and suddirectories.

Procedure: FindFileW

Files:

\ObjAsm\Code\ObjMem\64\FindFileW.asm \ObjAsm\Code\ObjMem\Common\FindFile\_TX.inc

Purpose: Search for a file in a list of paths.

Arguments: Arg1: → File name.

Arg2: → List of path strings. The end of the list is indicated with a ZTC.

Arg3: → Buffer containing the full path and file name in which the file was found.

Buffer length = MAX\_PATH.

eax = Number of chars copied to the destination buffer. 0 if the file was not found.

Example: invoke FindFile, \$0fsCStr("free.inc"), \$0fsCStr("\Here\*",0), addr cBuf Return:

Search free.inc in all \Here and suddirectories.

Procedure: FindModuleByAddrA

File:

\ObjAsm\Code\ObjMem\64\FindModuleByAddrA.asm
Find the module name from an address on a WinNT system. Purpose:

Arguments: Arg1: Address.
Arg2: → ANSI module name buffer.

eax = Number of characters copied into the buffer. Return:

Procedure: FindModuleByAddrW

File:

\ObjAsm\Code\ObjMem\64\FindModuleByAddrw.asm
Find the module name from an address on a WinNT system. Purpose:

Arguments: Arg1: Address.

Arg2: → WIDE module name buffer.

eax = Number of characters copied into the buffer. Return:

Procedure: GetAncestorID

Files: \ObjAsm\Code\ObjMem\64\GetAncestorID.asm

\ObjAsm\Code\ObjMem\Common\GetAncestorID\_XP.inc

Retrieve the ancestor type ID of an object type ID. Purpose:

Arguments: Arg1: → Object class ID.

Return: eax = Ancestor type ID or zero if not found.

Procedure: GetBottomWindow

\ObjAsm\Code\ObjMem\64\GetBottomWindow.asm Purpose: Get the Z order bottom child window HANDLE.

Arguments: Arg1: Parent HWND.
Return: eax = Z order bottom child window HANDLE.

Procedure: GetDlgBaseUnits

Purpose:

Files:

\ObjAsm\Code\ObjMem\64\GetDlqBaseUnits.asm \ObjAsm\Code\ObjMem\Common\GetDlqBaseUnits\_X.inc

Return the Dialog Base Units. Arguments: Arg1: Dialog DC.

eax = X DBU. ecx = Y DBU. Return:

Procedure: GetExceptionStrA

File: Purpose: \ObjAsm\Code\ObjMem\64\GetExceptionStrA.asm Translate an exception code to an ANSI string.

Arguments: Arg1: Exception code.  $rax \rightarrow ANSI string.$ Return:

Procedure: GetExceptionStrW

\ObjAsm\Code\ObjMem\64\GetExceptionStrW.asm Purpose: Translate an exception code to a WIDE string.

Arguments: Arg1: Exception code.

Return:  $rax \rightarrow WIDE string.$ 

Procedure: GetFileHashA

Purpose:

Arguments: Arg1: → Hash return value Arg2: → ANSI file name. Arg3: Hash type. Return: eax = 0 if succeeded.

http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297 Links:

Notes:

Original translation from MSDN library by Edgar Hansen It requires a fully qualified path to a file to generate a hash for and a pointer to a WIDE string buffer to hold the resulting hash in HEX (16 BYTES for MDX, 20 BYTES

for SHAx) and an algorithm ID, for MD5 set dHashType to GFH\_MD5.

See ObjMem.inc GFH\_xxx.

Procedure: GetFileHashW

File: \ObjAsm\Code\ObjMem\64\GetFileHashW.asm

Purpose: Compute the hash value from the content of a file.

Arguments: Arg1: → Hash return value

Arg2: → WIDE file name.

Arg3: Hash type.

Return: eax = 0 if succeeded.

http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297 Links:

Notes:

Original translation from MSDN library by Edgar Hansen It requires a fully qualified path to a file to generate a hash for and a pointer to a WIDE string buffer to hold the resulting hash in HEX (16 BYTES for MDX, 20 BYTES

for SHAx) and an algorithm ID, for MD5 set dHashType to GFH\_MD5.

See ObjMem.inc GFH\_xxx.

Procedure: GetFileLinesA

\ObjAsm\Code\ObjMem\64\GetFileLinesA.asm Files:

\ObjAsm\Code\ObjMem\Common\GetFileLines\_AX.inc
Return an array of line ending offsets of an ANSI text file. Purnose:

Arguments: Arg1: File HANDLE. Return:

eax = Number of lines.  $rcx \rightarrow Mem block containing an array of DWORD offsets.$ 

The user must dispose it using MemFree.

- Lines must be terminated with the ANSI char sequence 13, 10 (CRLF). Notes:

- The last line may not terminate with a CRLF.

Procedure: GetLogProcCount

File: \ObjAsm\Code\ObjMem\64\GetLogProcCount.asm

Return the number of logical CPUs on the current system. Purpose:

Arguments: None

Return: eax = Number of logical processors.

Procedure: GetObjectID

\ObjAsm\Code\ObjMem\64\GetObjectID.asm Retrieve the type ID of an object instance. Purnose:

Arguments: Arg1: → Object instance. Return: eax = Object class ID.

Procedure: GetObjectTemplate

Files:

\ObjAsm\Code\ObjMem\64\GetObjectTemplate.asm \ObjAsm\Code\ObjMem\Common\GetObjectTemplate\_XP.inc

Purnose: Get the template address of an object type ID.

Arguments: Arg1: Object type ID.

Return:  $rax \rightarrow Object template or NULL if not found.$ ecx = Object template size or zero if not found.

Procedure: GetPrevInstanceA

File:

\ObjAsm\Code\ObjMem\64\GetPrevInstanceA.asm
Return a HANDLE to a previously running instance of an application. Purpose:

Arguments: Arg1:  $\rightarrow$  ANSI application name. Arg2:  $\rightarrow$  ANSI class name.

rax = Window HANDLE of the application instance or zero if failed. Return:

Procedure: GetPrevInstanceW

\ObjAsm\Code\ObjMem\64\GetPrevInstanceW.asm File:

Return a handle to a previously running instance of an application. Purpose:

Arguments: Arg1: → WIDE application name.

Arg2: → WIDE class name.

Return: rax = Window HANDLE of the application instance or zero if failed.

Procedure: GetRawClientRect

Files: \ObjAsm\Code\ObjMem\64\GetRawClientRect.asm

\ObjAsm\Code\ObjMem\Common\GetRawClientRect\_X.inc
Calculate the window client RECT including scrollbars, but without the room needed Purnose:

for the menubar

Arguments: Arg1: Window HANDLE Arg2:  $\rightarrow$  RECT.

Nothing. Return:

Procedure: GUID2BStr

\ObjAsm\Code\ObjMem\64\GUID2BStr.asm

Purpose: Convert a GUID to a BStr.

Arguments: Arg1: → Destination BStr Buffer. It must hold at least

36 characters plus a terminating zero.

Arg2:  $\rightarrow$  GUID.

Nothing. Return:

Procedure: GUID2StrA

Files: \ObjAsm\Code\ObjMem\64\GUID2StrA.asm

\ObjAsm\Code\ObjMem\Common\GUID2Str\_AXP.inc

Purpose: Convert a GUID to an ANSI string.

Arguments: Arg1: → Destination ANSI string buffer.

It must hold at least 36 characters plus a ZTC (= 37 BYTEs).

Arg2: → GUID.

Nothing. Return:

Procedure: GUID2StrW

Files: \ObjAsm\Code\ObjMem\64\GUID2StrW.asm

\ObjAsm\Code\ObjMem\Common\GUID2Str\_WXP.inc

Convert a GUID to a WIDE string. Purnose:

Arguments: Arg1:  $\rightarrow$  Destination WIDE string Buffer.

It must hold at least 36 characters plus a ZTC (= 74 BYTEs).

Arg2:  $\rightarrow$  GUID.

Return: Nothing.

Procedure: hex2dwordA

File: \ObjAsm\Code\ObjMem\64\hex2dwordA.asm

Purpose: Load an ANSI string hexadecimal representation of a DWORD.

Arguments: Arg1: → ANSI hexadecimal string with 8 characters.

Return: eax = DWORD.

Return:

Procedure: hex2dwordW

\ObjAsm\Code\ObjMem\64\hex2dwordW.asm

Purpose: Load a WIDE string hexadecimal representation of a DWORD.

Arguments: Arg1: → WIDE hex string with 8 characters.

eax = DWORD.Return:

Procedure: hex2qwordA

File: \( \lambda \text{ObjAsm\Code\ObjMem\64\hex2qwordA.asm} \)
Purpose: Load an ANSI string hexadecimal representation of a QWORD.
Arguments: Arg1: → ANSI hexadecimal string with 16 characters.

rax = QWORD.Return:

Procedure: hex2qwordW

File:

\lambda \text{ObjAsm\Code\ObjMem\64\hex2qwordW.asm} \text{Load a WIDE string hexadecimal representation of a QWORD.} Purpose:

Arguments: Arg1: → WIDE hexadecimal string with 16 characters.

rax = QWORD.Return:

Procedure: IsAdmin

File:

\ObjAsm\Code\ObjMem\64\IsAdmin.asm Check if the current user has administrator rights. Purpose:

Arguments: None.

Return: rax = TRUE or FALSE.

Procedure: IsGUIDEqual

File: \ObjAsm\Code\ObjMem\64\IsGUIDEqual.asm

Compare 2 GUIDs.

Arguments: Arg1: → GUID1

Arg2: → GUID2

rax = TRUE if they are equal, otherwise FALSE.

Procedure: IsHardwareFeaturePresent

\ObjAsm\Code\ObjMem\64\IsHardwareFeaturePresent.asm
Check if a CPU hardware feature is present on the system. File: Purpose:

Arguments: Arg1: CPUID feature ID. rax = TRUE or FALSE.Return:

Procedure: IsPntInRect

\ObjAsm\Code\ObjMem\64\IsPntInRect.asm File: Purpose: Check if a point is within a rect.

Arguments: Arg1: → POINT. Arg2: → RECT

rax = TRUE or FALSE.Return:

Procedure: IsProcessElevated

Files: \ObjAsm\Code\ObjMem\64\IsProcessElevated.asm

\ObjAsm\Code\ObjMem\Common\IsProcessElevated\_X.inc

Purpose: Check if the current process has elevated privileges.

Arguments: Arg: Process HANDLE. Return: eax = TRUE or FALSE

Example: invoke GetCurrentProcess invoke IsProcessElevated, xax

Procedure: IsScrollBarVisible

\ObjAsm\Code\ObjMem\64\IsScrollBarVisible.asm
Determine if a Scrollbar is currently visible. File: Purpose:

Arguments: Arg1: Main window HANDLE that the scrollbar belongs to.
Arg2: Scrollbar type [SB\_HORZ or SB\_VERT].

eax = TRUE if the scrollbar is visible, otherwise FALSE. Return:

Procedure: IsWinNT

\ObjAsm\Code\ObjMem\64\IsWinNT.asm File: Purpose: Detect if the OS is Windows NT based.

Arguments: None.

rax = TRUE if OS is Windows NT based, otherwise FALSE. Return:

Procedure: LoadCommonControls

File:

\ObjAsm\Code\ObjMem\64\LoadCommonControls.asm
Invoke InitCommonControls with a correctly filled input structure. Purpose:

Arguments: Arg1: ICC\_COOL\_CLASSES, ICC\_BAR\_CLASSES, ICC\_LISTVIEW\_CLASSES, ICC\_TAB\_CLASSES,

ICC\_USEREX\_CLASSES, etc.

Return: Nothing.

Procedure: Mem2HexA

\ObjAsm\Code\ObjMem\64\Mem2HexA.asm
Convert the memory content into a hex ANSI string representation. Purnose:

Arguments: Arg1: → ANSI character buffer.

Arg2: → Source memory.

Arg3: Byte count.

Nothing. Return:

Procedure: Mem2HexW

File:

\ObjAsm\Code\ObjMem\64\Mem2HexW.asm
Convert the memory content into a hex WIDE string representation. Purpose:

Arguments: Arg1: → WIDE character buffer.

Arg2: → Source memory. Arg3: Byte count.

Nothing. Return:

Procedure: MemAlloc\_UEFI

Files:

\ObjAsm\Code\ObjMem\64\MemAlloc\_UEFI.asm \ObjAsm\Code\ObjMem\Common\MemAlloc\_X\_UEFI.inc

Purpose: Allocate a memory block.
Arguments: Arg1: Memory block attributes [0, MEM\_INIT\_ZERO].

Arg2: Memory block size in BYTEs.

Return: rax → Memory block or NULL if failed.

Procedure: MemClone

File:

\ObjAsm\Code\ObjMem\64\MemClone.asm
Copy a memory block from a source to a destination buffer. Purpose:

Source and destination must NOT overlap.

Destination buffer must be at least as large as number of BYTEs to copy, otherwise a

fault may be triggered.

Arguments: Arg1: → Destination buffer.
Arg2: → Source buffer.
Arg3: Number of BYTEs to copy.
Return: eax = Number of copied BYTEs.

MemComp Procedure:

\ObjAsm\Code\ObjMem\64\MemComp.asm File:

Purpose: Compare 2 memory blocks.

Both memory blocks must be at least as large as the maximal number of BYTEs to

otherwise a fault may be triggered.

Arguments: Arg1: → Memory block 1.

Arg2: → Memory block 2.

Arg3: Maximal number of BYTEs to compare. If MemBlock1 = MemBlock2, then eax <> 0.

If MemBlock1 == MemBlock2, then eax = 0. Return:

Procedure: MemFillB

\ObjAsm\Code\ObjMem\64\MemFillB.asm File:

Purpose: Fill a memory block with a given byte value.

Destination buffer must be at least as large as number of BYTEs to fill, otherwise a

fault may be triggered.

Arguments: Arg1: → Destination memory block.

Arg2: Memory block size in BYTEs.

Arg3: Byte value to fill.

Nothing. Return:

Procedure: MemFillW

\ObjAsm\Code\ObjMem\64\MemFillw.asm File:

Purpose: Fill a memory block with a given word value.

Destination buffer must be at least as large as number of BYTEs to fill, otherwise a

fault may be triggered.

Arguments: Arg1: → Destination memory block.

Arg2: Memory block size in BYTEs. Arg3: Word value to fill with.

Return: Nothing.

Procedure: MemFree\_UEFI

Files:

\ObjAsm\Code\ObjMem\64\MemFree\_UEFI.asm \ObjAsm\Code\ObjMem\Common\MemFree\_X\_UEFI.inc

Purpose: Dispose a memory block. Arguments: Arg1: → Memory block.

Return: rax = EFI\_SUCCESS or an UEFI error code.

Procedure: MemReAlloc\_UEFI

\ObjAsm\Code\ObjMem\64\MemReAlloc\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\MemReAlloc\_X\_UEFI.inc

Purpose: Shrink or expand a memory block.

Arguments:

Arg1: → Memory block
Arg2: Memory block size in BYTEs.
Arg3: New memory block size in BYTEs.

Arg4: Memory block attributes [0, MEM\_INIT\_ZERO].

 $rax \rightarrow New memory block.$ Return:

Procedure: MemShift

File: \ObjAsm\Code\ObjMem\64\MemShift.asm

Copy a memory block from a source to a destination buffer. Purpose:

Source and destination may overlap.

Destination buffer must be at least as large as number of BYTEs to shift, otherwise a

fault may be triggered.

Arguments: Arg1: → Destination buffer.
Arg2: → Source buffer.
Arg3: Number of BYTEs to shift.
Return: rax = Number of BYTEs shifted.

MemSwap Procedure:

\ObjAsm\Code\ObjMem\64\MemSwap.asm File:

Purpose: Exchange the memory content from a memory buffer to another.

They must NOT overlap.

Both buffers must be at least as large as number of BYTEs to exchange, otherwise a

fault may be triggered.

Arg1: → Memory buffer 1. Arg2: → Memory buffer 2. Arg3: Number of BYTEs to exchange. Arguments:

Return: Nothing.

Procedure: MemZero

File:

\ObjAsm\Code\ObjMem\64\MemZero.asm

Fill a memory block with zeros. A bit faster than MemFillB. Purpose:

The memory buffer must be at least as large as number of BYTEs to zero, otherwise a

fault may be triggered.

Arguments: Arg1: → Memory buffer.

Arg2: Number of BYTEs to zero.

Nothing. Return:

Procedure: MoveWindowVisible

\ObjAsm\Code\ObjMem\64\MoveWindowVisible.asm File:

Purnose: On a multimonitor system, move a window but remains always in the visible region.

Arguments: Arg1: HANDLE of the Window to move.

Arg2: Target X position in pixel. Arg3: Target Y position in pixel.

Return: Nothing.

Procedure:

\ObjAsm\Code\ObjMem\64\MsgBoxA.asm

\ObjAsm\Code\ObjMem\Common\MsqBox\_TX.inc

Show a customized MessageBox. Purpose:

Arguments: Arg1: Parent HANDLE.
Arg2: → Markup text.

Arg3: → Caption text.

Arq4: Flags.

eax = Zero if failed, otherwise pressed button ID. Return:

Caption, text etc. are transferred via a caption string which contains a header and Note:

the address of a MsgBoxInfo structure in text form.

Procedure:

\ObjAsm\Code\ObjMem\64\MsgBoxW.asm Files:

\ObjAsm\Code\ObjMem\Common\MsgBox\_TX.inc

Purpose: Show a customized MessageBox.

Arguments: Arg1: Parent HANDLE. Arg2: → Markup text. Arg3: → Caption text.

Arg4: Flags.

eax = Zero if failed, otherwise pressed button ID. Return:

Note: Caption, text etc. are transferred via a caption string which contains a header and

the address of a MsgBoxInfo structure in text form.

Procedure: NetErr2StrA

\ObjAsm\Code\ObjMem\64\NetErr2StrA.asm

Purnose: Translate a network error code to an ANSI string.

Arguments: Arg1: Error code.

Arg2: → ANSI character buffer.

Arg3: Buffer size in characters, inclusive ZTC. eax = Number CHRA stored in the output buffer, excluding the ZTC. Return:

Procedure: NetErr2StrW

File: \ObjAsm\Code\ObjMem\64\NetErr2Strw.asm

Purpose: Translate a network error code to a WIDE string.

Arguments: Arg1: Error code.

Arg2: → WIDE string buffer.

Arg3: Buffer size in characters, inclusive ZTC.

eax = Number CHRW stored in the output buffer, excluding the ZTC. Return:

Procedure: NewObjInst

\ObjAsm\Code\ObjMem\64\NewObjInst.asm \ObjAsm\Code\ObjMem\Common\NewObjInst\_XP.inc Files:

Create an object instance from an object ID. Purpose:

Arguments: Arg1: Object ID.

Return:  $rax \rightarrow New$  object instance or NULL if failed.

Procedure: NewObjInst\_UEFI

\ObjAsm\Code\ObjMem\64\NewObjInst\_UEFI.asm

\ObjAsm\Code\ObjMem\Common\NewObjInst\_XP.inc

Create an object instance from an object ID. Purpose:

Arguments: Arg1: Object ID.

 $rax \rightarrow New$  object instance or NULL if failed.

Procedure: ParseA

File:

\ObjAsm\Code\ObjMem\64\ParseA.asm
Extract a comma separated substring from a source string. Purpose:

Arguments: Arg1: → Destination buffer. Must be large enough to hold the ANSI substring.

Arg2: → Source ANSI string.

Arg3: Zero based index of the requested ANSI substring.

Return: eax = 1: success.

2: insufficient number of components. 3: non matching quotation marks.

4: empty quote.

Procedure: ParseW

File: \ObjAsm\Code\ObjMem\64\Parsew.asm

Purnose:

Extract a comma separated substring from a source string.

Arg1: → Destination buffer. Must be large enough to hold the WIDE substring.

Arg2: → Source WIDE string. Arguments:

Arg3: Zero based index of the requested WIDE substring.

Return: eax = 1: success.

2: insufficient number of components. 3: non matching quotation marks.

4: empty quote.

Procedure: PdfViewA

Files: \ObjAsm\Code\ObjMem\64\PdfViewA.asm

\ObjAsm\Code\ObjMem\Common\PdfView\_TX.inc

Display a PDF document on a named destination. Purpose:

Arguments: Arg1: Parent HANDLE. Arg2:  $\rightarrow$  PDF document. Arg3:  $\rightarrow$  Destination.

rax = HINSTANCE. See ShellExecute return values. Return:

A value greater than 32 indicates success.

Procedure: PdfViewW

Purpose:

\ObjAsm\Code\ObjMem\64\PdfViewW.asm Files:

\ObjAsm\Code\ObjMem\Common\PdfView\_TX.inc Display a PDF document on a named destination.

Arguments: Arg1: Parent HANDLE. Arg2:  $\rightarrow$  PDF document. Arg3:  $\rightarrow$  Destination.

rax = HINSTANCE. See ShellExecute return values.
A value greater than 32 indicates success. Return:

Procedure: qword2binA

File: \ObjAsm\Code\ObjMem\64\qword2binA.asm

Convert a QWORD to its binary ANSI string representation.

Arguments: Arg1: → Destination buffer.

Arg2: QWORD value.

Return: Nothing.

The destination buffer must be at least 65 BYTEs large to allocate the output string Notes:

(64 character BYTES + ZTC = 65 BYTES).

Procedure: gword2binW

File: \ObjAsm\Code\ObjMem\64\qword2binW.asm

Convert a QWORD to its binary WIDE string representation. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: QWORD value.

Nothing. Return:

The destination buffer must be at least 130 BYTEs large to allocate the output string Notes:

(64 character WORDS + ZTC = 130 BYTES).

Procedure: gword2hexA

File:

\ObjAsm\Code\ObjMem\64\qword2hexA.asm
Convert a QWORD to its hexadecimal ANSI string representation. Purpose:

Arguments: Arg1: → Destination ANSI string buffer.

Arg2: QWORD value.

Return: Nothing.

The destination buffer must be at least 17 BYTEs large to allocate the output string Note:

(16 character BYTES + ZTC = 17 BYTES).

Procedure: qword2hexW

\ObjAsm\Code\ObjMem\64\qword2hexw.asm File:

Convert a QWORD to its hexadecimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination WIDE string buffer.
Arg2: QWORD value.

Nothing. Return:

The destination buffer must be at least 34 BYTEs large to allocate the output string Notes:

(16 character WORDS + ZTC = 34 BYTES).

Procedure: RadixSortF32

\ObjAsm\Code\ObjMem\64\RadixSortF32.asm Files. \ObjAsm\Code\ObjMem\Common\RadixSort64.inc

Ascending sort of an array of single precision floats (REAL4) using a modified "4 passes radix sort" algorithm.

Arg1: → Array of single precision floats (REAL4). Arguments:

Arg2: Number of single precision floats contained in the array.

Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

Return:

Notes:

eax = TRUE if succeeded, otherwise FALSE.

- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included.

http://www.codercorner.com/RadixSortRevisited.htmhttp://en.wikipedia.org/wiki/Radix\_sort Tinks:

Procedure: RadixSortF64

\ObjAsm\Code\ObjMem\64\RadixSortF64.asm Files: \ObjAsm\Code\ObjMem\Common\RadixSort64.inc

Ascending sort of an array of double precision floats (REAL8) using a modified "8 passes radix sort" algorithm. Purpose:

Arguments:

Arg1: → Array of double precision floats (REAL8).
Arg2: Number of double precision floats contained in the array.

Arg3:  $\rightarrow$  Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

Return: eax = TRUE if succeeded, otherwise FALSE.

- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included. Notes:

- http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortI32

Links:

Links:

\ObjAsm\Code\ObjMem\64\RadixSortI32.asm Files:

\ObjAsm\Code\ObjMem\Common\RadixSort64.inc

Ascending sort of an array of SDWORDs using a modified "4 passes radix sort" Purpose:

algorithm.

Arg1:  $\rightarrow$  Array of SDWORDs. Arguments:

Arg2: Number of SDWORDs contained in the array.

Arg3:  $\rightarrow$  Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically. eax = TRUE if succeeded, otherwise FALSE.

Return:

Original code from r22. Notes:

http://www.asmcommunity.net/board/index.php?topic=24563.0

- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc

must be modified and stack probing must be included.
- http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrF32

\ObjAsm\Code\ObjMem\64\RadixSortPtrF32.asm Files: \ObjAsm\Code\ObjMem\Common\RadixSort64.inc

Purpose:

Ascending sort of an array of POINTERs to structures containing a single precision float (REAL4) key using a modified "4 passes radix sort" algorithm. Arg1: → Array of POINTERs.

Arguments:

Arg2: Number of POINTERs contained in the array.

Arg3: offset of the REAL4 key within the hosting structure.

Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

Return:

eax = TRUE if succeeded, otherwise FALSE.

For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc Notes:

must be modified and stack probing must be included.

Links: http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrF64

\ObjAsm\Code\ObjMem\64\RadixSortPtrF64.asm Files:

\ObjAsm\Code\ObjMem\Common\RadixSort64.inc

Purpose:

Arguments:

Ascending sort of an array of POINTERs to structures containing a double precision float (REAL8) key using a modified "8 passes radix sort" algorithm.

Arg1: → Array of POINTERs.

Arg2: Number of POINTERs contained in the array.

Arg3: offset of the REAL8 key within the hosting structure.

Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

eax = TRUE if succeeded. otherwise FALSE.

Return:

eax = TRUE if succeeded, otherwise FALSE.
- For short arrays, the shadow array can be placed onto the stack, saving the
expensive memory allocation/deallocation API calls. To achieve this, the proc Notes:

must be modified and stack probing must be included. - http://www.codercorner.com/RadixSortRevisited.htm

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrI32

Links:

Links.

\ObjAsm\Code\ObjMem\64\RadixSortPtrI32.asm \ObjAsm\Code\ObjMem\Common\RadixSort64.inc Files:

Ascending sort of an array of POINTERs to structures containing a SDWORD key using a modified "4 passes radix sort" algorithm. Purpose:

 $Arg1: \rightarrow Array of POINTERs.$ Arguments:

Arg2: Number of POINTERs contained in the array. Arg3: offset of the SDWORD key within the hosting structure.

 $Arg4: \rightarrow Memory$  used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically.

eax = TRUE if succeeded, otherwise FALSE.

- Original code from r22. Return:

Notes:

http://www.asmcommunity.net/board/index.php?topic=24563.0

- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included.

- http://www.codercorner.com/RadixSortRevisited.htm - http://en.wikipedia.org/wiki/Radix\_sort

Procedure: RadixSortPtrUI32

Files: \ObjAsm\Code\ObjMem\64\RadixSortPtrUI32.asm \ObjAsm\Code\ObjMem\Common\RadixSort64.inc

Ascending sort of a POINTER array to structures containing a DWORD key using the "4 passes radix sort" algorithm. Purpose:

Arguments:

Arg1: → Array of POINTERS.
Arg2: Number of POINTERS contained in the array.

Arg3: offset of the DWORD key within the hosting structure.

Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically. eax = TRUE if succeeded, otherwise FALSE.

Return:

Notes: - Original code from r22.

http://www.asmcommunity.net/board/index.php?topic=24563.0

- For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation API calls. To achieve this, the proc must be modified and stack probing must be included.

- http://www.codercorner.com/RadixSortRevisited.htm - http://en.wikipedia.org/wiki/Radix\_sort Links:

Procedure: RadixSortUT32

\ObjAsm\Code\ObjMem\64\RadixSortUI32.asm Files: \ObjAsm\Code\ObjMem\Common\RadixSort64.inc

Ascending sort of an array of DWORDs using the "4 passes radix sort" algorithm. Purpose:

Arg1: → Array of DWORDs. Arguments:

Arg2: Number of DWORDs contained in the array.

Arg3:  $\rightarrow$  Memory used for the sorting process or NULL. The buffer size must be at least the size of the input array. If NULL, a memory chunk is allocated automatically. eax = TRUE if succeeded, otherwise FALSE.

Return:

Original code from r22. Notes:

http://www.asmcommunity.net/board/index.php?topic=24563.0 - For short arrays, the shadow array can be placed onto the stack, saving the expensive memory allocation/deallocation\_API calls. To achieve this, the proc

must be modified and stack probing must be included. - http://www.codercorner.com/RadixSortRevisited.htm Links:

- http://en.wikipedia.org/wiki/Radix\_sort

Procedure: Random32

Files: \ObjAsm\Code\ObjMem\64\Random32.asm

\ObjAsm\Code\ObjMem\Common\Random32\_X.inc

Purpose: Generate a random 32 bit number in a given range [0..Limit-1]. Park Miller random number algorithm. Written by Jaymeson Trudgen (NaN) and optimized

by Rickey Bowers Jr. (bitRAKE). Arg1: Range limit (max. = 07FFFFFFFh).

eax = Random number in the range [0..Limit-1]. Return:

Procedure: Real4ToHalf

<u>\ObjAsm\Code\ObjMem\64\Real4ToHalf.asm</u> Files:

\ObjAsm\Code\ObjMem\Common\Real4ToHalf\_XP.inc

Convert a REAL4 to an HALF. Purpose:

Arguments: Arg1: REAL4 value.

Return: ax = HALF.

Note: alternative code using VCVTPS2PH:

movss xmm0, r4Value VCVTPS2PH xmm1, xmm0, 0

movd eax, xmm1

Procedure: RGB24To16ColorIndex

\ObjAsm\Code\ObjMem\64\RGB24To16ColorIndex.asm Files:

\lambda \text{ObjAsm\Code\ObjMem\Common\RGB24To16ColorIndex XP.inc} \text{Map a 24 bit RGB color to a 16 color palette index.} Purnose:

Arguments: Arg1: RGB color. Return: eax = Palette index.

Procedure: sdword2decA

Files: \ObjAsm\Code\ObjMem\64\sdword2decA.asm

\ObjAsm\Code\ObjMem\Common\sdword2decT64.inc

Convert a signed DWORD to its decimal ANSI string representation. Purpose:

Arguments: Arg1: → Destination ANSI string buffer. Arg2: SDWORD value.

Return:

eax = Number of BYTEs copied to the destination buffer, including the ZTC. The destination buffer must be at least 12 BYTEs large to allocate the output string (Sign + 10 ANSI characters + ZTC = 12 BYTEs). Note:

Procedure: sdword2decW

\ObjAsm\Code\ObjMem\64\sdword2decW.asm Files:

\ObjAsm\Code\ObjMem\Common\sdword2decT64.inc

Purpose: Convert a signed DWORD to its decimal WIDE string representation.

Arguments: Arg1: → Destination WIDE string buffer.

Arg2: SDWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 24 BYTEs large to allocate the output string (Sign + 10 WIDE characters + ZTC = 24 BYTEs). Note:

Procedure: SendChildrenMessage

\ObjAsm\Code\ObjMem\64\SendChildrenMessage.asm Files:

\ObjAsm\Code\ObjMem\Common\SendChildrenMessage\_X.inc

Purpose: Callback procedure for EnumChildWindows that sends a message to a child window.

Arguments: Arg1: Child window HANDLE.

Arg2: → CHILD\_MSG structure.

eax = always TRUE (continue the enumeration).

Procedure: SetClientSize

Files:

\ObjAsm\Code\ObjMem\64\SetClientSize.asm \ObjAsm\Code\ObjMem\Common\SetClientSize\_X.inc

Set the client window size. Purpose: Arguments: Arg1: Target window handle.

Arg2: Client area width in pixel. Arg3: Client area height in pixel.

Return: Nothing.

Procedure: SetPrivilegeTokenA

\ObjAsm\Code\ObjMem\64\SetPrivilegeTokenA.asm

Enable privilege tokens. Purpose: Arguments: Arg1: Process handle.
Arg2: → Privilege name (ANSI string).

Arg3: Eanble = TRUE, disable = FALSE. eax = Zero if failed.

Return:

Procedure: SetPrivilegeTokenW

\ObjAsm\Code\ObjMem\64\SetPrivilegeTokenW.asm

Purpose: Enable privilege tokens. Arguments: Arg1: Process handle.

Arg2: → Privilege name (ANSI string).

Arg3: Eanble = TRUE, disable = FALSE

Return: eax = Zero if failed.

Procedure: SetShellAssociationA

\ObjAsm\Code\ObjMem\64\SetShellAssociationA.asm Files:

\ObjAsm\Code\ObjMem\Common\SetShellAssociation\_TX.inc

Purpose:

Set association for a file extension.

Arg1: TRUE = system wide association, FALSE = user account only. Arguments:

Arg2: → File extension (without dot).

Arg3: → Verb ("open", "print", "play", "edit", etc.). This verb is displayed in the explorer context menu of a file with this extension.

Arg4: → Application to associate with (full path)

 $Arg5: \rightarrow Application arguments, usually $0fsCStr("%1").$ 

Return: eax = HRESULT.

dGlobal = TRUE requires adminitrative rights. Note:

Procedure: SetShellAssociationW

\ObjAsm\Code\ObjMem\64\SetShellAssociationW.asm Files:

\ObjAsm\Code\ObjMem\Common\SetShellAssociation\_TX.inc Set association for a file extension.

Purnose:

Arguments: Arg1: TRUE = system wide association, FALSE = user account only.

 $Arg2: \rightarrow File$  extension (without dot).  $Arg3: \rightarrow Verb$  ("open", "print", "play", "edit", etc.). This verb is displayed in the explorer context menu of a file with this extension.

Arg4: → Application to associate with (full path). Arg5: → Application arguments, usually \$ofscstr("%1").

eax = HRESULT.Return:

dGlobal = TRUE requires adminitrative rights. Note:

Procedure: SetShellPerceivedTypeA

Files:

\ObjAsm\Code\ObjMem\64\SetShellPerceivedTypeA.asm \ObjAsm\Code\ObjMem\Common\SetShellPerceivedType\_TX.inc Set shell perception of a file type.

Purnose:

Arg1: TRUE = system wide perseption, FALSE = user account only. Arguments:

Arg2: → File extension (without dot). Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,

Application, Gamemedia, Contacts)

Return: eax = HRESULT.

To retrieve the perceived type use the AssocGetPerceivedType API. Note:

dGlobal = TRUE requires adminitrative rights.

SetShellPerceivedTypeW Procedure:

Files:

\ObjAsm\Code\ObjMem\64\SetShellPerceivedTypeW.asm \ObjAsm\Code\ObjMem\Common\SetShellPerceivedType\_TX.inc

Set shell perception of a file type. Purpose:

Arguments: Arg1: TRUE = system wide perseption, FALSE = user account only.

Arg2: → File extension (without dot). Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,

Application, Gamemedia, Contacts)

Return: eax = HRESULT.

To retrieve the perceived type use the AssocGetPerceivedType API. Note:

dGlobal = TRUE requires adminitrative rights.

Procedure: ShortToLongPathNameA

File:

\ObjAsm\Code\ObjMem\64\ShortToLongPathNameA.asm
Allocate a new ANSI string containing the long path of a short path string. Purpose:

Arguments: Arg1: → Short path ANSI string.

Return: rax → Long path ANSI string or NULL if failed.

Procedure: SLR\_Calc\_AB\_DW

\ObjAsm\Code\ObjMem\64\SLR\_Calc\_AB\_DW.asm Files: \ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + B that minimize mean squared error (MSE) of a DWORD array. Purpose:

Arguments: Arg1: → SLR\_DATA structure.

eax = TRUE is succeeded, otherwise FALSE. Return:

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_KE2%8B%AFhttps://mathschallenge.net/library/number/sum\_of\_squares Links:

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas:

B = (Y-A\*X)/N

Procedure: SLR\_Calc\_AB\_MSE\_DW

\ObjAsm\Code\ObjMem\64\SLR\_Calc\_AB\_MSE\_DW.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_MSE\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + B that minimize mean squared error (MSE) and the MSE value of a DWORD array. Purpose:

Arguments: Arg1: → SLR\_DATA structure.

eax = TRUE is succeeded, otherwise FALSE. Return:

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_KE2%8B%AFhttps://mathschallenge.net/library/number/sum\_of\_squares links:

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)QFormulas:

B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_MSE\_QW

\ObjAsm\Code\ObjMem\64\SLR\_Calc\_AB\_MSE\_OW.asm \ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_MSE\_XP.inc Files:

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + BPurpose:

that minimize mean squared error (MSE) and the MSE value of a QWORD array.

Arguments: Arg1: → SLR\_DATA structure.

Return: eax = TRUE is succeeded, otherwise FALSE.

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Links:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Since X ranges from [0. N-1], the known formulas have to be adjusted accordingly by Note:

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

 $A = (XY*N-X*Y)\dot{/}Q$ Formulas:

B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_MSE\_R4

\ObjAsm\Code\ObjMem\64\SLR\_Calc\_AB\_MSE\_R4.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_MSE\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + BPurpose:

that minimize mean squared error (MSE) and the MSE value of a REAL4 array.

Arg1: → SLR\_DATA structure. Arguments:

Return:

eax = TRUE is succeeded, otherwise FALSE. https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Tinks:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas:

B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_MSE\_R8

\ObjAsm\Code\ObjMem\64\SLR\_Calc\_AB\_MSE\_R8.asm \ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_MSE\_XP.inc Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + B Files:

Purpose:

that minimize mean squared error (MSE) and the MSE value of a REAL8 array.

Arguments: Arg1: → SLR\_DATA structure

Return: eax = TRUE is succeeded, otherwise FALSE.

Links: https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-linec7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas: B = (Y-A\*X)/N

 $MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X^2 + 2*A*B*X)/N + B^2$ 

Procedure: SLR\_Calc\_AB\_QW

\ObjAsm\Code\ObjMem\64\SLR\_Calc\_AB\_QW.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + BPurpose:

that minimize mean squared error (MSE) of a QWORD array.

Arg1: → SLR\_DATA structure. Arguments:

eax = TRUE is succeeded, otherwise FALSE. Return:

https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Links:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by Note:

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)'/QFormulas: B = (Y-A\*X)/N

Procedure: SLR\_Calc\_AB\_R4

\ObjAsm\Code\ObjMem\64\SLR\_Calc\_AB\_R4.asm Files: \ObjAsm\Code\ObjMem\Common\SLR\_Calc\_AB\_XP.inc

Calculate the Slope (A) and Intercept (B) values of the linear equation y = A\*x + B that minimize mean squared error (MSE) of a REAL4 array. Purpose:

Arguments:

Arg1: → SLR\_DATA structure. eax = TRUE is succeeded, otherwise FALSE. Return:

 $https://en.wikipedia.org/wiki/1\_\%2B\_2\_\%2B\_3\_\%2B\_4\_\%2B\_\%E2\%8B\%AF$ Tinks:

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1

If an FPU exception occurs, the results are NaN.

A = (XY\*N-X\*Y)/QFormulas: B = (Y-A\*X)/N

Procedure: SLR\_Calc\_AB\_R8

Files:

Purnose:

that minimize mean squared error (MSE) of a REAL8 array.

Arguments: Arg1: → SLR\_DATA structure

Return: eax = TRUE is succeeded, otherwise FALSE.

Links: https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF

https://mathschallenge.net/library/number/sum\_of\_squares

https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-linec7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

If an FPU exception occurs, the results are NaN.

Formulas: A = (XY\*N-X\*Y)/Q

B = (Y-A\*X)/N

Procedure: SLR\_Init

\ObjAsm\Code\ObjMem\64\SLR\_Init.asm Files:

\ObjAsm\Code\ObjMem\Common\SLR\_Init\_XP.inc

Calculate in advance the invariant coefficients of a Simple Linear Regression (X, X2, Purpose:

Arguments: Arg1: → SLR\_DATA structure

Return:

eax = TRUE is succeeded, otherwise FALSE.
https://en.wikipedia.org/wiki/1\_%2B\_2\_%2B\_3\_%2B\_4\_%2B\_%E2%8B%AF Links:

https://mathschallenge.net/library/number/sum\_of\_squares https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-

c7dde9a26b93/

Note: Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by

replacing N with N-1.

Formulas: X = N\*(N-1)/2

X2 = X\*(2\*N - 1)/3 $Q = N^2*(N^2-1)/12$ 

Procedure: sqword2decA

Files: \ObjAsm\Code\ObjMem\64\sqword2decA.asm

\ObjAsm\Code\ObjMem\Common\sqword2decT64.inc

Convert a signed QWORD to its decimal ANSI string representation.

Arg1: → Destination ANSI string buffer.

Arg2: SQWORD value. Purpose:

Arguments:

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

Note: The destination buffer must be at least 21 BYTEs large to allocate the output string

(Sign + 19 ANSI characters + ZTC = 21 BYTEs).

Procedure: sqword2decW

Files: \ObjAsm\Code\ObjMem\64\sqword2decW.asm

\ObjAsm\Code\ObjMem\Common\sqword2decT64.inc

Convert a signed SQWORD to its decimal WIDE string representation.

Arg1: → Destination WIDE string buffer. Arguments:

Arg2: SQWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 42 BYTES large to allocate the output string (Sign + 19 WIDE characters + ZTC = 42 BYTES). Note:

Procedure: StOToStrA

Files: \ObjAsm\Code\ObjMem\64\StOToStrA.asm

\ObjAsm\Code\ObjMem\Common\StOToStr\_AXP.inc

Create an ANSI string representation of the content of the st(0) FPU register. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: Minimal number of places from the start of string up to the decimal point.

(f\_NOR only)

Arg3: Number of decimal places after the decimal point.

Return:

Notes:

Arg3: Number of decimal places after the decimal point.

Arg4: Format flag (f\_NOR, f\_SCI, f\_TRIM, f\_ALIGNED) defined in fMath.inc eax = Result code f\_OK, f\_ERROR, f\_NAN, ...

- Based on the work of Raymond Filiatreault (FpuLib).

- st4, st5, st6 and st7 must be empty.

- f\_NOR: regular output format

- f\_SCI: Scientific output format

- f\_TRIM: Trim zeros on the right - f\_ALIGN: Add a heading space to align the output with other negative numbers

- f\_PLUS: like f\_ALIGN, but using a + character.

Procedure: StOToStrW

\ObjAsm\Code\ObjMem\64\StOToStrW.asm Files:

\ObjAsm\Code\ObjMem\Common\StOToStr\_wXP.inc

Create a WIDE string representation of the content of the st(0) FPU register. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: Minimal number of places from the start of string up to the decimal point.

(f NOR only)

Arg3: Number of decimal places after the decimal point.

Arg4: Format flag (f\_NOR, f\_SCI, f\_TRIM, f\_ALIGNED) defined in fMath.inc eax = Result code f\_OK, f\_ERROR, f\_NAN, ...

- Based on the work of Raymond Filiatreault (FpuLib).

Return:

Notes:

- st4, st5, st6 and st7 must be empty. - f\_NOR: regular output format - f\_SCI: Scientific output format - f\_TRIM: Trim zeros on the right

- f\_ALIGN: Add a heading space to align the output with other negative numbers - f\_PLUS: like f\_ALIGN, but using a + character.

Procedure: StkGrdCallback

\ObjAsm\Code\ObjMem\64\StkGrdCallback.asm File: Purpose: StackGuard notification callback procedure.

It is called when StackGuard is active and a stack overrun was detected. It displays a MessageBox asking to abort. If yes, then Exitprocess is called

immediately.

Arguments: None. Nothing. Return:

Procedure: Str2BStrA

\ObjAsm\Code\ObjMem\64\Str2BStrA.asm File:

Purpose: Convert a ANSI string into a BStr.

Arguments: Arg1: → Destination BStr buffer = Buffer address + sizeof(DWORD).

Arg2: → Source ANSI string.

Return: eax = String length.

Procedure: Str2BStrW

\ObjAsm\Code\ObjMem\64\Str2BStrw.asm Convert a WIDE string into a BStr.

Arguments: Arg1: → Destination BStr buffer = Buffer address + sizeof(DWORD).

Arg2: → Source WIDE string.

Return: Nothing.

Procedure: StrA2StrW

\ObjAsm\Code\ObjMem\64\StrA2StrW.asm File: Purpose: Convert a ANSI string into a WIDE string. Arguments: Arg1:  $\rightarrow$  Destination WIDE string buffer. Arg2:  $\rightarrow$  Source ANSI string.

Nothing. Return:

Procedure: StrAllocA

\ObjAsm\Code\ObjMem\64\StrAllocA.asm

\ObjAsm\Code\ObjMem\Common\StrAlloc\_TX.inc

Allocate space for a string with n characters. Purpose:

Arguments: Arg1: Character count without the ZTC.
Return: rax → New allocated string or NULL if failed.

Procedure: StrAllocA UEFI

\ObjAsm\Code\ObjMem\64\StrAllocA\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\StrAlloc\_TX\_UEFI.inc Purpose: Allocate space for a string with n characters.

Arguments: Arg1: Character count without the ZTC.

Return:  $rax \rightarrow New$  allocated string or NULL if failed.

Procedure: StrAllocW

Files:

\ObjAsm\Code\ObjMem\64\StrAllocW.asm \ObjAsm\Code\ObjMem\Common\StrAlloc\_TX.inc Purnose: Allocate space for a string with n characters.

Arguments: Arg1: Character count without the ZTC.

 $rax \rightarrow New$  allocated string or NULL if failed. Return:

Procedure: StrAllocW\_UEFI

Files: \ObjAsm\Code\ObjMem\64\StrAllocW\_UEFI.asm

\ObjAsm\Code\ObjMem\Common\StrAlloc\_TX\_UEFI.inc Allocate space for a string with n characters. Purpose: Arguments: Arg1: Character count without the ZTC.

 $rax \rightarrow New$  allocated string or NULL if failed. Return:

Procedure: StrCatA

\ObjAsm\Code\ObjMem\64\StrCatA.asm File: Concatenate 2 ANSI strings. Purpose: Arguments: Arg1: Destrination ANSI buffer.

Arg2: Source ANSI string.

Return: Nothing.

Procedure: StrCatCharA

\ObjAsm\Code\ObjMem\64\StrCatCharA.asm

Purpose: Append a character to the end of an ANSI string.

Arguments: Arg1: Destrination ANSI buffer. Arg2: ANSI character.

Nothing. Return:

Procedure: StrCatCharW

File:

\ObjAsm\Code\ObjMem\64\StrCatCharW.asm
Append a character to the end of an WIDE string. Purpose:

Arguments: Arg1: Destrination ANSI buffer.

Arg2: WIDE character.

Nothing. Return:

Procedure: StrCatW

\ObjAsm\Code\ObjMem\64\StrCatW.asm File: Purpose: Concatenate 2 WIDE strings.
Arguments: Arg1: Destrination WIDE string.

Arg2: Source WIDE string.

Return: Nothing.

Procedure: StrCCatA

File: \ObjAsm\Code\ObjMem\64\StrCCatA.asm

Concatenate 2 ANSI strings with length limitation.

The destination string buffer should have at least enough room for the maximum number

of characters + 1.

Arg1:  $\rightarrow$  Destination ANSI character buffer. Arg2:  $\rightarrow$  Source ANSI string. Arguments:

Arg3: Maximal number of charachters that the destination string can hold including the

7TC.

rax = Number of added BYTEs. Return:

Procedure: StrCCatCharA

\ObjAsm\Code\ObjMem\64\StrCCatCharA.asm File:

Append a character to the end of an ANSI string with length limitation. Purpose:

Arguments: Arg1: → Destination ANSI character buffer.

Arg2: → ANSI character.

Arg3: Maximal number of characters that fit into the destination buffer.

Return: Nothing.

Procedure: StrCCatCharW

\ObjAsm\Code\ObjMem\64\StrCCatCharW.asm File:

Append a character to the end of a WIDE string with length limitation. Purnose:

Arguments: Arg1:  $\rightarrow$  Destination WIDE character buffer.

Arg2: → WIDE character.
Arg3: Maximal number of characters that fit into the destination buffer.

Return: Nothina.

Procedure: StrCCatW

File: \ObjAsm\Code\ObjMem\64\StrCCatW.asm

Concatenate 2 WIDE strings with length limitation. Purpose:

The destination string buffer should have at least enough room for the maximum number

of characters + 1.

Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string. Arguments:

Arg3: Maximal number of charachters that the destination string can hold including the

ZTC.

Return: rax = Number of added BYTEs.

Procedure: StrCCompA

\ObjAsm\Code\ObjMem\64\StrCCompA.asm

Purpose: Compare 2 ANSI strings with case sensitivity up to a maximal number of characters.

Arguments: Arg1:  $\rightarrow$  ANSI string 1. Arg2:  $\rightarrow$  ANSI string 2.

Arg3: Maximal number of characters to compare. If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0. Return:

Procedure: StrCCompW

File: \ObjAsm\Code\ObjMem\64\StrCCompW.asm

Purpose: Compare 2 WIDE strings with case sensitivity up to a maximal number of characters.

Arguments: Arg1:  $\rightarrow$  WIDE string 1.

Arg2: → WIDE string 2.

Arg3: Maximal number of characters to compare.

Return:

If string 1 < string 2, then eax < 0.
If string 1 = string 2, then eax = 0.
If string 1 > string 2, then eax > 0.

Procedure: StrCCopyA

\ObjAsm\Code\ObjMem\64\StrCCopyA.asm File:

Copy the the source ANSI string with length limitation. Purpose:

The destination buffer should be large enough to hold the maximum number of

characters + 1.

Arguments: Arg1:  $\rightarrow$  Destination buffer. Arg2:  $\rightarrow$  Source ANSI string.

Arg3: Maximal number of charachters to copy, excluding the ZTC.

Return: rax = Number of copied BYTEs, including the ZTC.

Procedure: StrCCopyW

File:

\ObjAsm\Code\ObjMem\64\StrCCopyW.asm
Copy the the source WIDE string with length limitation. Purpose:

The destination buffer should be big enough to hold the maximum number of

characters + 1.

Arguments:  $Arg1: \rightarrow Destination buffer.$   $Arg2: \rightarrow Source WIDE string.$ 

Arg3: Maximal number of charachters to copy, excluding the ZTC.

rax = Number of copied BYTEs, including the ZTC. Return:

Procedure: StrCECatA

\ObjAsm\Code\ObjMem\64\StrCECatA.asm File:

Concatenate 2 ANSI strings with length limitation and return the ZTC address. The destination string buffer should have at least enough room for the maximum Purpose:

number of characters + 1.

Arguments: Arg1: → Destination ANSI character buffer.

Arg2: → Source ANSI string.

Arg3: Maximal number of charachters that the destination string can hold including the

ZTC.

 $rax \rightarrow ZTC.$ Return:

Procedure:

\ObjAsm\Code\ObjMem\64\StrCECatW.asm File:

Concatenate 2 WIDE strings with length limitation and return the ZTC address. The destination string buffer should have at least enough room for the maximum Purpose:

number of characters + 1.

Arg1: → Destination WIDE character buffer. Arg2: → Source WIDE string. Arguments:

Arg3: Maximal number of charachters that the destination string can hold including the

ZTC.

Return:  $rax \rightarrow ZTC$ .

Procedure: StrCECopyA

File: \ObjAsm\Code\ObjMem\64\StrCECopyA.asm

Copy the the source ANSI string with length limitation and return the ZTC address. Purpose:

The destination buffer should hold the maximum number of characters + 1.

Source and destination strings may overlap. Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string. Arguments:

Arg3: Maximal number of characters not including the ZTC.

 $rax \rightarrow ZTC$ . Return:

Procedure: StrCECopyW

\ObjAsm\Code\ObjMem\64\StrCECopyW.asm File:

Purpose: Copy the the source WIDE string with length limitation and return the ZTC address.

The destination buffer should hold the maximum number of characters + 1.

Source and destination strings may overlap.

Arguments: Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string.

Arg3: Maximal number of characters not including the ZTC.

 $rax \rightarrow ZTC.$ Return:

Procedure: StrCICompA

\ObjAsm\Code\ObjMem\64\StrCICompA.asm File:

Purpose: Compare 2 ANSI strings without case sensitivity and length limitation.

Arguments: Arg1: → ANSI string 1.

Arg2: → ANSI string 2.

Return:

If string 1 < string 2, then eax < 0.
If string 1 = string 2, then eax = 0.
If string 1 > string 2, then eax > 0.

Procedure: StrCICompW

File:

\ObjAsm\Code\ObjMem\64\StrCICompW.asm
Compare 2 WIDE strings without case sensitivity and length limitation. Purpose:

Arguments: Arg1:  $\rightarrow$  WIDE string 1. Arg2: → WIDE string 2.

Return:

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0.

Procedure: StrCLengthA

\ObjAsm\Code\ObjMem\64\StrCLengthA.asm File:

Purpose: Get the character count of the source ANSI string with length limitation.

Arguments: Arg1: → Source ANSI string.

Arg3: Maximal character count. eax = Limited character count. Return:

Procedure: StrCLenathW

\ObjAsm\Code\ObjMem\64\StrCLengthW.asm

Purpose: Get the character count of the source WIDE string with length limitation.

Arguments: Arg1: → Source WIDE string. Arg3: Maximal character count.

eax = Limited character count. Return:

Procedure: StrCNewA

\ObjAsm\Code\ObjMem\64\StrCNewA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrCNewT64P.inc

Allocate a new copy of the source ANSI string with length limitation. Purpose:

If the pointer to the source string is NULL or points to an empty string, StrCNewA returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter.

Arg1: → Source ANSI string. Arguments:

Arg2: Maximal character count.

Return:  $rax \rightarrow New ANSI string copy.$ 

Procedure: StrCNewA\_UEFI

\ObjAsm\Code\ObjMem\64\StrCNewA\_UEFI.asm Files: \ObjAsm\Code\ObjMem\Common\StrCNewT64P.inc

Allocate a new copy of the source ANSI string with length limitation. Purpose:

If the pointer to the source string is NULL or points to an empty string, StrCNewA returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter.

Arg1: → Source ANSI string. Arg2: Maximal character count. Arguments: Return:  $rax \rightarrow New ANSI string copy.$ 

Procedure: StrCNewW

\ObjAsm\Code\ObjMem\64\StrCNewW.asm Files:

\ObjAsm\Code\ObjMem\Common\StrCNewT64P.inc
Allocate a new copy of the source WIDE string with length limitation. Purpose:

If the pointer to the source string is NULL or points to an empty string, StrCNewW returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter.

Arguments: Arg1: → Source WIDE string.

Arg2: Maximal character count. Return:  $rax \rightarrow New WIDE string copy.$ 

Procedure: StrCNewW\_UEFI

\ObjAsm\Code\ObjMem\64\StrCNewW\_UEFI.asm Files:

ObjAsm\Code\ObjMem\Common\StrCNewT64P.inc
Allocate a new copy of the source WIDE string with length limitation. Purpose:

If the pointer to the source string is NULL or points to an empty string, StrCNewW returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a duplicate of the source string. The maximal size of the new string is limited to the

second parameter.

Arguments: Arg1: → Source WIDE string.

Arg2: Maximal character count. Return:  $rax \rightarrow New WIDE string copy.$ 

Procedure: StrCompA

\ObjAsm\Code\ObjMem\64\StrCompA.asm

Purpose: Compare 2 ANSI strings with case sensitivity.

Arguments: Arg1: → ANSI string 1.

Arg2: → ANSI string 2.

Return: If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0.

Procedure: StrCompW

File: \ObjAsm\Code\ObjMem\64\StrCompW.asm

Purpose: Compare 2 WIDE strings with case sensitivity.

Arguments: Arg1:  $\rightarrow$  WIDE string 1. Arg2:  $\rightarrow$  WIDE string 2.

Return:

If string 1 < string 2, then eax < 0.
If string 1 = string 2, then eax = 0.
If string 1 > string 2, then eax > 0.

Procedure: StrCopyA

File: \ObjAsm\Code\ObjMem\64\StrCopyA.asm

Copy an ANSI string to a destination buffer. Purpose:

Arguments: Arg1: Destrination ANSI string.

Arg2: Source ANSI string. Return: rax = Number of BYTEs copied, including the ZTC.

Procedure: StrCopyW

\ObjAsm\Code\ObjMem\64\StrCopyW.asm File:

Purpose: Copy a WIDE string to a destination buffer. Arguments: Arg1: Destrination WIDE string buffer.

Arg2: Source WIDE string.

eax = Number of BYTEs copied, including the ZTC. Return:

Procedure: StrCPosA

\ObjAsm\Code\ObjMem\64\StrCPosA.asm File:

Scan for ANSI string2 into ANSI string1 with length limitation.

Arguments: Arg1: → Source ANSI string.

Arg2: → ANSI string to search for. Arg3: Maximal character count.

 $rax \rightarrow String position or NULL if not found.$ Return:

Procedure: StrCPosW

File: \( \lambda \text{ObjAsm} \text{Code} \lambda \text{bjMem} \\ 64 \\ \text{StrCPosw.asm} \)

Purpose: Scan for WIDE string2 into WIDE string1 with length limitation.

Arg1: \( \rightarrow \text{MIDE string} \)

Arg2: \( \rightarrow \text{WIDE string} \)

Arg3: \( \text{Maximal character sound} \)

Arg3: Maximal character count.

 $rax \rightarrow String position or NULL if not found.$ Return:

Procedure: StrCScanA

\ObjAsm\Code\ObjMem\64\StrCScanA.asm

Scan from the beginning of ANSI string for a character with length limitation. Purpose:

Arguments: Arg1: → Source ANSI string.
Arg2: Maximal character count.

Arg3: ANSI character to search for.

 $rax \rightarrow Character address or NULL if not found.$ Return:

Procedure: StrCScanW

\ObjAsm\Code\ObjMem\64\StrCScanW.asm

Purpose: Scan from the beginning of a WIDE string for a character with length limitation.

Arguments: Arg1: → Source WIDE string. Arg2: Maximal character count.

Arg3: WIDE character to search for.

 $rax \rightarrow Character address or NULL if not found.$ Return:

Procedure: StrDispose

\ObjAsm\Code\ObjMem\64\StrDispose.asm File:

Free the memory allocated for the string using StrNew, StrCNew, StrLENew or Purpose:

StrAlloc.

If the pointer to the string is NULL, StrDispose does nothing.

Arguments: Arg1: → String.

Nothing. Return:

Procedure: StrDispose\_UEFI

\ObjAsm\Code\ObjMem\64\StrDispose\_UEFI.asm Files:

\ObjAsm\Code\ObjMem\Common\StrDispose\_TX\_UEFI.inc

Free the memory allocated for the string using StrNew\_UEFI, StrCNew\_UEFI, Purpose:

StrLENew\_UEFI or StrAlloc\_UEFI.

If the pointer to the string is NULL, StrDispose\_UEFI does nothing.

Arguments: Arg1: → String.

Return: Nothina.

Procedure: StrECatA

File: \ObjAsm\Code\ObjMem\64\StrECatA.asm

Append an ANSI string to another and return the address of the ending zero character. Purpose:

StrCatA does not perform any length checking. The destination buffer must have room for at least StrLengthA(Destination) + StrLengthA(Source) + 1 characters.

Arguments: Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string.

 $rax \rightarrow ZTC$ .

Return:

Procedure: StrECatCharA

\ObjAsm\Code\ObjMem\64\StrECatCharA.asm File:

Append a character to an ANSI string and return the address of the ending zero. Purpose:

StrECatCharA does not perform any length checking. The destination buffer must have

enough room for at least StrLengthA(Destination) + 1 + 1 characters.

Arguments: Arg1:  $\rightarrow$  Destination ANSI string buffer. Arg2:  $\rightarrow$  ANSI character.

 $rax \rightarrow ZTC$ . Return:

Procedure: StrECatCharW

\ObjAsm\Code\ObjMem\64\StrECatCharw.asm File:

Append a character to a WIDE string and return the address of the ending zero. Purpose:

StrECatCharW does not perform any length checking. The destination buffer must have enough room for at least StrLengthW(Destination)  $+\ 1 + 1$  characters.

Arguments: Arg1:  $\rightarrow$  Destination WIDE string buffer. Arg2:  $\rightarrow$  WIDE character.

 $rax \rightarrow ZTC$ . Return:

Procedure:

File:

\ObjAsm\Code\ObjMem\64\StrECatw.asm
Append a WIDE string to another and return the address of the ending zero character.
StrCatw\_does not perform any length checking. The destination buffer must have room Purpose:

for at least StrLengthW(Destination) + StrLengthW(Source) + 1 characters.

Arg1: → Destination WIDE character buffer. Arg2: → Source WIDE string. Arguments:

 $rax \rightarrow ZTC$ . Return:

Procedure: StrECopyA

\ObjAsm\Code\ObjMem\64\StrECopyA.asm File:

Purpose: Copy an ANSI string to a buffer and return the address of the ZTC.

Source and destination strings may overlap.

Arg1: → Destination ANSI character string. Arguments:

Arg2: → Source ANSI string.

Return:  $rax \rightarrow ZTC.$ 

Procedure: StrECopyW

\ObjAsm\Code\ObjMem\64\StrECopyW.asm File:

Copy a WIDE string to a buffer and return the address of the ZTC. Purpose:

Source and destination strings may overlap.

Arg1:  $\rightarrow$  Destination WIDE character string. Arg2:  $\rightarrow$  Source WIDE string. Arguments:

 $rax \rightarrow ZTC$ . Return:

Procedure: StrEndA

\ObjAsm\Code\ObjMem\64\StrEndA.asm File:

Purpose: Get the address of the zero character that terminates the string.

Arguments: Arg1: → Source ANSI string.

 $rax \rightarrow ZTC$ . Return:

Procedure: StrEndsWithA

Files: \ObjAsm\Code\ObjMem\64\StrEndsWithA.asm

\ObjAsm\Code\ObjMem\Common\StrEndsWith\_TXP.inc

Purpose: Compare the ending of a string.

Arguments: Arg1: → Analized string.

Arg2: → Suffix string.

eax = TRUE of the ending matches, otherwise FALSE. Return:

Procedure: StrEndsWithW

Files:

\ObjAsm\Code\ObjMem\64\StrEndsWithW.asm \ObjAsm\Code\ObjMem\Common\StrEndsWith\_TXP.inc

Purpose: Compare the ending of a string.

Arguments: Arg1: → Analized string.

Arg2: → Suffix string.

Return: eax = TRUE of the ending matches, otherwise FALSE.

Procedure: StrEndW

\ObjAsm\Code\ObjMem\64\StrEndw.asm

Get the address of the zero character that terminates the string. Purpose:

Arguments: Arg1:  $\rightarrow$  Source WIDE string. Return: rax  $\rightarrow$  ZTC.

Procedure: StrFillChrA

\ObjAsm\Code\ObjMem\64\StrFillChrA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrFillChr\_TXP.inc Purpose: Fill a preallocated String with a character.

Arguments: Arg1: → String.

Arg2: Character. Arg3: Character Count.

Nothing. Return:

Procedure: StrFillChrW

Files:

\ObjAsm\Code\ObjMem\64\StrFillChrw.asm \ObjAsm\Code\ObjMem\Common\StrFillChr\_TXP.inc Fill a preallocated String with a character. Purpose:

Arg1:  $\rightarrow$  String. Arguments:

Arg2: Character. Arg3: Character Count.

Return: Nothing. Procedure: StrFilterA

\ObjAsm\Code\ObjMem\64\StrFilterA.asm File:

Perform a case sensitive string match test using wildcards (\* and ?).

Arguments: Arg1: → Source ANSI string.

Arg2: → Filter ANSI string.

Return: eax = TRUE if strings match, otherwise FALSE.

Procedure: StrFilterW

File:

\\\\ObiAsm\\Code\\obiMem\\64\\strFilterW.asm}\\\Perform a case sensitive string match test using wildcards (\* and ?). Purpose:

Arguments: Arg1: → Source WIDE string. Arg2: → Filter WIDE string.

Return: eax = TRUE if strings match, otherwise FALSE.

Procedure: StrICompA

\ObjAsm\Code\ObjMem\64\StrICompA.asm

Compare 2 ANSI strings without case sensitivity. Purpose:

Arguments: Arg1:  $\rightarrow$  ANSI string 1. Arg2:  $\rightarrow$  ANSI string 2.

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0. Return:

Procedure: StrICompW

\ObjAsm\Code\ObjMem\64\StrICompW.asm

Purpose: Compare 2 WIDE strings without case sensitivity.

Arguments: Arg1:  $\rightarrow$  WIDE string 1. Arg2:  $\rightarrow$  WIDE string 2.

Return:

If string 1 < string 2, then eax < 0. If string 1 = string 2, then eax = 0. If string 1 > string 2, then eax > 0.

Procedure: StrIFilterA

File: \ObjAsm\Code\ObjMem\64\StrIFilterA.asm

Purpose: Perform a case insensitive string match test using wildcards (\* and ?).

Arguments: Arg1:  $\rightarrow$  Source ANSI string. Arg2:  $\rightarrow$  Filter ANSI string.

eax = TRUE if strings match, otherwise FALSE.

Procedure: StrIFilterW

Return:

File: \ObjAsm\Code\ObjMem\64\StrIFilterw.asm

Purpose: Perform a case insensitive string match test using wildcards (\* and ?). Arguments: Arg1:  $\rightarrow$  Source WIDE string.

Arg2: → Filter WIDE string.

Return: eax = TRUE if strings match, otherwise FALSE.

Procedure: StrLeftA

\ObjAsm\Code\ObjMem\64\StrLeftA.asm

Extract the left n characters of the source ANSI string.

Arguments: Arg1: → Destination character buffer. Arg2: → Source ANSI string.

Return: eax = Number of copied characters, not including the ZTC.

Procedure: StrLeftW

File:

\ObjAsm\Code\ObjMem\64\StrLeftw.asm
Extract the left n characters of the source WIDE string. Purpose:

Arguments: Arg1: → Destination buffer.

Arg2: → Source WIDE string. eax = Number of copied characters, not including the ZTC. Return:

Procedure: StrLengthA

\ObjAsm\Code\ObjMem\64\StrLengthA.asm

Purpose: Determine the length of an ANSI string not including the zero terminating character. Arguments: Arg1: → Source ANSI string.

eax = Length of the string in characters. Return:

Procedure: StrLengthW

\ObjAsm\Code\ObjMem\64\StrLengthw.asm File:

Purpose: Determine the length of a WIDE string not including the ZTC.

Arguments: Arg1: → WIDE string.

rax = Length of the string in characters. Return:

Procedure: StrLowerA

\ObjAsm\Code\ObjMem\64\StrLowerA.asm

Purpose: Convert all ANSI string characters into lowercase.

Arguments: Arg1: → Source ANSI string.

 $rax \rightarrow String.$ Return:

Procedure: StrLowerW

\ObjAsm\Code\ObjMem\64\StrLowerW.asm File:

Convert all WIDE string characters into lowercase. Purpose:

Arguments: Arg1: → Source WIDE string.

 $rax \rightarrow String.$ Return:

Procedure: StrLRTrimA

\ObjAsm\Code\ObjMem\64\StrLRTrimA.asm

Trim blank and tab characters from the beginning and the end of an ANSI string. Purpose:

Arguments: Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string.

Nothing. Return:

Source and Destination may overlap. Note:

Procedure: StrLRTrimW

\ObjAsm\Code\ObjMem\64\StrLRTrimW.asm

Purpose: Trim blank and tab characters from the beginning and the end of a WIDE string.

Arguments: Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string.

Return: Nothing.

Note: Source and Destination may overlap.

Procedure: StrLScanA

\ObjAsm\Code\ObjMem\64\StrLScanA.asm File:

Purpose: Scan for a character from the beginning of an ANSI string. Arguments: Arg1: → Source ANSI string.

Arg2: Character to search.
rax → Character address or NULL if not found. Return:

Procedure: StrLScanW

\ObjAsm\Code\ObjMem\64\StrLScanW.asm

Purpose: Scan for a character from the beginning of a WIDE string.

Arguments: Arg1: → Source WIDE string.
Arg2: Character to search for.

rax → Character address or NULL if not found. Return:

Procedure: StrLTrimA

File:

\ObjAsm\Code\ObjMem\64\StrLTrimA.asm
Trim blank characters from the beginning of an ANSI string. Purpose:

Arguments: Arg1: → Destination ANSI character buffer. Arg2: → Source ANSI string.

Nothing.

Procedure: StrLTrimW

\ObjAsm\Code\ObjMem\64\StrLTrimW.asm File:

Trim blank characters from the beginning of a WIDE string. Purpose:

Arguments: Arg1: → Destination WIDE character buffer. Arg2: → Source WIDE string.

Return: Nothing.

Procedure: StrMidA

\ObjAsm\Code\ObjMem\64\StrMidA.asm File:

Purpose: Extract a substring from an ANSI source string.

Arguments: Arg1: → Destination ANSI character buffer.

Arg2: → Source ANSI string.

Arg3: Start character index. Index ranges [0 .. length-1].

Arg3: Character count.

eax = Number of copied characters. Return:

Procedure: StrMidW

File: \ObjAsm\Code\ObjMem\64\StrMidW.asm

Purpose: Extract a substring from a WIDE source string. Arg1: → Destination WIDE character buffer. Arguments:

Arg2: → Source WIDE string.

Arg3: Start character index. Index ranges [0 .. length-1].

Arg3: Character count.
eax = Number of copied characters. Return:

Procedure: StrMoveW

File: \ObjAsm\Code\ObjMem\64\StrMoveW.asm

Move part of a WIDE string. The ZTC is NOT appended automatically. Source and destination strings may overlap. Purpose:

Arg1:  $\rightarrow$  Destination buffer. Arguments:

Arg2: → Source WIDE string. Arg3: Character count.

Return: Nothina.

Procedure:

\ObjAsm\Code\ObjMem\64\StrNewA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc

Allocate a new copy of the source string. Purpose:

If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate any memory space. Otherwise, StrNew makes a duplicate of the source string. The allocated memory space is Length(String) + ZTC.

Arguments: Arg1: → Source WIDE string.

Return: rax → New string copy.

Procedure: StrNewA\_UEFI

Files: \ObjAsm\Code\ObjMem\64\StrNewA\_UEFI.asm

\ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc

Allocate a new copy of the source string. Purpose:

If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate any memory space. Otherwise, StrNew makes a duplicate of the source string.

The allocated memory space is Length(String) + ZTC.

Arg1: → Source WIDE string.

Arguments:

 $rax \rightarrow New string copy.$ Return:

Procedure: StrNewW

Files: \ObjAsm\Code\ObjMem\64\StrNewW.asm

\ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc Purpose: Allocate a new copy of the source string.

If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate

any memory space. Otherwise, StrNew makes a duplicate of the source string. The allocated memory space is Length(String) + ZTC.

Arguments: Arg1: → Source WIDE string.

Return:  $rax \rightarrow New string copy.$ 

Procedure: StrNewW\_UEFI

\ObjAsm\Code\ObjMem\64\StrNewW\_UEFI.asm Files: \ObjAsm\Code\ObjMem\Common\StrNew\_TXP.inc

Purpose:

Allocate a new copy of the source string.
If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate

any memory space. Otherwise, StrNew makes a duplicate of the source string.

The allocated memory space is Length(String) + ZTC. Arguments: Arg1: → Source WIDE string.

rax → New string copy. Return:

Procedure: StrPosA

File:

\ObjAsm\Code\ObjMem\64\StrPosA.asm
Find the occurence of string 2 into string1. Purpose:

Arguments: Arg1: → Source ANSI string.
Arg2: → Searched ANSI string.

rax → String occurence or NULL if not found. Return:

Procedure: StrPosW

Return:

\ObjAsm\Code\ObjMem\64\StrPosW.asm

Find the occurence of string 2 into string1. Purpose:

Arguments: Arg1: → Source WIDE string.
Arg2: → Searched WIDE string.

rax → String occurence or NULL if not found.

Procedure: StrRepChrA

\ObjAsm\Code\ObjMem\64\StrRepChrA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrRepChr\_TXP.inc Purpose: Create a new string filled with a given char.

Arguments: Arg1: Used character.

Arg2: Repetition count.

 $rax \rightarrow New string or NULL if failed.$ 

Procedure: StrRepChrW

\ObjAsm\Code\ObjMem\64\StrRepChrw.asm Files:

\ObjAsm\Code\ObjMem\Common\StrRepChr\_TXP.inc

Create a new string filled with a given char. Purpose:

Arguments: Arg1: Used character.

Arg2: Repetition count.

 $rax \rightarrow New string or NULL if failed.$ Return:

Procedure: StrRightA

\ObjAsm\Code\ObjMem\64\StrRightA.asm File:

Purpose: Copy the right n characters from the source string into the destination buffer.

Arguments: Arg1:  $\rightarrow$  Destination ANSI character buffer. Arg2:  $\rightarrow$  Source ANSI string.

Arg3: Character count.
rax = Number of characters. Return:

Procedure: StrRightW

\ObjAsm\Code\ObjMem\64\StrRightW.asm File:

Purpose: Copy the right n characters from the source string into the destination buffer. Arguments: Arg1:  $\rightarrow$  Destination WIDE character buffer.

Arg2: → Source WIDE string.

Arg3: Character count. rax = Number of characters. Return:

Procedure: StrRScanA

\ObjAsm\Code\ObjMem\64\StrRScanA.asm

Purpose: Scan from the end of an ANSI string for a character. Arguments: Arg1: → Source ANSI string.

Arg2: Character to search for.
rax → Character address or NULL if not found. Return:

Procedure: StrRScanW

File: \ObjAsm\Code\ObjMem\64\StrRScanW.asm

Purpose: Scan from the end of a WIDE string for a character.

Arguments: Arg1: → Source WIDE string.

Arg2: Character to search for

 $rax \rightarrow Character address or NULL if not found.$ Return:

Procedure: StrRTrimA

File:

\ObjAsm\Code\ObjMem\64\StrRTrimA.asm
Trim blank characters from the end of an ANSI string. Purpose:

Arguments: Arg1:  $\rightarrow$  Destination ANSI character buffer. Arg2:  $\rightarrow$  Source ANSI string.

Return: Nothing.

Procedure: StrRTrimW

\ObjAsm\Code\ObjMem\64\StrRTrimW.asm

Trim blank characters from the end of a WIDE string. Purnose:

Arguments: Arg1: → Destination WIDE character buffer.

Arg2: → Source WIDE string. eax = Number of characters in destination buffer. Return:

Procedure: StrSizeA

\ObjAsm\Code\ObjMem\64\StrSizeA.asm File:

Purpose: Determine the size of an ANSI string including the ZTC.

Arguments: Arg1: → ANSI string.
Return: eax = Size of the string in BYTEs.

Procedure: StrSizeW

\ObjAsm\Code\ObjMem\64\StrSizeW.asm

Determine the size of a WIDE string including the ZTC. Purpose:

Arguments: Arg1: → WIDE string.

rax = Size of the string in BYTEs. Return:

Procedure: StrStartsWithA

\ObjAsm\Code\ObjMem\64\StrStartsWithA.asm Files:

\ObjAsm\Code\ObjMem\Common\StrStartsWith\_TXP.inc

Purpose: Compare the beginning of a string.

Arguments: Arg1: → Analized string. Arg2: → Prefix string.

Return: eax = TRUE of the beginning matches, otherwise FALSE.

Procedure: StrStartsWithW

\ObjAsm\Code\ObjMem\64\StrStartsWithW.asm Files:

\ObjAsm\Code\ObjMem\Common\StrStartsWith\_TXP.inc

Compare the beginning of a string. Purpose:

Arguments: Arg1: → Analized string.

Arg2: → Prefix string.

eax = TRUE of the beginning matches, otherwise FALSE. Return:

Procedure: StrToStOA

\ObjAsm\Code\ObjMem\64\StrToStOA.asm File:

Purpose: Load an ANSI string representation of a floating point number into the st(0)

FPU register.

Arguments: Arg1: → ANSI string floating point number.

Return:

eax = Result code f\_OK or f\_ERROR.

- Based on the work of Raymond Filiatreault (FpuLib). Notes:

- Source string should not be greater than 19 chars + zero terminator.

Procedure: StrToStOW

\ObjAsm\Code\ObjMem\64\StrToStOw.asm File:

Load a WIDE string representation of a floating point number into the st(0) Purpose:

FPU register.

Arguments:  $Arg1: \rightarrow WIDE$  string floating point number.

Return:

eax = Result code f\_OK or f\_ERROR.
- Based on the work of Raymond Filiatreault (FpuLib).

- Source string should not be greater than 19 chars + zero terminator.

Procedure: StrUpperA

File:

\\\\ObjAsm\\Code\\ObjMem\\64\\StrUpperA.asm\\\Convert all ANSI string characters into uppercase. Purnose:

Arguments: Arg1:  $\rightarrow$  Source ANSI string. Return: rax  $\rightarrow$  String.

Procedure: StrUpperW

\ObjAsm\Code\ObjMem\64\StrUpperW.asm File:

Convert all WIDE string characters into uppercase.

Arguments: Arg1: → Source WIDE string.

 $rax \rightarrow String.$ Return:

Procedure: StrW2StrA

File: \ObjAsm\Code\ObjMem\64\StrW2StrA.asm

Convert a WIDE string into an ANSI string. WIDE characters are converted to BYTEs by Purpose:

decimation of the high byte.

Arguments: Arg1:  $\rightarrow$  Destination ANSI character buffer. Arg2:  $\rightarrow$  Source WIDE string.

rax = Number of characters.

Procedure: SysShutdown

Return:

\ObjAsm\Code\ObjMem\64\SysShutdown.asm

Purpose: Shut down the system. Arguments: Arg1: Shutdown type.

Arg2:

Return: Nothing.

Procedure: SysStandby

\ObjAsm\Code\ObjMem\64\SysStandby.asm Purpose: Set the system in standby modus.

Arguments: None. Nothing. Return:

Procedure: uCRC32C

\ObjAsm\Code\ObjMem\64\uCRC32C.asm Files:

ObjAsm\Code\ObjMem\Common\uCRC32C\_XP.inc Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an unaligned Purnose:

memory block.

Arguments:  $Arg1: \rightarrow Unaligned memory block.$ 

Arg2: Memory block size in BYTEs.

Return: eax = CRC32C.

Procedure: udword2decA

Files: \ObjAsm\Code\ObjMem\64\udword2decA.asm

\ObjAsm\Code\ObjMem\Common\udword2decT64.inc

Convert a unsigned DWORD to its decimal ANSI string representation. Purpose:

Arg1: → Destination ANSI string buffer. Arguments:

Arg2: DWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 11 BYTEs large to allocate the output string (10 ANSI characters + ZTC = 11 BYTEs). Note:

Procedure: udword2decW

\ObjAsm\Code\ObjMem\64\udword2decW.asm Files:

\ObjAsm\Code\ObjMem\Common\udword2decT64.inc

Convert an unsigned DWORD to its decimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination WIDE string buffer.

Arg2: DWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

The destination buffer must be at least 22 BYTEs large to allocate the output string Note:

(10 WIDE characters + ZTC = 22 BYTEs).

Procedure: UefiGetErrStrA

\ObjAsm\Code\ObjMem\64\UefiGetErrStrA.asm Files:

\ObjAsm\Code\ObjMem\Common\UefiGetErrStr\_TX.inc

Purnose: Return a description ANSI string from an UEFI error code.

Arguments: Arg1: UEFI error code.  $rax \rightarrow Error string.$ Return:

Procedure: UefiGetErrStrW

\ObjAsm\Code\ObjMem\64\UefiGetErrStrW.asm Files:

\ObjAsm\Code\ObjMem\Common\UefiGetErrStr\_TX.inc Return a description WIDE string from an UEFI error code. Purpose:

Arguments: Arg1: UEFI error code.  $rax \rightarrow Error string.$ Return:

Procedure: ugword2decA

\<u>ObjAsm\Code\ObjMem\64\uqword2decA.asm</u> Files:

\ObjAsm\Code\ObjMem\Common\uqword2decT64.inc Convert an unsigned QWORD into its decimal ANSI string representation. Purpose:

Arguments: Arg1: → Destination ANSI string buffer.

Arg2: QWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC. Return:

Note: The destination buffer must be at least 21 BYTEs large to allocate the output string

(20 ANSI characters + ZTC = 21 BYTES).

Procedure: ugword2decW

\ObjAsm\Code\ObjMem\64\uqword2decW.asm Files:

<u>\ObjAsm\Code\ObjMem\Common\uqword2decT64.inc</u>

Convert an unsigned QWORD into its decimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination WIDE string buffer.

Arg2: QWORD value.

eax = Number of BYTEs copied to the destination buffer, including the ZTC.

The destination buffer must be at least 42 BYTEs large to allocate the output string Note:

(20 WIDE characters + ZTC = 42 BYTES).

Procedure: UrlEscDecode

\ObjAsm\Code\ObjMem\64\URL.asm

Translate a wide string containig URL escape sequences to a plain wide string. Purpose:

Arguments: Arg1: → Input wide string.

Arg2: → Output Buffer.

Arg3: Output Buffer size in BYTEs.

eax = Number of chars written, including the ZTC.Return:

Procedure: UrlEscEncode

Translate a plain wide string to a wide string containig URL escape sequences.

Arguments: Arg1:  $\rightarrow$  Input wide string. Arg2:  $\rightarrow$  Output Buffer.

Arg3: Output Buffer size in BYTEs. eax = Number of chars written.

Return:

Procedure: UTF8ToWide

\ObjAsm\Code\ObjMem\64\UTF8ToWide.asm Files:

\ObjAsm\Code\ObjMem\Common\UTF8ToWide\_XP.inc

Convert an UTF8 byte stream to a WIDE (UTF16) string. Arg1:  $\rightarrow$  Destination WIDE buffer. Purpose:

Arguments:

Arg2: → Source UTF8 BYTE stream. Must be zero terminated.

Arg3: Destination buffer size in BYTEs.

Return: eax = Number of BYTEs written.

ecx = 0: succeeded 1: buffer full 2: conversion error

- The destination WIDE string is always terminated with a ZTC Notes:

(only if buffer size >= 2).

Procedure: WaitForProcess

\ObjAsm\Code\ObjMem\64\WaitForProcess.asm File:

Synchronisation procedure that waits until a process has finished. Purpose:

Arguments: Arg1: Process ID.

Arg2: Timeout value in ms.

eax = Wait result (WAIT\_ABANDONED, WAIT\_OBJECT\_0 or WAIT\_TIMEOUT) or -1 if failed. Return:

Procedure: WideToUTF8

\ObiAsm\Code\ObjMem\64\WideToUTF8.asm Files:

\ObjAsm\Code\ObjMem\Common\WideToUTF8\_XP.inc

Convert an WIDE string to an UTF8 encoded stream. Purpose:

Arguments: Arg1: → Destination buffer. Arg2: → Source WIDE string.

Arg3: Destination buffer size in BYTEs.

eax = Number of BYTEs written.Return:

ecx = 0: succeeded

1: buffer full

Notes: - The destination stream is always zero terminated.

Procedure: WndFadeIn

\ObjAsm\Code\ObjMem\64\WndFadeIn.asm File:

Fade in a window when WS\_EX\_LAYERED is set. Arg1: Window HANDLE. Purpose:

Arguments:

Arg2: Transparency start value.
Arg3: Transparency end value.
Arg4: Transparency increment value.
Arg5: Delay between steps.

Return: Nothing.

Procedure: WndFadeOut

File: \ObjAsm\Code\ObjMem\64\WndFadeOut.asm

Fade out a window when WS\_EX\_LAYERED is set.

Arguments: Arg1: Window HANDLE.

Arg2: Transparency start value. Arg3: Transparency end value. Arg4: Transparency decrement value. Arg5: Delay between steps.

Return: Nothing.

Procedure: word2hexA

File:

\ObjAsm\Code\ObjMem\64\word2hexA.asm
Convert a DORD to its hexadecimal ANSI string representation. Purpose:

Arguments: Arg1: → Destination ANSI string buffer. Arg2: WORD value.

Nothing. Return:

Notes: The destination buffer must be at least 5 BYTEs large to allocate the output string

(4 character BYTES + ZTC = 5 BYTES).

Procedure: word2hexW

\ObjAsm\Code\ObjMem\64\word2hexW.asm File:

Convert a WORD to its hexadecimal WIDE string representation. Purpose:

Arguments: Arg1: → Destination WIDE string buffer.

Arg2: WORD value.

Return: Nothing.

Notes: The destination buffer must be at least 9 BYTEs large to allocate the output string

(4 character WORDS + ZTC = 9 BYTES).