## Table of Contents

Table of Contents ..... 1
Getting Started ..... 24
Road Map ..... 24
Supported Document Types ..... 24
File Editing ..... 24
Macros ..... 24
Usage Comfort and Environment Customization ..... 24
Main Window Details ..... 24
Data Processing Engine ..... 25
Operation History ..... 25
Multiple Selection ..... 25
Multiple Cursors ..... 26
Operation Efficiency ..... 26
Multi-Document \& Multi-Editor Environment ..... 26
Encodings Support ..... 26
Native Support of x64 Platform ..... 26
Parallel Operation ..... 26
Programming Access ..... 26
Features ..... 26
What's New ..... 28
Version 7.21 ..... 28
Version 7.09 ..... 28
Version 7.01 ..... 28
Essential Updates ..... 28
New Features ..... 29
Updated Features ..... 29
Version 6.25 ..... 29
Version 6.20 ..... 29
Version 6.12 ..... 29
Version 6.01 ..... 29
Version 5.14 ..... 30
Version 5.13 ..... 30
Version 5.12 ..... 30
Version 5.11 ..... 30
Version 5.10 ..... 30
Version 5.01 ..... 31
Version 4.97 ..... 31
Version 4.96 ..... 31
Version 4.95 ..... 32
Version 4.94 ..... 32
Portable Installations Improvements ..... 32
Version 4.93 ..... 32
Version 4.92 ..... 32
General Bug Fixes ..... 32
General ..... 32
Structure Viewer ..... 32
Version 4.91 ..... 33
Transactional Installation ..... 33
Restart Manager Support ..... 33
Per-User Installation ..... 33
Portable Installation ..... 33
Command-line Parameters ..... 33
Version 4.85 ..... 34
Version 4.83 ..... 34
Version 4.82 ..... 34
Version 4.81 ..... 34
Version 4.76 ..... 35
Version 4.75 ..... 35
Version 4.73 ..... 35
Version 4.72 ..... 36
Version 4.71 ..... 36
Version 4.64 ..... 37
Version 4.51 ..... 37
Version 4.41 ..... 37
Version 4.33 ..... 37
Version 4.32 ..... 37
Version 4.21 ..... 38
Version 4.11 ..... 38
Version 4.01 ..... 38
Definitive Guide ..... 39
User Interface Elements ..... 39
Commands ..... 39
Main Menu ..... 39
Toolbars ..... 39
Keyboard Shortcuts ..... 39
Tool Windows ..... 40
Location ..... 40
Floating Tool Window ..... 40
Docked Tool Window ..... 40
Auto-Hidden Tool Windows ..... 41
Tool Windows as Tabbed Documents ..... 42
Tool Window Visibility ..... 42
Tool Window Configuration ..... 42
Window Switching ..... 42
Other Window-Switching methods ..... 43
Default Layout ..... 43
Choosing Another Default Layout ..... 43
Documents ..... 43
Supported Document Types ..... 43
Files and Streams ..... 43
HEX Files ..... 43
Working with Files in Shared Mode ..... 43
Volumes ..... 44
Disks ..... 44
Process Memory ..... 44
Scripting ..... 44
Opening Documents ..... 44
Errors Opening Files ..... 44
Opening Files ..... 44
File Open Command ..... 44
Using the Right-click Menu in the Windows Explorer ..... 45
Drag\&Drop ..... 45
Specify Files to Open in Command Line ..... 45
Scripting ..... 45
Opening Files in Shared Mode ..... 45
File Open Shared Command ..... 45
Using the Right-click Menu in the Windows Explorer ..... 45
Specify Files to Open in Command Line ..... 45
Scripting ..... 46
Opening Encoded Hex Files ..... 46
Scripting ..... 46
Opening Volumes ..... 46
Volume Navigator Support ..... 46
Scripting ..... 46
Opening Disks ..... 46
Scripting ..... 47
Opening Devices ..... 47
Opening Processes ..... 47
Creating New Documents ..... 47
Scripting ..... 47
Closing Documents ..... 47
Scripting ..... 47
Saving Documents ..... 48
Save As Command ..... 48
Save All Command ..... 48
Always Create Backups Option ..... 48
Requirements ..... 48
Scripting ..... 48
Working with Shared Files ..... 48
Supporting Scenarios ..... 48
Limitations ..... 48
Scripting ..... 48
Solving File Conflicts ..... 48
Rename Change Notification ..... 49
Delete Change Notification ..... 49
Modify Change Notification ..... 49
Highlighting Changes in the File ..... 49
Notes ..... 49
Working with Processes ..... 50
Process List Window ..... 50
Sorting, Filtering and Tree View ..... 50
Showing More Information ..... 50
Terminating Processes ..... 50
Opening Processes ..... 50
Scripting ..... 50
Process Window ..... 50
Process Exit Code ..... 51
Snapshots List ..... 51
Memory Layout ..... 51
Blocks ..... 51
Modules ..... 51
Integration with Disassembler ..... 52
Memory Map ..... 52
Scripting ..... 52
Process Snapshots ..... 52
Opening Snapshot Memory Regions ..... 52
Comparing Snapshots ..... 52
Launch External Debugger ..... 52
Save Snapshot Dump ..... 52
Editor Windows ..... 53
Basic Elements ..... 53
Scripting ..... 54
Encodings ..... 54
Working with Encodings ..... 55
UTF-8 Support ..... 56
Scripting ..... 56
Byte Order ..... 56
Effect on Floating-Point Types ..... 56
Data Inspector and Structure Viewer ..... 56
Scripting ..... 56
Simple Editing ..... 56
Cursor Movement and Navigation ..... 56
"Unlimited" Editing Space ..... 56
Multiple Cursors ..... 56
Keyboard Movement ..... 56
Navigating with Mouse ..... 57
Navigating with Touch ..... 57
Cursor Movement with Mouse ..... 57
Scripting ..... 57
Insert Mode ..... 57
Influence on Editor Commands ..... 57
Scripting ..... 58
Data Modification ..... 58
View Types ..... 58
Modifying File Data ..... 58
Scripting ..... 58
Editor Windows List ..... 58
Editor Windows Placement ..... 59
Selection ..... 59
Multiple Selection ..... 59
Single Selection Mode Option ..... 59
Selecting with Keyboard ..... 60
Scripting ..... 60
Selecting with Mouse ..... 60
Selection Tool Window ..... 60
Information ..... 60
Details ..... 60
Working with Selection ..... 60
Scripting ..... 61
Performance Considerations ..... 61
Editor Commands ..... 61
Pattern Window ..... 61
Byte Order ..... 61
Regular Expressions ..... 61
Scripting ..... 62
Fill ..... 62
Scripting ..... 62
Insert ..... 62
Scripting ..... 62
Delete ..... 62
Scripting ..... 62
Insert File ..... 62
Scripting ..... 62
Insert Hex ..... 63
Scripting ..... 63
Go to Offset ..... 63
Selection ..... 63
Scripting ..... 63
Go to Offset (Process) ..... 63
Change File Size ..... 64
Scripting ..... 64
Encrypt ..... 64
Scripting ..... 64
Decrypt ..... 65
Scripting ..... 65
Modify Bits ..... 65
Scripting ..... 65
Find and Replace ..... 65
Find \& Find All ..... 65
Find Window ..... 65
Find ..... 66
Find Operations ..... 66
Regular Expressions ..... 66
Scripting ..... 66
Find All ..... 66
Find All Results ..... 66
Regular Expressions ..... 66
Scripting ..... 66
Replace \& Replace All ..... 67
Find \& Replace Window ..... 67
Replace ..... 67
Find \& Replace Operations ..... 67
Regular Expressions ..... 67
Scripting ..... 67
Replace All ..... 68
Replace All Results ..... 68
Regular Expressions ..... 68
Scripting ..... 68
Find in Files \& Replace in Files ..... 68
Scalability ..... 68
Safety ..... 68
Regular Expressions ..... 68
Find in Files Window ..... 68
Folder List Window ..... 69
Find in Files Tool Window ..... 69
Find in Files ..... 70
Find at Least One Occurrence ..... 70
Find All Occurrences ..... 70
Regular Expressions ..... 71
Replace in Files ..... 71
Replace All Occurrences and Save ..... 71
Replace All Occurrences and Open ..... 71
Regular Expressions ..... 71
Regular Expressions ..... 71
Capturing Sub-expressions ..... 71
Usage Tips and Performance Considerations ..... 71
Regular Expressions Syntax ..... 72
Replace Pattern Syntax ..... 73
Examples ..... 73
Named Classes ..... 73
Recursive Grammars ..... 73
Creating and Managing Classes ..... 73
Regular Expression Syntax Errors ..... 74
Example ..... 74
Clipboard ..... 74
Supported Formats ..... 74
Internal Format ..... 74
Windows Explorer Format ..... 74
Hex Editor Neo's Document Data to File ..... 74
File Data to Hex Editor Neo's Document ..... 75
Text ..... 75
Scripting ..... 75
Copying, Cutting and Merge Modifier ..... 75
Important Considerations ..... 75
Advanced Copy \& Export ..... 75
Limitations ..... 75
Scripting ..... 75
Pasting Data ..... 75
Paste Command Complexity ..... 75
Scripting ..... 76
Pasting Text ..... 76
Pasting Raw Text ..... 76
Pasting Formatted Numbers ..... 76
Examples ..... 76
Scripting ..... 76
Advanced Copy \& Export ..... 76
Copy \& Export Tool Window ..... 76
Raw Text Format ..... 77
Formatted Data Format ..... 79
Encoded Data Format ..... 81
NTFS Streams ..... 81
Streams Support in Hex Editor Neo ..... 81
NTFS Streams Tool Window ..... 82
Find Stream Mode ..... 82
Searching for Streams ..... 82
Search Process and Results ..... 82
Operation History ..... 82
Organization ..... 83
Branches ..... 83
Pinned Branches ..... 84
History Tool Window ..... 84
Operation Tree Actions ..... 84
History Commands ..... 84
Purging History ..... 85
Saving and Loading History ..... 85
Saving History ..... 85
Loading History ..... 85
Operation History Tree Window ..... 85
Full Mode ..... 86
Forks Only Mode ..... 86
Navigation ..... 86
Legend ..... 86
Shortcut Menu ..... 86
File Comparison ..... 86
Compare Files Window ..... 86
File Comparison Tool Window ..... 87
Comparison Results Filters Window ..... 87
Operation ..... 87
Comparison Algorithms ..... 87
Simple Comparison Algorithm ..... 87
Difference Algorithm ..... 87
Patches ..... 87
Distribution ..... 87
Limitations ..... 87
Creating Patches ..... 88
Applying Patches ..... 88
PatchApply.exe ..... 88
Command Line Interface ..... 89
Patch API ..... 89
Pattern Coloring ..... 89
Pattern Coloring Tool Window ..... 89
Creating and Editing Rules ..... 89
Rule Schemes ..... 90
Regular Expressions ..... 90
Structure Viewer ..... 90
Support for Kaitai Struct ..... 90
What's New in Structure Viewer ..... 90
What's New in Structure Viewer (version 7.25) ..... 90
What's New in Structure Viewer (version 7.09) ..... 90
What's New in Structure Viewer (version 7.01) ..... 90
What's New in Structure Viewer (version 6.31) ..... 91
What's New in Structure Viewer (version 6.25) ..... 91
New Language Features ..... 91
What's New in Structure Viewer (version 6.01) ..... 91
New Language Features ..... 91
Fixed bugs ..... 91
What's New in Structure Viewer (version 5.12) ..... 91
New Language Features ..... 91
Fixed bugs ..... 91
What's New in Structure Viewer (version 5.01) ..... 92
New Language Features ..... 92
New UI Features ..... 92
What's New in Structure Viewer (version 4.96) ..... 93
What's New in Structure Viewer (version 4.92) ..... 93
What's New in Structure Viewer (version 4.71) ..... 93
Structure Library Tool Window ..... 94
Library Management ..... 95
Compilation ..... 95
Structure Editor ..... 95
Find ..... 95
Replace ..... 96
Go to Line ..... 96
Structure Editor Coloring ..... 96
Structure Binding ..... 96
Structure Viewer Tool Window ..... 97
Managing Bound Structures ..... 97
Working with Schemes ..... 98
Structure Viewer Settings Page ..... 98
Public Repository Synchronization ..... 98
Include Directories ..... 98
File Associations ..... 98
Language Reference ..... 98
Workflow ..... 99
Tokenization ..... 99
Comments ..... 99
Preprocessor ..... 99
\#include directive ..... 100
Using Absolute and Relative Paths ..... 100
\#pragma once Directive ..... 100
\#define Directive ..... 100
Defining Constants ..... 101
Defining Macros ..... 101
Variadic Macros ..... 102
\#undef Directive ..... 102
\#error Directive ..... 102
Preprocessor Operators ..... 102
\# Stringizing Operator ..... 102
\#\# Token-Pasting Operator ..... 102
Conditional Compilation Directives ..... 102
\#if, \#elif, \#else, \#endif, defined() operator ..... 103
\#ifdef, \#ifndef ..... 103
Predefined Macros ..... 103
Built-in Types ..... 104
Integer Types ..... 104
Type Modifiers ..... 104
Floating-Point Types ..... 104
String Types ..... 104
Expressions ..... 104
Operators ..... 105
Optimization ..... 105
Immediates ..... 105
Integer Number ..... 105
Floating-Point Number ..... 105
Character Constants ..... 106
String Constants ..... 106
References ..... 106
Limitations ..... 106
Byte Arrays ..... 106
Field Access ..... 106
this Pseudo-Field ..... 107
array_index Built-In Variable ..... 107
current_offset Built-In Variable ..... 107
Field Access Operator ..... 107
[] Array Indexing Operator ..... 108
() Expression Grouping Operator ..... 108
() Function Call Operator ..... 108

- Unary Minus Operator ..... 108
~ Bitwise NOT Operator ..... 108
\& Bitwise AND Operator ..... 109
^ Bitwise XOR Operator ..... 109
| Bitwise OR Operator ..... 109
! Logical NOT Operator ..... 109
\&\& Logical AND Operator ..... 109
|| Logical OR Operator ..... 109
sizeof() Operator ..... 110
\& Address-Of Operator ..... 110
* Multiplication Operator ..... 110
/ Division Operator ..... 110
\% Modulo Division Operator ..... 111
+ Addition Operator ..... 111
- Subtraction Operator ..... 111
\ll Left Shift Operator ..... 111
$\gg$ Right Shift Operator ..... 111
$\ggg$ Right Unsigned Shift Operator ..... 111
< Less Than Operator ..... 112
<= Less Than or Equal Operator ..... 112
> Greater Than Operator ..... 112
$>=$ Greater Than or Equal Operator ..... 112
== Equality Operator ..... 112
!= Inequality Operator ..... 112
?: Conditional Operator ..... 113
Functions ..... 113
Internal Functions ..... 113
Built-In Functions ..... 113
Examples ..... 114
Native Functions ..... 114
Function Scope ..... 115
Function Optimization ..... 115
External Functions ..... 115
Attaching Scripts ..... 115
Examples ..... 115
javascript Keyword ..... 115
External Functions ..... 115
Script Objects ..... 116
Example ..... 116
Statements ..... 116
if Statement ..... 116
switch Statement ..... 117
break Statement ..... 117
while Statement ..... 117
for Statement ..... 117
do...while Statement ..... 118
return Statement ..... 118
Scopes ..... 118
Constants and Constant Arrays ..... 119
Constant Arrays ..... 119
Variables and Variable Arrays ..... 119
Optimizations ..... 119
Using Variables ..... 119
Variable Arrays ..... 120
Enumerations ..... 120
Using Enumerations ..... 120
User-Defined Types ..... 120
Supported Types ..... 121
Structures ..... 121
Packing and Alignment ..... 122
Byte Order ..... 122
Unions ..... 122
Case Unions ..... 122
Case Union Optimization ..... 123
Forward Declarations ..... 123
Data Fields ..... 123
Plain Field ..... 123
Array Field ..... 124
Simple and Ordinary Arrays ..... 124
"Infinite" Arrays ..... 124
Visualization ..... 124
Bit Field ..... 124
Pointer Field ..... 125
Attributes ..... 125
Field Attributes ..... 125
Type Attributes ..... 126
Field Attributes ..... 126
noindex Attribute ..... 126
noautohide Attribute ..... 126
onread Attribute ..... 126
format Attribute ..... 126
description Attribute ..... 126
color_scheme Attribute ..... 126
exact_only Attribute ..... 126
Type Attributes ..... 127
display Attribute ..... 127
Typedefs ..... 127
Directives ..... 127
\$assert Directive ..... 127
\$print Directive ..... 128
\$break_array Directive ..... 128
\$bind Directive ..... 128
\$alert Directive ..... 128
\$revert_to Directive ..... 128
\$shift_by Directive ..... 129
\$remove_to Directive ..... 129
Format String Syntax ..... 129
Errors ..... 130
Examples ..... 132
Collaboration ..... 132
Disassembler ..... 132
Example ..... 132
Supported Instruction Sets ..... 132
Definitions ..... 132
Starting Disassembler ..... 133
Disassembler Window ..... 133
Symbols Window ..... 133
Disassembler View ..... 134
View Refresh ..... 135
Positioning within a Document ..... 135
Searching for a Text ..... 135
Exporting and Copying to Clipboard ..... 135
Data Annotations ..... 135
Data Annotations Tool Window ..... 135
Create/Modify Annotation ..... 135
Bookmarks ..... 136
Bookmarks Tool Window ..... 136
Active vs. Selected Bookmark Group ..... 136
Creating and Deleting Bookmarks Groups ..... 136
Working with Bookmarks ..... 136
Working with Active Bookmarks Group ..... 136
Working with Selected Bookmarks Group ..... 136
Other Commands ..... 137
Statistics ..... 137
Statistics Tool Window ..... 137
Histogram View ..... 137
Table View ..... 137
General Statistics ..... 137
Pattern Statistics ..... 138
Edit Pattern Window ..... 138
Regular Expressions ..... 138
Descriptive Statistics and Entropy Analysis ..... 138
Descriptive Statistics ..... 138
Entropy Analysis ..... 138
Attributes ..... 138
Attributes Tool Window ..... 138
Base Converter ..... 139
Base Converter Tool Window ..... 139
Data Inspector ..... 140
Fields ..... 140
Data Modification ..... 141
Byte Order ..... 141
Creating New Format ..... 141
Checksum Calculation ..... 141
Checksum Tool Window ..... 142
Displaying Results ..... 142
Adding, Removing and Restoring Algorithms ..... 142
Performance Considerations ..... 142
Configuring Custom CRC ..... 143
Configuring Custom FNV-1 ..... 143
Volume Navigator ..... 143
Features ..... 143
Volume Navigator Tool Window ..... 143
Volume Navigator Commands ..... 144
Built-In Explorer ..... 144
Explorer Tool Window ..... 144
Data Operations ..... 144
Bitwise Operations ..... 144
Bitwise Operation Window ..... 144
Using Patterns as Operands ..... 145
Alignment ..... 145
Scripting ..... 145
Arithmetic Operations ..... 145
Arithmetic Operation Window ..... 145
Alignment ..... 145
Scripting ..... 145
Shift Operations ..... 145
Shift Operation Window ..... 145
Alignment ..... 146
Scripting ..... 146
Case Change Operations ..... 146
Scripting ..... 146
Reverse Operations ..... 146
Reverse Bits ..... 146
Byte Swap ..... 146
Scripting ..... 146
Output Window ..... 146
Structure Errors ..... 146
Volume Errors ..... 147
Console ..... 147
Open Source Components ..... 147
Scintilla ..... 147
TypeScript ..... 147
ChakraCore ..... 148
Monaco Editor ..... 148
Boost ..... 148
yaml-cpp ..... 149
Eigen3 ..... 149
Capstone ..... 149
Macros ..... 150
Macro Support ..... 150
Editor Support ..... 150
Debugging ..... 150
Asynchronous Execution ..... 150
Awaiting Promises ..... 150
Scripts ..... 150
Running Scripts ..... 150
Working with Scripts ..... 151
Script Execution ..... 151
Debugging ..... 151
Break State ..... 151
Debug Watch Tool Window ..... 151
Debug Call Stack Tool Window ..... 152
Recording Macros ..... 152
Edit Macro Window ..... 152
Macros Tool Window ..... 152
Macro Playback ..... 152
API Reference ..... 152
Global Object ..... 152
Infrastructure Methods ..... 152
Creating New Documents ..... 153
Opening Documents ..... 153
Managing Opened Documents ..... 153
Working With Settings ..... 153
Find in Files, Replace in Files ..... 153
Working with Processes ..... 153
Events ..... 153
Global Object Reference ..... 153
Declaration ..... 153
IGlobal Properties ..... 154
apiVersion ..... 154
activeDocument ..... 154
activeView ..... 155
wholeFile ..... 155
documents ..... 155
activeProcessWindow ..... 155
IGlobal Methods ..... 155
log ..... 155
alert ..... 156
input ..... 156
async ..... 156
cancelAsync ..... 157
delay ..... 157
loadTextFile ..... 158
stopScript ..... 158
createPattern ..... 158
createPattern ..... 158
createPattern ..... 159
createPattern ..... 159
newDocument ..... 159
openFile ..... 160
openVolume ..... 160
openDisk ..... 161
openHexAsync ..... 161
activate ..... 162
closeAll ..... 162
displaySettings ..... 162
exportSettings ..... 163
importSettings ..... 163
restart ..... 163
exportConfiguration ..... 163
importConfiguration ..... 164
resetConfiguration ..... 164
findInFilesAsync ..... 164
replacelnFilesAsync ..... 165
cancelFindInFilesOperation ..... 165
findProcess ..... 165
findProcesses ..... 166
openProcess ..... 166
IGlobal Events ..... 166
activeDocumentChanged ..... 166
activeViewChanged ..... 166
Document Object ..... 167
IDocument Interface ..... 167
IDocument Properties ..... 167
name ..... 167
size ..... 167
views ..... 168
IDocument Methods ..... 168
saveAsync ..... 168
saveAsAsync ..... 168
undo ..... 168
redo ..... 169
cancel ..... 169
close ..... 169
View Object ..... 170
Cursor ..... 170
Selection ..... 170
Clipboard ..... 170
Data Modification ..... 170
Find ..... 170
Replace ..... 170
Data Operations ..... 171
IDocumentView Interface ..... 171
Declaration ..... 171
IDocumentView Properties ..... 173
document ..... 173
cursor ..... 173
selection ..... 173
insertMode ..... 173
displayAs ..... 173
addressDisplayAs ..... 174
groupBy ..... 174
columns ..... 174
currentColumns ..... 174
bigEndian ..... 174
encoding ..... 174
codePane ..... 175
textPane ..... 175
IDocumentView Methods ..... 175
activateCodePane ..... 175
activateTextPane ..... 175
editCell ..... 175
typeChar ..... 176
refresh ..... 176
connectWith ..... 176
disconnect ..... 176
lockUpdate ..... 177
cutAsync ..... 177
copyAsync ..... 177
pasteAsync ..... 178
clearAsync ..... 178
selectAll ..... 178
selectModifiedAsync ..... 178
copySelectionRangesAsync ..... 178
exportSelectionRangesAsync ..... 179
saveSelectionAsync ..... 179
loadSelectionAsync ..... 179
goto ..... 180
fillAsync ..... 180
insert ..... 180
insertFile ..... 180
insertHexAsync ..... 181
encryptAsync ..... 181
decryptAsync ..... 182
modifyBits ..... 182
findAsync ..... 182
findRegExpAsync ..... 183
findNextAsync ..... 183
findPrevAsync ..... 183
findAllAsync ..... 183
findAllRegExpAsync ..... 184
replaceAsync ..... 184
replaceRegExpAsync ..... 185
replaceAllAsync ..... 185
replaceAllRegExpAsync ..... 186
bitwiseOpAsync ..... 186
arithmeticOpAsync ..... 186
shiftOpAsync ..... 187
caseOpAsync ..... 187
reverseOpAsync ..... 188
Process Objects ..... 188
Process Object ..... 188
IProcess Interface ..... 188
Declaration ..... 188
IProcess Properties ..... 189
processld ..... 189
name ..... 189
commandLine ..... 189
IProcess Methods ..... 189
kill ..... 189
Process Window Object ..... 190
IProcessWindow Interface ..... 190
Declaration ..... 190
IProcessWindow Properties ..... 190
processld ..... 190
name ..... 190
commandLine ..... 191
path ..... 191
isRunning ..... 191
exitCode ..... 191
snapshots ..... 191
IProcessWindow Methods ..... 191
createSnapshot ..... 192
deleteSnapshot ..... 192
clearSnapshots ..... 192
compareSnapshots ..... 192
Process Snapshot Object ..... 192
IProcessSnapshot Interface ..... 193
Declaration ..... 193
IProcessSnapshot Properties ..... 193
isSnapshot ..... 193
date ..... 193
IProcessSnapshot Methods ..... 193
openRegion ..... 193
delete ..... 194
saveDump ..... 194
debug ..... 194
Support Objects ..... 194
IWindow Interface ..... 194
Declaration ..... 195
IWindow Properties ..... 195
name ..... 195
IWindow Methods ..... 195
close ..... 195
ICursor Interface ..... 195
Declaration ..... 195
ICursor Properties ..... 195
offset ..... 195
singleCursor ..... 196
all ..... 196
ICursor Methods ..... 196
move ..... 196
moveTo ..... 196
ISelection Interface ..... 197
Declaration ..... 197
ISelection Properties ..... 197
startOffset ..... 197
endOffset ..... 197
singleRange ..... 198
rangeCount ..... 198
size ..... 198
empty ..... 198
ISelection Methods ..... 198
selectRange ..... 198
invert ..... 199
clear ..... 199
IPattern Interface ..... 199
Declaration ..... 199
IPattern Properties ..... 199
empty ..... 199
size ..... 200
data ..... 200
AlertButtons Enumeration ..... 200
AlertIcon Enumeration ..... 200
AlertResult Enumeration ..... 200
CloseMode Enumeration ..... 201
FindInFilesOperation Enumeration ..... 201
Replace InFilesOperation Enumeration ..... 201
FindInFilesFlags Enumeration ..... 201
FindInFilesResult Object ..... 201
Declaration ..... 201
FindInFilesResult Properties ..... 201
filesSearched ..... 202
foldersSearched ..... 202
bytesProcessed ..... 202
filesMatched ..... 202
FindResult Object ..... 202
Declaration ..... 202
FindResult Properties ..... 202
offset ..... 203
size ..... 203
ActionFlags Enumeration ..... 203
CursorAction Enumeration ..... 203
GotoMode Enumeration ..... 203
SelectionUpdateMode Enumeration ..... 203
OpeningMode Enumeration ..... 204
OpenVolumeMode Enumeration ..... 204
Encodings Enumeration ..... 204
Default Encodings ..... 204
ANSI Encodings ..... 204
Arabic Encodings ..... 204
ISO Encodings ..... 204
IBM EBCDIC Encodings ..... 204
ISCII Encodings ..... 205
MAC Encodings ..... 205
OEM Encodings ..... 206
Asian Encodings ..... 206
Other Encodings ..... 206
FindFlags Enumeration ..... 207
ReplaceCommand Enumeration ..... 207
FillFlags Enumeration ..... 207
ValueType Enumeration ..... 207
HexFormat Enumeration ..... 208
InsertHexFlags Enumeration ..... 208
DisplayAs Enumeration ..... 208
AddressDisplayAs Enumeration ..... 208
GroupBy Enumeration ..... 208
SelectionLoadType Enumeration ..... 208
SelectRangeMode Enumeration ..... 208
BitwiseOperation Enumeration ..... 209
ArithmeticOperation Enumeration ..... 209
ShiftOperation Enumeration ..... 209
CaseOperation Enumeration ..... 209
ReverseOperation Enumeration ..... 209
OperationRange Enumeration ..... 209
OperandSize Enumeration ..... 210
SplitMode Enumeration ..... 210
DebuggerType Enumeration ..... 210
Scripting ..... 211
File Document Object ..... 211
Multiple Selection Object ..... 211
Sequence Object ..... 211
Thread Safety ..... 211
Lengthy Operations Limitations ..... 211
Document Object ..... 211
Creating Document Object ..... 211
Working with Document Object ..... 211
Initializing Document Object ..... 211
Closing Document Object ..... 211
Saving Changes ..... 211
Reading and Writing Data ..... 212
Inserting Data ..... 212
Deleting Data ..... 212
Filling Data ..... 212
Find and Find All ..... 212
Replace and Replace All ..... 212
Calculating Statistics ..... 212
Inserting Files ..... 212
Controlling File's Size ..... 212
Undo and Redo ..... 212
Working with Operation History ..... 212
Clipboard Methods ..... 212
Bitwise and Other Operations ..... 213
Regular Expressions ..... 213
Multiple Selection Object ..... 213
Creating Multiple Selection Object ..... 213
Working with Multiple Selection Object ..... 213
Sequence Object ..... 213
Creating Sequence Object ..... 213
Working with Sequence Object ..... 213
Parser Object ..... 213
Obtaining Parser Object ..... 213
Working with Parser Object ..... 214
Reference ..... 214
Document Object ..... 214
IFileDocument Interface ..... 214
Declaration ..... 214
IFileDocument Properties ..... 218
AutoCreateBranch ..... 218
FileName ..... 218
FileSize ..... 218
Modified ..... 219
Stream ..... 219
ReadOnly ..... 219
RegExpSyntaxError ..... 219
CanRedo ..... 219
CanUndo ..... 220
CanPaste ..... 220
IFileDocument Methods ..... 220
Close ..... 220
New ..... 221
Open ..... 221
OpenPhysicalDisk ..... 221
OpenProcess ..... 222
OpenVolume ..... 222
Save ..... 222
SaveAs ..... 223
GetModifiedSel ..... 223
Decrypt ..... 223
Delete ..... 224
DeleteMulti ..... 224
Encrypt ..... 225
Fill ..... 225
FillByte ..... 226
FillByteMulti ..... 226
FillMulti ..... 227
Insert ..... 227
InsertByte ..... 228
InsertPattern ..... 228
PutFile ..... 228
Read ..... 229
Write ..... 229
CreatePatch ..... 230
LoadHistory ..... 230
Purge ..... 230
Redo ..... 231
SaveHistory ..... 231
Undo ..... 231
Copy ..... 231
Cut ..... 232
Paste ..... 232
Paste2 ..... 233
Paste3 ..... 233
PasteText ..... 233
PasteText2 ..... 234
Find ..... 234
FindRegExp ..... 235
FindRegExp2 ..... 236
FindAll ..... 236
FindAllRegExp ..... 237
FindAllRegExp2 ..... 237
Replace ..... 238
ReplaceAll ..... 239
ReplaceAllRegExp ..... 239
ReplaceAllRegExp2 ..... 240
ReplaceAllRegExpWithRegExp ..... 241
ArithmeticOp ..... 241
BitwiseOp ..... 242
CaseOp ..... 242
ShiftOp ..... 242
ReverseOp ..... 243
GatherStatistics ..... 243
PatternStatistics ..... 243
PatternStatisticsRegExp ..... 244
PatternStatisticsRegExp2 ..... 245
CreateEmptySelection ..... 245
CreateSequence ..... 246
BitwiseOpS ..... 246
FillMultiS ..... 246
FillS ..... 247
FindAllS ..... 247
FindS ..... 248
InsertPatternS ..... 248
InsertS ..... 249
ReadS ..... 249
ReplaceAllRegExpS ..... 250
ReplaceAllRegExp2S ..... 250
ReplaceAllS ..... 251
ReplaceS ..... 251
ToNumber ..... 252
WriteS ..... 252
PatchOptions Enumeration ..... 253
HistoryPurgeLevel Enumeration ..... 253
TextType Enumeration ..... 253
Paste3Flags Enumeration ..... 253
RegExpFlags Enumeration ..... 253
ArithmeticOpType Enumeration ..... 253
OperandSizeType Enumeration ..... 253
BitwiseOpType Enumeration ..... 253
CaseOpType Enumeration ..... 254
ShiftOpType Enumeration ..... 254
ReverseOpType Enumeration ..... 254
Multiple Selection Object ..... 254
IMultiSelection Interface ..... 254
Declaration ..... 254
IMultiSelection Properties ..... 255
Count ..... 255
Empty ..... 255
TotalSize ..... 255
IMultiSelection Methods ..... 256
AddRange ..... 256
Clear ..... 256
DeleteRange ..... 256
InsertRange ..... 257
Invert ..... 257
Isln ..... 257
Load ..... 258
RemoveRange ..... 258
Save ..... 259
ToggleRange ..... 259
LoadMode Enumeration ..... 259
Sequence Object ..... 260
ISequence Interface ..... 260
Declaration ..... 260
ISequence Properties ..... 260
Empty ..... 260
Length ..... 260
Value ..... 260
ISequence Methods ..... 261
AddData ..... 261
AddText ..... 261
Clear ..... 261
Copy ..... 261
CreateSubSequence ..... 262
Remove ..... 262
SequenceDataType Enumeration ..... 262
Parser Object ..... 262
IParser Interface ..... 262
Declaration ..... 262
IParser Methods ..... 263
abort ..... 263
alert ..... 263
bind ..... 263
eval ..... 264
print ..... 264
add_coloring_scheme ..... 265
Customization ..... 266
Settings Window ..... 266
Scripts ..... 266
Visual ..... 266
Themes Settings ..... 266
User Interface ..... 267
Language Settings ..... 267
Online Languages ..... 267
Keyboard Settings ..... 267
Keyboard Map ..... 268
Commands Settings ..... 268
Creating New Toolbar ..... 269
Deleting Toolbar ..... 269
Configuring a Toolbar ..... 269
Other Options ..... 269
Network \& Support ..... 269
Beta Settings ..... 269
Crash Reporting Settings ..... 269
Crash Reporting Configuration ..... 269
Crash Report Processing ..... 269
Proxy Settings ..... 270
Server Settings ..... 270
Proxy Server Authentication ..... 270
Security Considerations ..... 270
Binary Editor ..... 270
Tasks Settings ..... 270
Regular Expressions Settings ..... 271
Editor Settings ..... 271
Editor Schemes ..... 272
Configurable Options ..... 272
Default Values ..... 272
Data Annotations Settings ..... 272
Other ..... 273
General Settings ..... 273
General Group ..... 273
Editor Group ..... 274
Notifications Group ..... 274
NTFS Streams ..... 274
Printing ..... 274
Copy\&Export ..... 274
Disassembler ..... 274
Process ..... 275
Structure Viewer ..... 275
Purchasing Hex Editor Neo ..... 276
Choosing Appropriate Edition ..... 276
Choosing Additional Options ..... 276
Lifetime Upgrades Service ..... 276
How to Buy ..... 276
Online Orders ..... 276
Offline Orders ..... 276
Purchase Orders ..... 276
Ordering from Local Resellers and Distributors ..... 276
My Account Web Site Section ..... 276
Downloading License Files ..... 277
Installing License Files ..... 277
Upgrading Hex Editor Neo ..... 278
Upgrading from the Free Edition ..... 278
Upgrading from Standard or Professional Editions ..... 278
Getting Support ..... 279
How to Get Support ..... 279
Built-In Updater ..... 280
Component Updates Window ..... 280
Settings Manager ..... 281
Exporting \& Importing Settings ..... 281
Settings Page ..... 281
Settings Management ..... 281


## Getting Started

## Road Map

This documentation provides an in-depth description of Hex Editor Neo features. It is structurally divided into several sections. TheGetting Started with Hex Editor Neo section provides generic information about the product, briefly describes its features and technologies. The Hex Editor Neo Definitive Guide section provides detailed description of every Hex Editor Neo feature. The File Document Scripting section describes the programming interface provided by the product.


## Supported Document Types

Hex Editor Neo supports editing files, shared files, volumes, disks and process memory.

## File Editing

Hex Editor Neo is an advanced file editor. Numerous powerful features, such as Pattern Coloring, Structure Viewer, Bookmarks as well as highly-optimized commands can be utilized to make file editing task quick and simple.

Any loaded or created file is called a document in the editor. There is at least one editor window for each document. For convenience, multiple editor windows may be opened for a single document, each having its own properties (view type, encoding, cursor position and selection). Hex Editor Neo provides a very flexible window organization tools, including window splitting and separating the workspace with frames. A huge set of supported encodings allows you to interpret (and edit) document's data in different ways.

Multiple cursors feature simplifies mass-editing tasks by allowing several editor commands to operate on several document locations simultaneously.
Data encryption and decryption functions are provided, utilizing any installed block or stream cipher.
Large set of data operations allow you to modify data in large blocks.
Hex Editor Neo fully supports NTFS alternate data streams. Several special tools are provided that simplify working with streams and even locating them on your computer.
Powerful and flexible multiple selection concept is used throughout the Hex Editor Neo. A multiple selection is a collection of contiguous ranges of selected document data. Such selection may arise as a result of user's action or as a result of executing a command. All Hex Editor Neo's commands and modules fully support multiple selections and work with them in a very efficient way.

A full "basic" editing command set is supported by the editor: you can modify data, insert data, delete data and change file's size. In addition, advanced editing commands, such as Fill and Insert File are provided.

Hex Editor Neo provides the most powerful and flexible solution when it comes to searching and replacing data in files: the Find command is used to search for a pattern and Replace command is used to search for a pattern and then replace it with another pattern. The size of the search and replace pattern may differ and the replace pattern may even be empty. Full support for regular expression is also provided.

Even more powerful Find All command is capable of locating all pattern occurrences and returning them as a selection. It's needless to say that selection is fully browseable and can be saved to a file or exported as a text file.

Yet more powerful Replace All command does it all in one step: locates all occurrences of the pattern in a file (or a selection - all Hex Editor Neo commands support multiple selections) and replaces all of them with another pattern. Again, pattern sizes may differ and replace pattern may be empty. The latter case is equivalent to using the Find All command and then the Delete command with only difference that Replace All command operates much faster!

Find in Files and Replace in Files commands do the same for a group of files. These commands are extremely powerful and provide a flexible setup. They are also capable of utilizing the full power provided by the multi-core and multi-processor computers.

Other advanced Hex Editor Neo features are Pattern Coloring, Structure Viewer, Bookmarks, Operation History, Statistics, Attributes, Data Inspector and Base Converter, Patch Creation, standard Clipboard support and advanced Copy \& Export feature. Checksum Calculation feature provides a lot of checksum calculation algorithms.

## Macros

Hex Editor Neo supports in-product scripting (in addition to File Scripting Interface, which allows external applications to utilize the power of the editor's engine), allowing you to automate most editing tasks. There's also a Record Macro command that automatically records all user actions as a script. The recorded macro may later be played back, applying the same actions either to the same or completely different document

## Usage Comfort and Environment Customization

A number of user interface features, such as toolbars and tool windows are all called to simplify most everyday's tasks and make editing faster and simpler.
Every aspect of the user interface is customizable. More detail is provided in corresponding documentation sections.

## Main Window Details

Below is a screenshot of the main Hex Editor Neo window:


All important elements are numbered and are described below:

## Main Menu

Hex Editor Neo's main menu contains all supported commands. See the main menu section for the detailed information on using the main menu.

## Main Toolbar

Main toolbar provides a quick access to a subset of Hex Editor Neo's commands. The toolbar is fully customizable. You can also create additional toolbars.
See the toolbars section for the detailed information on using and customizing toolbars.

## Tool Windows

Hex Editor Neo provides most of its powerful features by means of Tool Windows. These features include Pattern Coloring, Structure Viewer, Bookmarks, Operation History and much more.
See the Tool Windows section for the detailed information about tool windows.

## Editor Windows

Editor windows display the documents, loaded into the Hex Editor Neo. They allow you to view and edit the document's content.
Detailed information on editor windows, their features, usage and customization is provided in the Editor Windows section.

## Context Menus

Context or shortcut menus are used extensively in the editor. Right-clicking or pressing the corresponding keyboard button (or Shift+F10) bring up a context menu with a list of commands, which are most relevant in the current situation.

## Status Bar

A status bar displays the current cursor's offset, current file's size, a view type of a current editor window and a number of indicators. When a command is selected in a main menu, the command's description is also displayed on the status bar. Status bar is described in more detail throughout the documentation.

## Data Processing Engine

Hex Editor Neo implements the unique data processing engine which endows it with a number of interesting and powerful features.
The main characteristics of the data engine include:

## Operation History

Hex Editor Neo implements an unlimited operation history with branches and immediate switching between operations. Every command that modifies the state of the document (overwrites existing data, inserts new data, deletes document's data or changes the size of the document) creates a new operation and adds it to the operation history.
The amount of RAM required to store operation's data is minimal and is much less than corresponding memory amounts required by most competitive products. In addition, operation's data is automatically compressed and moved to disk when not required. The required data is automatically loaded and decompressed on-demand.

Branches feature allows you to create several operation sequences that "grow" from a common "root" operation and represent different variants of the document. All such sequences are "equal in rights": you can switch between them at any time, copy data from one branch to the Clipboard in order to paste it into another branch and so on.

Switching between operations is performed immediately.
The entire operation history can be saved to a file and later loaded for the same document. Imagine you begin working with a file. You open it in the Hex Editor Neo, perform several operations on it, create branches and so on... Then you save the operation history to a file and close the edited file without saving changes. Now you have an option of opening the same file and loading saved operation history to continue your work from the same point.

The Operation History section in this documentation provides you with a detailed description.

## Multiple Selection

All Hex Editor Neo commands are capable working with a multiple selection. A multiple selection is a collection of contiguous ranges. A multiple selection may arise as a result of user actions, or as a result of the Find All command. It can also be saved to a file and later loaded from it.

Algorithms, implemented in the Hex Editor Neo, provide an extremely efficient way to store and process multiple selections. This makes, for example, executing the Find All command for a several gigabytes-long file quite an ordinary operation that will not "eat up" all available virtual memory and disk space, as it happens in most competitive binary file editors.

The "native" support of multiple selections by every Hex Editor Neo's command, starting from Fill and Clipboard to Statistics makes this instrument a valuable and helpful editing tool in everyday's work.

The Multiple Selection section provides in-depth description of this feature.

## Multiple Cursors

Editor Windows also support the concept of a multiple cursors. In addition to a default cursor, the user may create any number of additional cursors. Several editor commands, like data modification, Clipboard operations and several others respect multiple cursors, providing with even richer editing set.

## Operation Efficiency

Due to unique characteristics of the Hex Editor Neo's data processing engine, most Hex Editor Neo commands complete in constant-time. In other words, they have a constant-time complexity. Constant-time complexity means that a command operates in approximately constant time and this time does not depend on latent or obvious command's complexity. For example, Insert File command has a constant-time complexity. This means, that inserting a 1 KB file takes exactly the same time as inserting a 1 GB file.

Other Hex Editor Neo commands complete in linear-time. In other words, they have a linear-time complexity. The time required to process such command usually linearly depends on the file's size, selection's complexity or some other factor. None of Hex Editor Neo commands has worse than linear-time complexity.

Most linear-time commands depend on the selection's complexity, that is, number of ranges in a selection. That is, for single-range selections these commands have a constant-time complexity. For example, the Fill command has a linear-time complexity, depending on selection's complexity. That is, if the command is used to fill a singlerange selection, it operates in constant-time: filling 1 KB file takes exactly the same time as filling a 1 GB file. Filling a selection that has 20 million ranges will take 20 times longer than filling a selection that has 1 million ranges

This documentation always states the command's complexity when it describes a given command. When linear-time complexity is mentioned, a dependent factor is given.

## Multi-Document \& Multi-Editor Environment

Multiple documents (files) may be opened for editing in the Hex Editor Neo. They all have their own operation history, which are not related with each other. Multipleeditor windows may be opened for each document, each of which also has its own state: view type, cursor position, selection and encoding.

## Encodings Support

Each editor window has two panes: code pane and text pane (the panes are configurable). Text pane is displayed when the editor is in BYTE or worD grouping mode. For the latter, text pane automatically displays UNICODE (UTF-16-encoded) characters. For the first mode, you can select one of over 130 supported encodings, including UTF-8.

After that, you get a full support for the selected encoding - the editor will correctly parse and display encoded characters and automatically encode characters pressed on the keyboard or pasted from the Clipboard.

## Native Support of x64 Platform

Hex Editor Neo's data processing engine natively supports the x64 platform and utilizes all features provided by it, including extended register list, larger address space, new processor commands and full support for SSE2 and AVX instructions.

Hex Editor Neo's installer automatically installs correct binaries (installation package contains compiled binaries for $\times 86$ and x64 platforms).

## Parallel Operation

Starting from version 5.01, Hex Editor Neo utilizes multiple processors or cores when it performs complex lengthy operations. It also allows running multiple lengthy operations on several opened documents in parallel.

## Programming Access

Hex Editor Neo provides two kinds of programming accesses: External Scripting and In-Product Scripting.
External Scripting provides programming interface that can be used by external code to access all powerful features of the Hex Editor Neo. This interface is implemented by an ActiveX component. When an external code uses functions provided by the component, the Hex Editor Neo's user interface is not required and not displayed.

The Scripting section contains in-depth description of the programming interface. Interface overview and detailed reference with a lot of sample codes in different programming languages, including $\mathrm{C}++$, JavaScript and $\mathrm{C} \#$, are provided.

Internal Scripting is a set of programming interfaces exposed by internal application components and allows the user to create macros that control the application. This includes an ability to create, manage and close documents, do basic editing tasks, execute editor commands and a lot of other things. A Macros » Start Recording command starts automatic recording of all user actions. As a result of this operation, a new macro is created. The user may edit the macro script, and also provide an application-wide hot key for this macro.

## Features

This section contains a comprehensive list of Hex Editor Neo features. It also contains links to other parts of this documentation that describe a given feature in more detail.

## Instantly opening files of any size

Hex Editor Neo does not force any limit on file's size. Any file supported by the underlying file system (FAT, FAT32 or NTFS) is supported.

## Working with shared file

Hex Editor Neo may share edited files with other applications. Should another application make a change to a shared file, Hex Editor Neo detects it and allows you to either discard a change or to merge it with changes in the editor. Automatic file compare may be used to find differences between the current and previous version. As an option, all previous versions of a file may be kept by the editor and used in your work.

## Opening volumes (logical disks)

Hex Editor Neo supports opening volumes (logical disks) for viewing and editing

## Opening physical disks

Hex Editor Neo supports opening physical disks for viewing and editing.

## Working with running processes

Hex Editor Neo allows you to view the information about running processes, open processes to investigate their memory layout and create light-weight process memory snapshots. Memory for a running process or any process snapshot may be opened for read-only and read/write access and two snapshots may be compared using the built-in File Comparison module. Application also supports launching external debugger for a selected process snapshot or saving a snapshot dump.

## Full support for NTFS alternate data streams

Hex Editor Neo fully supports NTFS alternate data streams. A special NTFS Streams Tool Window displays all streams for the currently active file, allows you to delete or rename them, open them for editing or create new streams. In addition, the powerful Find Stream feature allows you to locate all streams on your computer.

## Multiple Selection

Hex Editor Neo widely uses the concept of multiple selection. All Hex Editor Neo commands are fully compatible with multiple selection.
Multiple Cursors

Editor Windows allows settings alternate cursors in addition to a default one, allowing several editor commands to execute on multiple document locations simultaneously.

## Selection Saving, Loading and Exporting

Selection object of any complexity can be compressed and saved to the file. Later, you may load this selection and/or merge it with an existing selection using different algorithms. In addition, all ranges in a selection may be copied to the Clipboard or saved to a text file. See the Working with Selection topic for more information.

## Basic File Editing (write, fill, insert, delete, put/insert file, Clipboard operations, change file size, modify bits)

All these commands are described in detail in the Editor Commands documentation section.

## Data encryption and decryption

A whole document or a part of it may be encrypted and decrypted, using any installed block or stream cipher.

## Data Operation

A large set of data operation (bitwise, arithmetic, shifts, case change and bit/byte reversing) are provided.

## Text Clipboard Operations

In addition to basic Clipboard operations (moving data blocks between the same or different documents or exchanging file data with Windows Explorer); Hex Editor Neo is capable of pasting text from the Clipboard. Text may be pasted "as is" or parsed as a sequence of numbers (decimal, hexadecimal, octal or binary). Parsed numbers are then put into the document

## Different encodings

Hex Editor Neo supports more than 130 different encodings, configurable separately for each editor window. They are used to interpret and edit document's data in different formats. Starting from version 4.95, UTF-8 encoding is supported as well.

## Find and Find All

Hex Editor Neo provides an extremely efficient implementation of Find and Find All commands. Full support for ECMAScript-compatible regular expressions is provided. The result of a Find All command is a multiple selection - a browseable and easily editable object.

## Replace and Replace All

Hex Editor Neo provides an extremely efficient implementation of Replace and Replace All commands. Full support for ECMAScript-compatible regular expressions is provided. Replace pattern may also be a regular expression.

## Find in Files \& Replace in Files

Hex Editor Neo provides you with a simple and effective mechanism of pattern searching and replacing in files residing in different folders. This feature is described in more detail in the Find in Files and Replace in Files sections. Include streams option may be used to add alternate data streams to a search list. Full support for ECMAScript-compatible regular expressions is provided.

## Unlimited Undo \& Redo steps

In fact, a more advanced Operation History management is provided by the Hex Editor Neo.

## Operation history browsing

Hex Editor Neo allows you to instantly switch to any history operation.

## History Save \& Load

An entire document's operation history may be compressed and saved to a file and later loaded from it, effectively restoring all operations performed on the document.

## Data Annotations

The user may provide additional text annotations or comments and associate them with specific locations in the edited document. These comments are later saved with a document.

## File comparison

Hex Editor Neo provides two file comparison algorithms, which are fast and accurate and can be used to compare text and binary data files. The feature may also quickly and efficiently compare contents of two process snapshots.

## Disassembler

Hex Editor Neo includes a disassembler module. The current implementation is capable of disassembling x86, x86/x64, MSIL, ARM, ARM64, MIPS, PowerPC, Sparc, SystemZ, XCore, 68K, TMS320C64x, 680X, MOS65XX and RISC-V architectures and instruction sets. Disassembler supports PE and ELF executables and automatically searches for symbol files and uses the information from them to increase the assembly output readability. The Disassembler View may be connected to ordinary editor window which provides you with another handy data viewing and manipulation tool. The disassembler module works with files and processes' memory and is even automatically updated when you modify the document.

## Patch creation

Hex Editor Neo supports creation of a patch, which represents the changes made to a document. The patch is saved to the file in a compressed format. A patch may then be freely distributed and applied on the same or another computer. A patch may be packaged inside the executable file. A Patch API is also provided to allow programmatic patch installation.

## Editor Window customization

All aspects of an editor window are fully customizable. Several predefined schemes are shipped with a product.
Displayed data may be grouped as bytes, words, double or quad words and represented as hexadecimal, decimal, octal, binary integer numbers as well as single or double precision floating-point numbers.
Document's data may be represented in one of the listed view types.

## Fully customizable user interface

Hex Editor Neo's interface is fully customizable. You can customize toolbars, tool windows, keyboard shortcuts and editor windows. You even have an option of choosing one of three window switching mechanisms.

## Connecting Editor Windows

You can connect two editor windows with each other. This synchronizes cursor movement in both windows, which is an extremely useful editing technique in some scenarios.

## Two Built-In Explorer Windows

Hex Editor Neo hosts two Explorer windows. Use them to browse your file system and network neighborhood. All standard operations are supported, including Clipboard operations and Drag\&Drop. Explorer windows are tightly integrated with an editor and Find in Files feature

## File attributes

The Attributes Tool Window displays current file's attributes and allows you to modify them. It also displays the total number of streams in a file, size of all streams and total size of the file, including its unnamed and named streams.

## Pattern Coloring

Hex Editor Neo can be configured to automatically highlight given patterns in its editor windows. See the Pattern Coloring section for more information. Full support for ECMAScript-compatible regular expressions is provided.

## Data Inspector

Data Inspector Tool Window interprets data under the cursor in several formats. In addition, it supports highlighting data boundaries and allows data modification.

## Base Converter

Base Converter Tool Window provides tools to convert data from one format or encoding into another.

## Bookmarks

Hex Editor Neo's bookmarks provide you with an extremely flexible way of document navigation. You can define any number of bookmarks groups, exchange data between a bookmarks group and selection, save and load bookmarks and so on. See the Bookmarks section for more information.

## Structure Viewe

The powerful Structure Viewer feature allows you to define data structures and bind them to specific locations in edited files. Full support for Kaitai structures is also provided. See the Structure Viewer section for more information

## Statistics

Hex Editor Neo contains tools that can be used to quickly compute several file statistics. See the Statistics section for more information

## Advanced Copy \& Export

Hex Editor Neo provides an advanced data conversion tool. Selected binary data from the editor may be processed in a number of formats to produce a text output to be placed into the Clipboard or written to text file. Data may be converted into character string (more than 150 different encodings are supported); packaged as a sequence of numbers (to be put into spreadsheet application, for example) or as programming language array declaration ( 7 popular programming languages supported). In addition, data may be encoded in Base64, UUencode, Quoted-Printable, Intel HEX or Motorola S-Records formats.

## Checksum Calculation

Hex Editor Neo supports more than twenty checksum calculation algorithms, including MD5, SHA-1, CRC-16, CRC-32, FNV-1(a) and others. Calculation of CRC and FNV-1 algorithm with custom parameters are also provided.

## Automatic updates

Hex Editor Neo can be configured to automatically check for released updates, display a list of changes, download and install updates (by your request)
Macros
Most application components expose programming interfaces, allowing them to be used by scripts running inside the Hex Editor Neo, allowing the user to record repetitive tasks as a macro and execute them at a later time.

## Built-in Debugger

Built-in debugger simplifies the development of scripts, providing the step execution functionality, breakpoints, Watch Tool Window and Call Stack Tool Window.

## File Document Scripting

Hex Editor Neo exposes its root functionality for external programs through automation-compatible ActiveX library. See the Scripting section for more information.

## Native $\mathbf{x} 64$ platform support

Hex Editor Neo natively supports an x64 platform. A number of basic algorithms have been revised and optimized to use the powerful features available on this platform, including larger virtual address space, native wider registers and new processor commands.

## Native support for high-DPI displays

Hex Editor Neo fully supports high-DPI displays. All Hex Editor Neo's user interface elements, including texts and icons, are properly scaled. As long as all graphics elements are rendered using vector technology, there are no artifacts on high-DPI displays. In addition, Hex Editor Neo allows you to change the size of icons on the main toolbar in a wide range.

## Theming support

Colors of all application components, including the menu, toolbars, editor windows components are fully customizable and the user can create named themes. Hex Editor Neo provides several themes, including the "Automatic" one, which adopts to the current system settings, with full support for Windows Dark Mode.

## Native support for Windows 10 and Windows 11 Dark Mode

Hex Editor Neo fully supports the dark mode introduced in Windows 10 and Windows 11

## Support of Windows 11

Hex Editor Neo completely supports Windows 11 and its updated visual look, including rounded windows corners, Mica material and improved font rendering.

## Support for Intel HEX and Motorola S files

File» Open » Open Hex... and Edit» Insert Hex... commands allow the user to convert Intel HEX and Motorola S files to binary form and insert them into the Hex Editor Neo. File» New » New Intel Hex and File » New » New Motorola S-Records commands allow the user to create empty HEX files. HEX files may also be directly opened in the editor.

## Volume Navigator

Built-in support for parsing NTFS, FAT, ExFAT and CDFS volume structures, opening file and directory records and data locations. Ability to copy files, directories or entire volumes.

## Language Packs

Hex Editor Neo's user interface provides full support for localization. Application automatically discovers published language packs and allows the user to download and install them.

## What's New

## Version 7.21

This version features an improved File Comparison module with support for filtering and sorting, improved performance and accuracy.
New Processes Tool Window is added that displays an interactive list of all running processes, allows opening them to retrieve additional information. A light-weight process snapshot may be created for a running process to "freeze" the contents of process address space at a given point of time with an ability to view the memory region in a snapshot, compare snapshots, launch external debugger or save dump of a snapshot.

Full scripting support (with recording) for this new feature is provided, allowing macros to automate all process-related tasks.

## Version 7.09

This version provides an updated disassembler with support for PE and ELF executables and the following architectures: x86, x86/x64, MSIL, ARM, ARM64, MIPS, PowerPC Sparc, SystemZ, XCore, 68K, TMS320C64x, 680X, MOS65XX and RISC-V.

It also adds synchronization with an online structure definition repository.

## Version 7.01

This is a new major version release that brings a lot of new features and improvements.

## Essential Updates

This version features a complete redesign of the application, bringing new visual styles for menus, toolbars, tool windows and completely refreshed icon set. Application now has full support for high-definition displays.

Color themes provide the way to set colors of each visual element and a few predefined color themes are provided. The default, "automatic" color theme automatically adapts to current OS visual settings, and fully supports Windows Dark Mode.

New version provides completely new implementation of the editor window. It features modern graphics and text rendering technologies and provides best visual quality with the help of GPU.

## New Features

## Macros

Several internal application components now expose programming interfaces, allowing the internal scripts that use them to be created, edited directly in Hex Editor Neo and executed. The built-in debugger is provided to simplify script development
Named macros may be associated with scripts and executed on demand.

## Multiple cursors

Editor windows now support the concept of multiple cursors, allowing the user to add additional cursors providing multiple editing locations for various editor commands, including typing and Clipboard.
New sample structures
Structure Viewer sample structure set has been extended with new structures for QOI, MessagePack and webp formats.

## Support for Kaitai structures

Structure Viewer now natively supports Kaitai structure definitions.

## Data Annotations

The user may now provide text annotations or comments for various document positions or ranges. These annotations are stored with the document and displayed in the editor window.

## Tool window toolbar customization

The user may now customize the toolbar of any tool window, in addition to main window toolbars.

## Updated Settings Page

This release features an updated Settings page with improved page layout and built-in Search. The search function provides localized search support if corresponding language pack is installed.

## Updated file opening behavior

The File » Open » Open File... command now automatically selects the way to open a file depending on its extension. By default, it opens .ts, .js files as scripts, .hex files as Intel HEX and .s19,.s28 and .s37 as Motorola S-Records files. Other files are opened for binary editing. The user has an option to override the default behavior by choosing the corresponding file type in the Open File Window.

## Updated Features

## Improved search in process

When searching for the pattern in an opened process, only allocated regions are now searched

## Structure Viewer enhancements

using declaration may be used to create type aliases, in addition to previously supported typedef declaration

## New File Systems support in Volume Navigator

Volume Navigator adds native support for FAT, FAT32, ExFAT and CDFS file systems in addition to previously supported NTFS file system.

## New checksum algorithms

FNV-1, FNV-1a and FNV-1 with custom parameters have been added to the list of supported checksum algorithms.

## Editor window layouts

Custom editor window layouts are now associated with a specified regular expression that matches the full document path, not the individual document, as previously.

## Improved HEX files support

Previously, the File» Open» Open Hex... command was implemented by creating a new document and executing the Edit » Insert Hex... command. The user then had to use the Advanced Copy \& Export function if he wanted to save the changes back to the HEX file. Now the editor opens such files directly. If the user saves changes made to a HEX file, editor uses the original file format. The user also has an option to change the output format using the File » Save As... command. In addition, the File » New » New Intel Hex and File » New » New Motorola S-Records commands may be used to create empty HEX files.

## Version 6.25

This version adds a few new directives and attributes to the Structure Viewer language, adding look-ahead support to structure binding.

## Version 6.20

Version 6.20 introduces the Settings Manager which centralizes application settings management.

## Version 6.12

This version adds support for creating volume snapshots and opening files or entire volumes from these snapshots.

## Version 6.01

This is a major release. This version introduces the following new modules, technologies and updates to current features:

## Updated Data Inspector

This releases updates Data Inspector, adding support for custom types.

## Updated Structure Library

Structure Viewer has been updated significantly, featuring separate library tool window, language and stability improvements.

## Built-in Structure Editor

This version adds a built-in Structure Editor.

## Volume Navigator

Volume Navigator, with support for NTFS volumes is capable of parsing NTFS volume structure on disk.

## Version 5.14

This is a bug fix release. Fixed bugs:

## Possible crash on startup on 64-bit systems

Hex Editor Neo could crash if launched on some 64-bit systems. Fixed.

## Invalid editor window scrolling with mouse wheel

Editor window was scrolled incorrectly using high-resolution mouse wheel. Fixed.

## Structure Viewer incorrectly compiled and bound some structures

Structure Viewer could incorrectly compile and bind some structure definitions, namely, the ones that contained huge arrays of static types. It could start allocating more and more memory and binding could take too long. Fixed.

Incorrect file opened after double clicking on error in Structure Errors window
Fixed.
Incorrect handling of multi-byte encodings
The editor incorrectly rendered Text Pane when multi-byte encoding was selected. This bug has been fixed.

## Several reported bugs fixed

Several reported bugs, including application crashes, have been fixed.
Updated functionality:

## Selection is now retained when number of columns change

Multiple selection is not dropped when you change the number of columns in the editor window.
Installer now keeps user settings on upgrade
Hex Editor Neo's installer now tries to keep user settings during upgrade from previous version.

## Version 5.13

Due to significantly better compression of included images (mostly in documentation), this version greatly reduces the installation size and installed disk footprint. In addition, this version fixes following important bugs:

- Several bugs in Structure Viewer fixed
- Structure Viewer now respects the "Byte order change affects floating-point types" setting
- Floating-point numbers in scientific format are now displayed correctly.
- Error saving changes to logical and physical disks Hex Editor Neo hung during saving of changes to logical and physical disks if the application was launched non-elevated on Windows Vista or later operating systems. Fixed.
- Rare crash on exit fixed Application could crash on exit. Fixed.


## Version 5.12

This release introduces significant improvements to the Structure Viewer. See the What's New in Structure Viewer (version 5.12) topic for more information
In addition, the following bugs have been fixed:

## Crash in disassembler

Hex Editor Neo could crash when you invoked Disassembler in some situations. Fixed.
Updated and new functionality:

## Language Pack storage location changed

This release changes the default language pack's storage location, allows user to change it and better supports language packs in portable mode: in addition to userconfigurable storage location, language packs are also searched in <program launch folder>\Localization folder.

## Some dialogs re-designed

Some dialogs were re-designed to better fit German localization.

## Version 5.11

Fixed bugs:
[noindex] and [noautohide] attributes ignored
Structure Viewer ignored [noindex] and [noautohide] attributes. This resulted in incorrect behavior and increased memory usage. Fixed.

## Proxy tab was missing from Settings

One of the recent updates incorrectly removed the Proxy settings tab. Fixed.

## Crash after pressing Copy in Keyboard Map dialog

Program crashed after you pressed the Copy button in Keyboard Map dialog. Fixed.

## Program crashed after saving downloaded update in non-default location

Program crashed when you tried to save to or execute the downloaded update from non-default location.

## Structure Viewer's scheme loading bug

Hex Editor Neo could hang if you loaded Structure Viewer's scheme with multiple bound structures. Fixed.
New and updated features:

## "All opened documents" mode for Find in Files feature

Find in Files/Replace in Files function now provides "All opened documents" pseudo-location, allowing you to perform an operation on all currently opened documents.

## Automatic detection of Language Pack availability

Hex Editor Neo now automatically detects if on-line language pack for user's default language is available and offers to download it

## Improved Structure Viewer performance

Structure Viewer's compilation and binding time reduced by about 10\%.

## Version 5.10

This release adds support for localization. Language packs are created as part of open source project and will be available through built-in updater once they are finished The first available language pack is Russian.

Besides, this release fixes a number of reported bugs. Please consult the View Changes page for detailed information.

## Version 5.01

## Major update to Structure Viewer

This version has completely new Structure Viewer, with a lot of new functionality and improvements. See the What's New in Structure Viewer (version 5.01 ) topic for more information.

## Parallel operation

Several operations, like Find All and Replace All, now utilize several cores or processors (if they are available). It does not give linear speed-up, but sometimes up to $30 \%$ performance gain is obtained. Performance gain is more noticeable in Replace All commands, especially in regular expression mode.

## True parallel execution of all lengthy operations

Previous versions allowed you to execute multiple operations in parallel, but user interface was rather limited: for example, it did not allow you to finish the first operation before the second finishes. Now, all started operations are equal and are executed truly in parallel. Application user interface is not blocked. Only windows of the current document are disabled until the operation execution is completed.

## Support for Intel HEX and Motorola S files

Advanced Copy \& Export now allows exporting selection into Intel HEX and Motorola S formats. In addition, new commands, File » Open» Open Hex... and Edit» Insert Hex... allow the user to convert Intel HEX and Motorola S files to binary form and insert them into the Hex Editor Neo

## Additional information is displayed for several lengthy operations

For Find All, Replace All and other operations, the number of occurrences found so far is displayed on the progress dialog

## Completely rewritten Checksum module

Checksum module has undergone complete rewrite, featuring new algorithms, greatly improved performance and parallel execution. That is, when several algorithms are selected at once, Hex Editor Neo utilizes multiple cores or processors (if they are available) to execute them in parallel.
More hash algorithms are offered (the actual list depends on algorithms available through Cryptography API). More different ways to sum all values in a document or selection are offered, including signed and unsigned sums.

User interface for a Checksum module has also been improved.

## Ability to copy/export the current selection

Hex Editor Neo now allows you to get the list of selected ranges into the Clipboard or directly to text file

## Improved experience when opening volumes and disks

Hex Editor Neo is now capable of opening volumes and disks in the same running instance, even if it runs under non-privileged user account.

## Goto Navigation panel

Each editor window has its own navigation panel with individual location history.

## Improved setup speed

Speed of Hex Editor Neo setup application has been greatly improved. Installer now needs to update less files and system settings and operates much quicker.

## Improved application start-up time

We have improved the application start-up time by reducing time required by various components to initialize and allowing several components to initialize in parallel.

## Improved Find in Files/Replace in Files commands.

New version provides better utilization of multiple cores as well as proper operation on single-core computers; smoother operation on SSDs.

## Improved support for eastern languages

Hex Editor Neo now provides better visual output of eastern languages characters in text pane.

## Version 4.97

## Support for Using Regular Expressions in Replace Pattern

Replace and Replace All commands now support regular expressions as replace patterns, allowing you to refer to parts of matched expression when performing a replace.

## Support for Encodings in all Commands that Work with Regular Expressions

Find, Find All, Replace, Replace All, Find in Files, Replace in Files, Pattern Statistics - all these commands now fully support currently selected encoding when they work in regular expressions mode.

## New scripting methods

The following new methods are available in IFileDocument:

- IFileDocument.FindAllRegExp2
- IFileDocument.FindRegExp2
- IFileDocument.ReplaceAllRegExp2
-IFileDocument.ReplaceAllRegExpWithRegExp
- IFileDocument.ReplaceAllRegExp2S
- IFileDocument.PatternStatisticsRegExp2


## Version 4.96

## Integration with Windows Error Reporting

Hex Editor Neo now fully integrates with Windows Error Reporting. We will be able to receive each submitted crash or hang report, analyze it and provide a solution to the customer.

## New Data Operations - Reverse Bits and Byte Swap

New Reverse Operations commands were implemented in this release. Use the first command to reverse the order of bits in each value in a selection and use the second command to reverse the order of bytes.

## New Arithmetic Operations - Set Minimum and Set Maximum

Two new arithmetic operations allow you to limit each value in a selection.

## Regular Expression Scheme Association

Structure Viewer now uses regular expressions to match a scheme with a the document

## Improvements in the editor

Various small bug fixes and minor improvements in the editor:

- Application now remembers the state of the Details Selection tab between launches.
- Keyboard focus is now moved into the opened document.
- Shift+Home and Ctrl+Shift+Home keyboard combinations now correctly update selection.
- Insert Mode can now be switched during editing.


## Several Bug Fixes

Several reported bugs were fixed in Structure Viewer (see What's New in Structure Viewer (version 4.96) for more information) and Disassembler. In addition, reported errors in exporting selected data as Delphi array were fixed.

## Version 4.95

## UTF-8 Support

This release finally adds the UTF-8 to a list of supported encodings. Now it is not only supported in theAdvanced Copy \& Export module, but also in editor windows, Structure Viewer and everywhere else.
Hex Editor Neo supports single-byte, two-byte, three-byte and four-byte UTF-8 characters. Please note, however, that Hex Editor Neo has limited support for UTF-16 surrogate pairs. See more information on UTF-8 support in the Encodings section.

## Encodings in Structure Viewer

This release covers the last "hole" in encoding support in Hex Editor Neo - now Structure Viewer also uses the code page of the current editor window when it displays single-byte character arrays or strings.

## Pasting Encoded Text

In this release, Hex Editor Neo also uses the current encoding when pasting text into the Text Pane of the current editor window.

## Bug fixes

Several bugs were fixed in the Structure Viewer.

## Version 4.94

This release introduces several bug fixes and a few improvements to various product functions:

- Ability to specify the proxy server to use. Configured proxy server will be automatically used by such features as Check for Updates, Downloader and Crash Dump Uploader.
- An additional column is added to the Open Disk dialog. It displays the physical drive information in a form of $\backslash \backslash . \$ PHYSICALDRIVEN, where $N$ is a disk ordinal number assigned by operating system. This allows you to distinguish between the same disks from the same manufacturer.
- All operations (bitwise, arithmetic and shift) dialogs now include the option "Whole File". You can use it to target an operation to a whole file without the need to select it first.
- Current bookmarks are now displayed in the Go to Offset dialog.
- Added an ability to manipulate the current selection directly from Go to Offset dialog. You can replace, add, remove or invert the current selection while changing the cursor offset in the document


## Portable Installations Improvements

Before this version, when you use the Hex Editor Neo in portable installation, it still uses the target system's registry and file system to store configuration data as well as temporary files.

Version 4.94 of the editor has been modified the following way:

1. On startup, Hex Editor Neo creates a key in the registry and imports the configuration data from portable.config file (located in the product startup folder).
2. On exit, Hex Editor Neo saves the configuration data back to this file and removes the registry key.

In addition, Hex Editor Neo stores vector image cache (used to decrease start-up time) in the user's profile temporary folder. You can switch off this behavior in the Tools » Settings..., General Tab.

## Version 4.93

This release fixes a number of Structure Viewer bugs and further improves structure binding performance.
Here is a list of all changes in this release:

- New tool window layout "Structure Analysis" provides maximum working space for analyzing the document's structure.
- TGA file structure is updated to support run-length encoded images.
- Changed the displayed type name for bit fields. It now shows the number of bits a field occupies (for example: int: 4).
- More accurate structure binding progress update. In addition, a progress window may switch to the marquee mode when binding complex structure with forward and backward pointers.
- A bug that prevented bit fields from being indexed is fixed. Now when you click on the cell which contains a bit field in the editor, the field is properly highlighted in Structure Viewer.
- The execution of the assignment statement (variable = expr;) is greatly optimized. Such complex iterative structures as AVI, TGA and others will benefit from this optimization.
- Structure Viewer now displays a single common progress when binding a structure that executes delayed binds (like PE header, for example).
- Structure Viewer tool window's main operations, like navigation, expanding/collapsing, scrolling and so on were optimized.


## Version 4.92

## General Bug Fixes

- Sometimes when Compare Files command was launched as Windows 7 task from the pinned application icon, it crashed after you pressed the OK button.
- Hex Editor Neo's patch files were incorrectly registered by setup.


## General

- Hex Editor Neo now prevents the computer from going to sleep (unattended) when performing lengthy operation. This of course does not prevent the user to manually sleep the computer
- Now there is an option to move the cursor to the end of the pasted data after the Paste operation (turned on by default).
- Now it is much more easier to select data with mouse. The behavior is more consistent with other Windows applications.
- All Hex Editor Neo commands (such as Edit» Fill..., Edit» Find..., Edit» Replace... etc) now respect the current window's encoding, when single-byte string type is chosen. The current encoding is also displayed in the command's dialog. Note: when using regular expressions, default system encoding is always used.
- Current editor window's encoding is also respected by Pattern Coloring module.
- Mouse wheel scrolling now correctly works with high-definition mice.


## Bug Fixes

- Incorrect scope resolution for complex array indices.
- Empty structures incorrectly had zero alignment, breaking any structure that referenced them.
- Hex Editor Neo could crash when expanding string or wstring values.
- When a script file referenced by a structure definition file was modified, Hex Editor Neo did not refreshed it properly, although notified the user of the change.
- Enumeration visualization could produce incorrect results, if you used signed values in an unsigned enum.
- Structure Viewer tool window could incorrectly display vertical lines.

New features

- New statements: while, for and do...while.
- \$print directive now does not automatically convert its argument to string; instead it uses Structure Viewer's data visualization engine. This allows it, for example, to respect the "Display Hex Values" setting when displaying integer values.
- For pointed types, not only the pointer value, but the snapshot of pointed type contents is visualized on a line (as it is for arrays and structures). In addition, a computed address is also displayed
- JavaScript code now may be specified inline with a new javascript keyword. Note that VBScript code must still be referenced with \#pragma script directive.
- Structure Viewer now uses the current editor window's encoding when displaying non-Unicode strings.
- PE header sample has been greatly enhanced. Now it parses more PE file structures, including import and export directory, resource directory, base relocation table, debug directory and others. It also illustrates how to use Javascript to extend the power of Structure Viewer's declaration syntax.

Checksum

- Click-and-calculate feature implemented: non-computed checksum value is automatically computed when you double-click it (no need to click checkbox and then Refresh)


## File Compare

- File Compare tool window's commands are now application-wide commands. You can put them on the toolbar or assign keyboard shortcuts to them.

Clipboard

- Now Edit » Copy and Edit » Cut commands automatically put selected data in text format into the Clipboard. Copy \& Export tool window is still used to fine-tune the format used.
- "Formatted Values" copy\&export mode now allows exporting data without separators (as text stream).
- "Raw Text" copy\&export mode also lists "Current Encoding" as encoding type. When selected (default), encoding from the current editor window is used.


## Version 4.91

This version features a completely rewritten setup for Hex Editor Neo.

## Transactional Installation

Setup now performs installation of the product as a single transaction (on supported operating systems and configurations, see below). If anything goes wrong during installation, including software or hardware errors, computer configuration is not modified. All setup activities are part of the transaction, including modifications done to the file system and system registry.

Setup always tries to use transactions on supported operating system. If installation fails due to the transaction error, it retries the operation without using transactions. It also does not use transactions on legacy operating systems.

## Restart Manager Support

Now, when setup needs to replace or remove a locked file, it uses the system restart manager (Windows Vista or later) to find the application that has a file open. It also offers a way to gracefully restart the application in order to update the file. This new feature greatly reduces the need for system restart during setup.

On legacy operating systems setup emulates the restart manager's behavior, still trying to minimize the need to restart a computer.

## Per-User Installation

In addition to the previous, "per-machine" installation mode, a new installation mode is offered. It allows the user to install the application only for himself. In this mode, setup does not modify any per-machine settings. In addition, a user does not have to have administrative rights to install the Hex Editor Neo. After the product is installed in peruser mode, it is not visible to other users of the computer, but may also be installed by them in per-user mode.

Limitations
Take the following limitations into account when installing Hex Editor Neo in per-user mode:
On Windows Vista and later operating systems, standard user (and even computer administrator without elevation) does not have enough privileges to create System Restore points. This does not apply to administrator user on Windows XP computer. Setup always tries to create a system restore point, but will silently ignore this step if it does not have enough privileges.

## Portable Installation

Another new installation mode is used to make a portable copy of Hex Editor Neo. Once installed, the copy of the product may be transferred to other locations, or to other computers where it can be used without installation. In this mode setup allows the user to install the product on the removable media and network shares.

Limitations
Take the following limitations into account when installing Hex Editor Neo in portable mode:

- Setup does not create shortcuts, does not update system registry and does not publish installed application (it will not appear in Add/Remove Programs Control Panel applet (Programs and Features starting with Windows Vista).
- You will need to manually delete product installation folder in order to remove the product from your computer.
- Setup does not register Hex Editor Neo's type libraries, preventing you from using the product in scripting environment.
- Setup does not register Windows shell extension, making "Edit with Hex Editor Neo" and "Edit with Hex Editor Neo (shared)" context menu options unavailable.


## Command-line Parameters

New Hex Editor Neo's setup allows you to specify several parameters on the command line:

| Parameter | Description |
| :--- | :--- |
| -silent | Suppress any user interface dialogs during setup |
| -uninstall or -u | Turn on uninstall mode |
| -portable | Turn on portable mode |
| -prop INSTALLDIR $=[$ path $]$ | Specify the installation folder location. Must end with a backslash. |
| -user | Turn on per-user mode |
| -machine | Turn on per-machine mode |
|  |  |
| Hex Editor Neo in per-user mode into the "c:\Programs $\backslash$ Hex Editor Neo" folder: |  |

## Version 4.85

Below is a list of changes in Hex Editor Neo 4.85:

## Shared Files Support

Hex Editor Neo is now capable of working with files shared with other applications. Using the File » Open » Open Shared File...command you can open a file in the editor, at the same time allowing other applications to make changes to the file.

This feature is supported on all operating systems Hex Editor Neo natively supports. A shared file may be located on a local, removeable or remote media formatted with any supported file system (FAT, FAT32, NTFS, ExtFS...)

After you open a shared file, Hex Editor Neo starts watching for file's modifications. If another application makes changes to a file, Hex Editor Neo displays aShared File Conflict Dialog, which gives you options to ignore a change, update editor's copy, and/or merge changes with local changes.
In addition, for all "update"-kind options you may instruct the editor to compare the new version with a previous one. Two file comparison algorithms provided by File Compare module are supported. Hex Editor Neo also allows you to retain older versions of a file in the editor. You may edit these old versions as any other kind of document in the editor, save your changes and so on.

Hex Editor Neo also detects file deletion. In this case it gives you an option to close a file or continue working with it, retaining all the changes you made to a file.
A configurable option exists which governs how Hex Editor Neo reacts on file renames. If enabled, renaming a file in another application (like Windows Explorer, for example) automatically changes the file's name in Hex Editor Neo's interface (without any prompts).

## Single selection mode option

"Single selection mode" option, which was previously only available in General Settings page (under name "Auto drop selection"), is now also available from the Select menu and on Selection Tool Window's toolbar.

In addition, a number of reported bugs have been fixed in this release.

## Version 4.83

This version fixes a potential crash on application exit and introduces new formatted text output format: Assembler array.
In addition, MBR format (for Structure Viewer) definition is fixed (see mbr.h).

## Version 4.82

Below is a list of changes in Hex Editor Neo 4.82:

## Restore original file attributes

File Attributes tool window now offers a way to restore original file attributes, even after you modify them and apply your changes.

## Auto completion in Explorer

Address bar in Explorer tool window now offers autocompletion.

## NTFS streams support in File Compare

When compared files are identical, they are also checked for one or more alternate streams. A user is warned if two identical files contain NTFS streams.

## 2D graph in Statistics.

Statistics Tool Window now provides you with two graphical data representations, instead of one.

## Extended goto in Disassembler

Goto command in Disassembler Window now allows you to go to a named function or method. Autocompletion is supported.
In addition, the following reported or known bugs were fixed:

## Elevation bugs

When Hex Editor Neo elevates itself (on Windows Vista or later operating systems), elevate.exe process is not terminated and remains running forever. Fixed.

## Invalid progress bar

Invalid progress bar is displayed for some operations. Fixed.

## Crash in operations

Bitwise, arithmetic, shift and case change operations may crash when used in some configurations. Fixed

## Version 4.81

Below is a list of changes in Hex Editor Neo 4.81:

## Full history tree window

You can now view the entire document change history in a special window. History is represented as a tree with a node for each document change. You can quickly switch between states, get more information about each state or execute one of the purge commands. Comfortable zooming and panning features are provided by the window.

## Complete rewrite of command routing and toolbar customization system

This release brings completely new version of the command routing and toolbar customization subsystem. We have greatly optimized Hex Editor Neo's startup and runtime performance and brought greater flexibility to user interface customization. Built-in and user-created layouts now not only affect tool windows, but also change commands on default toolbars and size of their buttons.

## Checksums and Statistics now store per-document information

Checksums module and Statistics module now store per-document information. For example, you calculate statistics for a document, switch to another one and calculate statistics for it as well. Now, if you switch back to the first document, Statistics tool window will immediately display information it previously calculated for the first document.

## Changes in the Free Edition End User License Agreement

Free version's EULA now grants commercial usage of the free edition of the Hex Editor Neo.

## Improved Explorer integration

"Edit with Hex Editor Neo" shortcut menu option is now added to drives as well.

## Improved Windows dialog

Windows dialog (it displays a list of all opened editor windows) now automatically activates a window under the cursor. There is no need to click the Activate button anymore.

In addition, the following reported or known bugs were fixed:

## Several bugs in Structure Viewe

A number of non-critical bugs were found and fixed in Structure Viewer.
32-bit integer types (both signed and unsigned) were incorrectly handled in big-endian mode. Note that this did not affect 32 -bit signed and unsigned long types. This bug is fixed as well.

## Decrypt function is not working

On some computers it was unable to run a Edit » Decrypt... command. This has been fixed
A lot of small glitches that were in our "suggestion queue" have finally found their time to be fixed in this release.

## Version 4.76

This release focuses on major performance improvement in one of the core Hex Editor Neo components, responsible for working with large files. (Hex Editor Neo considers files bigger than 64 MB on 32 -bit systems and bigger than 128 MB on 64 -bit systems as large files).

Careful optimizations and wiser use of system resources allowed us to greatly optimize this core component. As performance of most editor functions depends on it, we have managed to improve the speed of various functions from $2 x$ to $8 x$ !

The most boost is displayed in "cheap" operations, like General Statistics and Pattern Statistics, moderate boost in heavier functions as Replace All and less boost in "heavy" functions like Find All, for example.

In addition, document saving performance has also been greatly improved. On some of our test platforms, file copying (File » Open command followed by File » Save As... command) works faster than in Windows Explorer!

In addition, several reported bugs have been fixed.

## Version 4.75

New and updated features in this release:

## Revised Operation History

User interface for document operation history has been completely revised. Now it is much more comfortable to use one of the most powerful and unique Hex Editor Neo's features - history branches.

## Application load-time improvement

A lot of attention in this release has been paid to reduce the time Hex Editor Neo needs to launch.

## Offset display type

An offset (or address) may now be displayed as hexadecimal, decimal or octal numbers.

## New view types

In addition to hexadecimal, decimal, float and double, data in editor windows may now be displayed as octal and binary values. All Hex Editor Neo's commands and modules are also compatible with new view types, including Copy \& Export module.

## New range rule type in Pattern Coloring

Pattern Coloring introduces a special rule type: range rule. For range rule, you define a range which is then hilighted in a document.

## Advanced support for Editor Window settings

You can now configure the default editor window settings on special settings page (Tools » Settings»Editor). These settings include offset display type, data display type and grouping, byte ordering, a number of columns, character encoding and visibility of code pane and text pane. In addition, there is a special option to associate a different set of window settings for each opened document and restore them next time the document is opened.

In addition, several known and reported bugs have been fixed.

## Version 4.73

New and updated features in this release:

## Windows 7 Beta support

This release introduces support for new cool features of the Windows 7 Beta taskbar. Progress indication and jump lists are fully supported and utilized by the Hex Editor Neo, and in addition, user is in control of customizing the tasks presented on the taskbar.

## Structure Viewer

New structure formats: RAR, ICC, VHD, ZIP, TIFF, WAV, TGA, PSD, PIC, PCX, PAL, EMF, EPS, CAB, MBR, CDA.

## New column: Size (to see the size of the field).

See also below for a list of fixed bugs
File Compare
Recompare Files command added.
Color picker dialogs
A color may now be selected with a double-click.

Bugs fixed in this release:

## Structure Viewer

The following bugs are fixed:

- Nested pointers are not supported.
- Array size is parsed incorrectly if multiplication is used in size expression.
- Invalid script file causes a crash.
- Several parser bugs fixed.


## Find in Files

Deadlocks and possible crashes were fixed. In addition, performance for the feature has been increased.

## Explorer

Several bugs are fixed in Explorer and it is now compatible with Windows 7 Beta.

## Version 4.72

New and updated features in this release:

## Disassembler

Version 4.72 brings support for x 64 executable files.
In addition, a bug that sometimes lead to program crash has been fixed.

## Version 4.71

New and updated features in this release:

## Disassembler

Version 4.71 brings support for .NET assemblies. It also features an updated disassembler view window, with support for fast navigation, selection of assembly text, column configuration and much more. It also allows you to display an assembly listing in AT\&T format.
Disassembler has an improved quality of the assembly listing. Module imports are now also analyzed and displayed, in addition to symbols.

## Structure Viewer

The most of the new functionality is within the Structure Viewer module again. It now understands new built-in types, statements and directives. It also supports variables, constant and variable arrays and expression evaluation with dynamic type inferring.

This release eliminates most of previously imposed limits on structure size, array size etc. The Structure Viewer Tool Window has been updated to easily handle billions of tems with very fast and clean navigation. Commands like Expand All and Collapse All will take less than a second expanding/collapsing a structure with a billion of items. If you bind a complex structure to a huge file (for example, AVI), you can estimate the editor to easily carry the job without running out of memory.
A number of handy features, such as Synchronize Position (which works "both sides" - from structure to document and from document to structure), Mouse over field tracking (which highlights a field under the mouse cursor) greatly simplify the file structure analysis.

Structure definitions may extend their functionality by referring to functions written in JavaScript or VBScript. These functions have access to the underlying parser and document, thanks to the IFileDocument and IParser interfaces, exposed to them by the Hex Editor Neo.
The overall compilation and binding performance has been greatly improved.
There is a special topic in this documentation that describes the changes in the Structure Viewer module.

## Copy\&Export

The Advanced Copy \& Export module now follows the model offered by the editor window. It now allows you to execute the Copy and Export commands with no selection present. In this case, a cell under the cursor is copied/exported.
The current byte order is now taken into account when performing copy or export functions.
It also offers an option to configure the number of digits to be exported for floating-point and double precision floating-point types.

## Refresh View Command

When working with logical or physical disks, process or physical memory, this command allows you to refresh the editor window to display any possible changes to the underlying data, performed by some other application or operating system.

## /s Command-Line Parameter

When launched with "/s" command-line parameter, Hex Editor Neo forwards the list of files to open to an already running instance, if there is one.

## "Display cursor address on the left" Option

Several customers complained about the address of the cursor location being displayed at the start of the line. Now this functionality can be switched off. See the General Settings topic for more information.

## Reverse Sorting on Statistics Columns

Statistics window now allows you to sort columns in both ascending and descending order in its table mode.

## New IFileDocument Methods to Be Called from Scripts

A previously introduced ISequence object was called to make it possible to call methods of the IFileDocument interface from scripting languages (such as JavaScript and VBScript). Although, our tests has shown, that in some cases it is still impossible for a script to call one or another method due to the limitations in the Automation model and the way scripting engines are implemented.
This release finally resolves the problem by providing a special variant of each affected method. Every such variant may be easily called from the scripting languages.

## Go to in Process

Go to in Process command now interactively updates the cursor position when you click on the image sections or modules (without closing the Go to window).

## UI Sugar

You can now make any tool window appear as a tabbed document. That means that the tool window will not be dockable to one of the four sides of the mindow anymore, nor it will float over the main window, but instead will be placed where all editor windows reside. You may then take advantage of the Editor Window Placement feature to position a tool window.
Each tool window now also memorizes its floating position separately of its parent frame.
Ctrl+Mouse Wheel now quickly changes the font size in an editor window. This is a quick way to change a font size without using the Layouts dialog.
The following is a fixed bug list:

## Purge All History and Load History Commands

These commands did not work properly in Free Edition mode.
Copy\&Paste Bugs

Several bugs with copying and pasting document data have been found and fixed. This may include several application crashes.

## A Lot of Minor Bugs

Quite a few of minor bugs or inconsistencies were found and fixed throughout the product.

## Version 4.64

New and updated features in this release:

## Structure Viewer

Structure Viewer has been completely redesigned and rewritten. It now supports a sophisticated structure definition language with expression evaluation, pointers and bit fields, case unions and assertions, constants and enumerations and much much more.
In addition, C99-compliant preprocessor is run before a source file is compiled, a preprocessor with full support for macros and conditional compilation.
Source files are optimized and saved into precompiled files, which are then quickly loaded by Hex Editor Neo on demand. Structure binding performance has also been greatly increased.
In addition to the ability to bind a structure to an absolute or relative address or to a cursor position, a structure may now be bound to an expression. This expression may reference constants and enumerations declared on the global source file scope or even reference previously bound structures.

## Scripting

New Sequence Object provides scripting languages (such as JavaScript and VBScript) a way to build a pattern and pass it to File Document Object methods that require a pattern.

## Version 4.51

A number of reported bugs are fixed in this release. Hex Editor Neo version 4.51 introduces a major performance boost.
Thanks to a new internal data structure and handling algorithms, a major performance boost is found in the following functions:

- Fill, Delete, Cut and Paste - when they operate over a multiple selection - have up to $3 x$ performance boost.
- Find All, File Compare, Selection Loading - up to $10 \%$ performance boost.
- Replace All - from $1.5 x$ to $2 x$ performance boost.

Similar numbers refer to Find in Files and Replace in Files functions.
New features in this release:

## Disassembler

Hex Editor Neo now includes a disassembler module. The current implementation is capable of disassembling 32-bit x86 code, including support for FPO, MMX, SSE, SSE2 and 3DNow! instruction sets. Disassembler automatically searches for symbol files and uses the information from them to increase the assembly output readability. The Disassembler View may be connected to ordinary editor window which provides you with another handy data viewing and manipulation tool. The disassembler module works with files and processes' memory and is even automatically updated when you modify the document.

## Regular Expressions

Find, Find All, Replace, Replace All, Find in Files, Replace in Files, Pattern Coloring and Pattern Statistics commands and modules now fully support regular expressions, ECMAScript-compatible implementation of regular expressions provides you with extreme power ever found in binary editors. Very efficient implementation of "bulk" commands (Find All, Replace All, Find in Files and Replace in Files) combined with unique features of Hex Editor Neo's multiple selection feature, together form an invaluable editing tool. A concept of named regular expression classes even allows you to define recursive grammars.

## Window Switching

Hex Editor Neo now provides fast and convenient tool that can be used to instantly activate any visible tool window or editor window. When you press Ctrl + Tab or Ctrl + Shift + Tab (user configurable), a navigation window is displayed. It provides the list of visible tool windows and all opened editor windows for you to activate. See the Window Switching section for more information.

## Code Pane

Hex Editor Neo now provides you with a method to completely hide a code pane, leaving only the text pane visible. Combined with the auto columns mode, it is a powerful tool to edit text or binary-text data.

## Default Layouts (Three Different Ones)

When you run the Hex Editor Neo for the first time, you are provided with a choice of default working layout. You can choose between a "Novice User Layout", "Advanced User Layout" and "Expert User Layout". You can later change the chosen layout or make your own. See the Default Layout section for more information.

## Version 4.41

Version 4.41 fixes a number of reported bugs and introduces some performance optimizations. New features in this release:

## Case Change Operations

Hex Editor Neo now offers three new Data Operations: Lower Case, Upper Case and Invert Case. Case change operations take into account current encoding and are capable to work with Unicode data.

## Application-Wide Byte Order Support

Hex Editor Neo now fully supports both little-endian and big-endian byte orders. Different byte order can be set separately for each editor window and there is an option that controls whether floating-point types are affected by byte order change. Byte order change commands have been removed from Data Inspector and Structure Viewer modules, as they now respect the byte order configured for a window.

## Modify Bits Command

New Modify Bits command allows you to see a visual binary representation of the number and modify individual bits.

## Copy and Cut Work Without Selection

Copy and Cut commands can now be used when there is no selection in the current editor window. In this case, only the cell under the cursor is copied/cut into the Clipboard.

## Version 4.33

Hex Editor Neo version 4.33 fixes a number of reported bugs. Affected modules: Data Inspector, Structure Viewer, NTFS Streams, File Compare.
It also adds the "Hide All Tool Windows" command on the toolbar so you can quickly maximize the workspace and then return to the previous configuration at one mouse click.

## Version 4.32

Hex Editor Neo version 4.32 fixes a number of reported bugs and introduces a lot of new cool features. New features in this release:

## New Document Types

Hex Editor Neo is now capable of opening volumes, physical disks, process virtual memory and computer physical memory (on supported operating systems)

## New Editor Commands

Encrypt and Decrypt commands may now be used to encrypt and decrypt a document (or part of it), using one of the installed block or stream ciphers.
A large set of new data operations: bitwise (NOT, OR, AND, XOR), arithmetic (Negation, Addition, Subtraction, Multiplication, Division and Remainder) and shift (Logical Left and Right, Arithmetic Left and Right and Rotation), are provided. They operate in linear time (depending on selection's complexity), that is, for single-range selection they execute immediately, regardless of data size.

## File Comparison

Hex Editor Neo provides two methods, simple comparison algorithm and difference algorithm to locate changes between two files.

## Little-endian/Big-endian in Structure Viewer

Structure Viewer may now be switched between little-endian and big-endian data representation.

## Version 4.21

Hex Editor Neo version 4.21 fixes a number of reported bugs, provides a number of optimizations as well as introduces new cool features. New features in this release:

## Performance Optimizations

This release has undergone several performance optimizations resulted in significant speed-ups in the following commands and features:

- A special image cache speeds up application loading time. This special cache stores pre-rendered vector images (which are used throughout the Hex Editor Neo) to significantly reduce the application startup time, which was very small even before this cache was implemented!
- A number of algorithm improvements allowed several editing functions to operate much faster and consume much less RAM. This include typing, inserting and deleting (with complex multiple selection).
- Several improvements were made to the Find in Files feature, leading to much less memory consumption by this function.
- File saving algorithms undergone several optimizations leading to smoother and faster saving.


## NTFS Streams Support

Hex Editor Neo now fully supports NTFSalternate data streams. A special NTFS Streams Tool Window displays all streams for the currently active file, allows you to delete or rename them, open them for editing or create new streams.
In addition, the powerful Find Stream feature allows you to locate all streams on your computer.
File Attributes has also been updated to display more information related to streams: the total number of streams, size of all file's streams and the total size occupied by a file (including all its alternate streams)
Find in Files now has an option "Include streams" that you may use to include alternate data streams in Search or Search and Replace operations.

## Version 4.11

Hex Editor Neo version 4.11 fixes a number of reported bugs as well as introduces new cool features. New features in this release:

## Built-In Explorer

Hex Editor Neo now has two Windows Explorer-like windows as part of its user interface. They can be used to perform usual file and folder operations and also tightly coupled with Hex Editor Neo. For example, you can open files in the editor by double-clicking them in the Explorer window simply dragging them to the editor's window.

## Encodings

Hex Editor Neo is now capable of displaying text in the text pane according to a selected encoding. More than 130 different encodings are supported (subject to installed code page files and fonts). When the cursor is in the text pane, typed characters are also converted using the selected encoding. Encoding can be set separately for each editor window. Different ANSI, UNICODE, ISO, OEM, MAC, EBCDIC and country-specific encodings are supported.

## Checksum Calculation

Hex Editor Neo supports 16 checksum calculation algorithms, including MD5, SHA-1, CRC-16, CRC-32 and others. Custom CRC algorithm is also provided. You can compute the checksum for the whole file or for the current (multiple) selection. Several checksums may be computed at the same time. Results are easily exportable and copyable to the Clipboard.

## Advanced Copy \& Export

Selected document data may now be converted into a number of supported text formats and placed into the Clipboard or exported to a file. Space-separated or commaseparated numbers, raw text with selected encoding, programming language array declaration, Base64, UUencode and Quoted-Printable are among supported formats.

## Pasting Text

Text may now be pasted into the editor. Depending on the active pane, the text is either pasted "as is" (if the text pane is active), or parsed as a sequence of hexadecimal or decimal integers (if the code pane is active).

## Version 4.01

Version 4.01 is an absolutely new release, designed and developed from scratch. It does not contain any code from the previous product version
On the other hand, Hex Editor Neo shares some code base with our Serial Monitor and USB Monitor products, which are part of the Hex Editor Neo product.
Hex Editor Neo retains all interesting and useful features of its predecessor, such as multiple selection, multiple editor windows, vector icons and bookmarks, but implements all these features in a new way, significantly lowering memory requirements and increasing performance and stability.

The performance of all functions has been significantly increased. Several functions that previously executed in several steps now execute in one step (this includes Replace All command, for example).

Hex Editor Neo does not longer support Windows 98 and Windows ME operating systems. But the list of supported operating systems has actually increased:

- Windows Vista (x86)
- Windows Vista (x64)
- Windows XP (x86)
- Windows XP (x64)
- Windows Server 2003 (x86)
- Windows Server 2003 (x64)
- Windows 2000

Pentium III or newer processor is now required by the Hex Editor Neo
A lot of new features have been implemented in this release. The full list is provided in the Hex Editor Neo's Features section.

## Definitive Guide

## User Interface Elements

## Commands

All functions of the Hex Editor Neo are invoked through commands. For convenience, almost every command is accessible using several user interface tools:

- Each command is accessible through the main menu. Several commands used to execute local actions of specific tool windows are accessible through context, or shortcut menu. Shortcut menu is invoked by clicking the right mouse button, or by pressing "Context Menu" keyboard button (usually located next to the right ctrl key).
- Most commands are also displayed on toolbars. Two kinds of toolbars are used in the Hex Editor Neo. Main toolbar is located right under the menu bar and is fully customizable. You can also create new toolbars and place them near the menu bar. Other toolbars are located in tool windows and contain the most used commands that are specific for those tool windows. Starting from version 7.01 , tool window toolbars also support customization.
- Several most used commands have associated keyboard shortcuts or hot keys. Pressing such associated key or key combination invokes the command. You can change assigned key combinations as well as assign new key combinations to other commands.

All commands described in this documentation are mentioned by their location in the main menu. For example, the Save command, located under the File main menu item is referenced as File» Save. Other examples are File» Close for a Close command under File menu and Tools » Settings... for a Settings... command under Tools menu

By convention, if ellipsis symbol (...) terminates menu item's name, it means that user is required to provide additional information to execute this command. This may include displaying a dialog box, asking for confirmation etc. For example, the File» New command does not require any additional action from you, while the File » Open » Open File... command opens a File Open dialog, therefore requiring you to provide additional information.

## Main Menu

Main menu bar is located at the top of application window, right under the title bar.

```
FFile Edit View Select Operations Bookmarks Annotations Macros Tools History Window Help
```

Main menu contains almost all commands implemented by the Hex Editor Neo. It is structurally divided into the following groups:
File
Contains commands related to creating, opening, saving and closing of documents and scripts.
Edit
Contains all editing commands, Clipboard commands, Edit » Undo and Edit »Redo commands.
View
Contains the list of tool windows, commands to export/import tool window configuration or load a predefined configuration. It also contains commands that can be used to switch the layout of editor window.

## Select

Contains commands that manage a current selection

## Operations

Contains commands that perform bitwise, arithmetic and shift operations on an opened document.

## Bookmarks

Contains commands that control the Bookmarks Tool Window.

## Annotations

Contains commands that allow the user to provide text comments for the specific document positions and ranges and also to control the Data Annotations Tool Window.

## Macros

Contains commands related to support of macros in the editor.
Tools
Contains commands that control different tool windows, including Pattern Coloring, Structure Viewer, Attributes, Statistics, NTFS Streams, Checksum and Data Inspector. The Tools» Settings... command is an entrance to the central place where you fine tune the Hex Editor Neo by changing different options.

## History

Contains commands related to operation history management
Debug
Contains commands related to creating, managing, running and debugging scripts.

## Window

Contains commands that manage the number and location of editor windows.

## Help

Contains commands that can be used to access this documentation file, check for program updates, display keyboard map (a table of associated shortcuts) as well as display information about the Hex Editor Neo.

Several main menu options are hidden when there are no documents open in the editor. The Scripting menu option is also hidden when no scripts are open in the editor.

## Toolbars

Toolbars present a subset of Hex Editor Neo commands to the user by means of displaying commands' images. Main toolbar and user created toolbars are located under the menu bar:



To execute a command located on the toolbar left-click its image. If you hold a mouse pointer over an image for a while, command's name is displayed in a tooltip window, along with assigned keyboard shortcuts. More detailed command description is displayed on the status bar at the same time

Main toolbar, custom toolbars as well as tool window toolbars are fully customizable. Advanced toolbar customization is described in the Toolbar Customization topic.

Hex Editor Neo associates a number of its commands with keyboard shortcuts. You can always see the list of all currently assigned shortcuts using the Help " Keyboard Map command. The list of assigned shortcuts may also be printed for reference or copied to the Clipboard to be used in another application.

Most commands that are present in other applications, like File » Open » Open File.., File » New, File » Save, Edit » Find... and others are assigned standard Windows shortcuts ( Ctrl+0, Ctrl+N, Ctrl+s and Ctrl+F for listed commands) by default. For those commands for which several standard shortcuts exist (like the Edit» Replace... command, for which both $\operatorname{Ctrl}+\mathrm{H}$ and Ctrl +R are widely used), several shortcuts are assigned by default.

We also refer you to Keyboard Customization section where it is illustrated how you can change default keyboard shortcuts or create your own.

## Tool Windows

Tool windows are a special-purpose windows that are always visible to the user (unless they are hidden) and always ready to perform a task. Hex Editor Neo has the following tool windows:

- Base Converter
- Bookmarks
- Checksum
- Copy \& Export
- Data Annotations
- Data Inspector
- Debug Call Stack
- Debug Watch
- Downloader
- Explorer
- File Attributes
- File Comparison
- Find in Files
- Operation History
- Macros
- NTFS Streams
- Output
- Pattern Coloring
- Selection
- Statistics
- Structure Library
- Structure Viewer
- Volume Navigato


## Location

A tool window may either be floating or docked to the application frame. A docked tool window may also be auto-hidden. All these terms are described later in this section.

## Floating Tool Window

## Pattern Coloring

## (5) 4 ( ) $x$ त

- 1 "for" (ASCII)
- 2 \{0x0d,0xOa\} (BYTE)

Description Sample Text

Floating tool window is convenient when you want to maximize the working space used by editor windows. It may be positioned outside of the main application window, and may even be located on another monitor if you have one. This is contrary to the docked tool window, which may only be located within main application window.
The position of the floating tool window may be changed by dragging it by its title with a mouse. You can also use mouse to change the size of the floating tool window by dragging either its frame or one of four corners.

The important thing to have in mind while working with floating tool windows is the keyboard focus. Operating system directs all keyboard input to the window owning keyboard focus. Usually an active title is used to indicate the window that has keyboard focus. Blinking caret may also be used to indicate the focused window. Activating floating tool window by a mouse click always brings a keyboard focus to this window, allowing you to provide input to the window. Clicking outside of the floating tool window usually takes a keyboard focus away from it. To continue sending keyboard input to the tool window, you may need to activate it again.

## Docked Tool Window

A tool window may also be docked. When docked, it is attached to a tool window frame. A frame may be attached next to another frame or directly to the top, left, right or bottom side of application window. Frames may be located next to each other, inserted one into another and merged. Each frame contains at least one docked tool window and may also contain several. Below is a screenshot of a frame with a single tool window docked to the right side of application window:


Here's another example, where frame contains several docked tool windows. Several docked windows are tabbed within a single docked frame:

| Checksum |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |
| $\square$ Algorithm | Value (Hex) | Value (Dec) | Parameters |  |
| Simple summators |  |  |  |  |
| 8-bit sum | N/A | N/A |  |  |
| - 16 -bit sum | N/A | N/A |  |  |
| -32-bit sum | N/A | N/A |  |  |
| Zero bits | N/A | N/A |  |  |
| $\triangle$ One bits | N/A | N/A |  |  |
| More summators |  |  |  | ^ |
| 64-bit sum | N/A | N/A |  |  |
| Sum of Little-Endian 16-bit values | N/A | N/A |  |  |
| Sum of Big-Endian 16-bit values | N/A | N/A |  |  |

You can dock any tool window by dragging it by its title bar or tab. During dragging, special placeholders appear on the screen. When you move the mouse pointer over the placeholder, semi-transparent rectangle appears on the screen, indicating an approximate position of the docked window. Releasing the mouse button docks a window in the indicated position.

Side placeholders always appear, allowing you to dock a tool window into a new frame and attach it to the top, left, right or bottom side of the application window.


When you drag a window over a docked frame or over the editor windows area, another placeholder appears. It is composed of five boxes: top, left, middle, right and bottom. Dragging a window over the side box docks the window into a new frame, which is attached to the top, left, right or bottom of the frame currently under mouse pointer.


If you drop a window to the middle box, you will dock it into the frame currently under mouse pointer. Dropping a window to the docked window's title also docks it to target window's frame.

You can use mouse to change position of docked windows within a docked frame, as well as drag the window "out of the frame" to make it floating or to dock it into another frame or new frame.

Double-clicking on the docked frame's title makes a frame floating. Double-clicking on the docked window's tab makes only that window floating. Double-clicking on the floating frame's title docks it back to the same location it previously occupied.

All docked frames are separated with each other by splitters. When you position a mouse pointer over such splitter, it changes its shape, telling you that you can drag the splitter to change size of adjoining frames. Splitter position is proportional to the application window size; frames are resized automatically and proportionally when you resize application window.

Any docked tool window frame may be auto-hidden. To auto hide a frame, click the Auto Hide button:
Volume Navigator


Four auto-hide bars are located on four application window sides. They are hidden when empty and appear as soon as you add at least one frame to it. When you auto-hide a docked frame, it chooses one of four auto-hide bars, depending on the docked position. That is, if the frame was docked closer to the right side of application window, right auto-hide bar is chosen and so on


Each docked window's tab in the auto-hidden frame is displayed on the auto-hide bar. Two auto-hidden frames are separated from each other with a slightly larger gap. Clicking the mouse on the tab opens corresponding tool window. The tool window smoothly drives out of the auto-hide bar. Click in the tool window to switch keyboard focus to it if you need to provide any keyboard input to the window. The window is automatically closed (quickly drives back into the auto-hide bar) when keyboard focus is lost by the window and mouse cursor is not over it.

Auto hiding tool windows allow you to save the screen space, while still having quick access to functionality provided by the tool windows.
Pressing Auto Hide button again "unhides" the docked frame and returns it to its original docked position

Tool Windows as Tabbed Documents

A tool window may be dragged and dropped into the document frame or placed next to an opened document. Alternatively, right-click on the tool window caption and select the Tabbed Document context menu option. Right-click again and select Dockable Tool Window to revert back to a docked state.

## Tool Window Visibility

Any tool window is always in one of two states: visible and hidden. Visible means the tool window is floating or docked to any of the docked frames or auto-hidden and appears on one of four auto-hide bars. Hidden tool window does not appear anywhere on the screen. Note that a tool window, which is not directly visible by the user, but which tab is visible in one of the docked frame, is considered visible by this definition.

Pressing the Close button hides the tool window or entire frame, depending on the "Close affects the current tool window only" setting on the General settings page. To make a previously hidden window visible, select its name in the View menu. This restores the previous position, docked and auto-hidden state of the window and activates it. It also activates the tool window even if it is in visible state, therefore, allows you to quickly move keyboard focus to the tool window, without using mouse.

## Tool Window Configuration

The current location of all tool windows, their state and visibility, comprise the tool window configuration. Hex Editor Neo allows you to export and import this configuration to the file for sharing, exchanging or archiving purposes. There are also a number of predefined tool window configurations shipped with a product. You can quickly switch between them using the View Predefined Layouts menu group.

First three options (Novice User Layout, Advanced User Layout and Expert User Layout) correspond to three options described in theDefault Layout section. Selecting one of those options not only switches to a given layout, but also sets it as a default one

## Window Switching

Hex Editor Neo interface presents you with a large number of tool windows, as well as an unlimited number of editor windows. Application has several window switching mechanisms, with the "Navigation Window" being the default one.


Navigation window is opened when you press the Ctrl + Tab or Ctrl + Shift + Tab key combination (see also Keyboard Customization). It lists all visible (possibly autohidden) tool windows as well as all opened editor windows. Using the arrow keys, Tab, Shift + Tab or mouse, you may select the window you want to activate. As soon as you release a ctrl key, the navigation window is closed and selected window becomes active.

Navigation window provides a quick window activation mechanism

## Other Window-Switching methods

The General Settings page contains a "Window switching style" setting, using which you may set how Hex Editor Neo switches windows.
Legacy
Window switching operates exactly the same as in previous (prior to 4.51) versions of Hex Editor Neo.

## Last recently used

Navigation window is not used. Ctrl + Tab and Ctrl + Shift + Tab work on the list of editor windows using the last recently used algorithm. You cannot activate tool windows using this method

## Using navigation window

The navigation window is displayed and allows you to select any visible (or auto-hidden) tool window as well as any opened editor window.

## Default Layout

The first time you start the Hex Editor Neo, you are asked to choose a default tool window layout


You are offered a choice of three options: Novice user, Advanced user and Expert user.
After you click an option, Hex Editor Neo's user interface immediately switches to the selected layout. Once you have chosen your layout, press the Continue button.

## Choosing Another Default Layout

Although the "Choose Default User Interface Layout" window is presented only once, you can easily change a layout at a later time. To do this, go to the View » Predefined Layouts menu, and select a layout from the list. The "Choose Default Layout" window may also be invoked using the Quick UI Tweak... button on the toolbar.

## Documents

## Supported Document Types

Hex Editor Neo supports several documents to be viewed and edited in its environment. All supported document types are listed below.

## Files and Streams

The first and most commonly used document type is a file. Hex Editor Neo allows you to edit any file for which you have Modify access and allows you to view the contents of any file for which you have a Read access. In addition, full support for NTFS alternate data streams is provided

Starting from version 6.12, Hex Editor Neo provides a way to open files for which you normally receive "Sharing Violation" error

## HEX Files

Hex Editor Neo has native support for Intel Hex and Motorola S-Records encoded binary files. It automatically decodes these files when the user opens them in the editor using the File » Open » Open File... or File » Open » Open Hex... commands.

When the user saves changes made to the HEX files, the editor automatically saves the file using the original format.

Hex Editor Neo normally locks opened files, granting only Read access to other processes. The lock is held until a file is closed, preventing other applications from making modifications to a file. In addition, if the file is already opened by another application, Hex Editor Neo may fail to open a file if it fails to install a lock.

While this scheme allows an editor to work with files of virtually any size in a very effective way, it sometimes may be convenient to work with a file in a shared mode
Starting from version 4.85 Hex Editor Neo supports opening files in shared mode.

## Volume

Hex Editor Neo allows you to view or edit contents of any mounted volume, or logical disk. Hex Editor Neo is also able to open volumes that do not have associated drive letters, that is volumes, only mounted to directory on another volume

Starting from version 6.12, Hex Editor Neo allows you to create and open volume snapshots

## Disk

In addition to volume editing, direct access to physical disk is provided by the Hex Editor Neo. Thus, you can get access to disk's partition table, boot sector and so on.

## Process Memory

With Hex Editor Neo you can view and modify the virtual memory of any running process, provided you are granted required rights. The editor gives you detailed representation of the edited process memory space.

## Scripting

Each opened document is represented by individual scripting object that implements the IDocument interface

## Opening Documents

Document represents a file, shared file, disk, volume or process memory, opened in the Hex Editor Neo for viewing and editing. Each opened document has at least one editor window associated with it. You can also have multiple editor windows for an opened document.

Hex Editor Neo does not allow you to have multiple documents for a single file, but you can open as many windows for each document as you need.
Any file opened in the Hex Editor Neo is accessed with Read-Only access. Sharing access is set to allow Read operation only. Starting from version 6.12 , if Hex Editor Neo is unable to get a Read access to a file due to sharing violation, it asks the user if he wants to create a volume snapshot and open the file from a snapshot.

Hex Editor Neo also opens disks, volumes or process memory with Read-Only access, but no sharing restrictions are applied. That is, if some other application changes the contents of the edited documents, you will need to scroll document's editor window to view changes or use the View » Refresh... command.

## Errors Opening Files

If there are any errors the editor encounters during opening files it displays the list of failed files with corresponding error messages. The following window is displayed regardless of the way you used to open files, in case editor encounters any errors trying to open them.


## Opening Files

There are several ways a file may be opened by the Hex Editor Neo.

## File Open Command

Use the File » Open » Open File... command to open the standard File Open window. Select a file or multiple files and click the Open button.
Files you select may be located either on the local computer, or on the remote computer. Make sure your connection to the remote host is fast enough before opening the remote file in the editor. Usually, 10 Mbit connection will be enough for most editing tasks.

If a "Sharing Violation" error is received, which means other process(es) have the file opened and do not allow Read access, Hex Editor Neo asks if you still want to open a file using volume snapshot. This allows you to access file data even if another process has the file locked. Note that operating system may cache file data in memory. In this case those data may not be available in the snapshot. A snapshot is automatically discarded as soon as file is closed. Support for opening locked files is available starting from version 6.12

The File Open window has a drop-down with supported formats. By default the "All supported documents" format is selected. In this mode Hex Editor Neo automatically determines the type of the file you are opening, depending on file's extension:

| Extension | Document type |
| :--- | :--- |
| . ts,.$j s$ | Script |
| . hex | Intel HEX |
| $. s 19, .528, .537$ | Motorola S-Records |
| $*$ | Binary file |

You can also force the editor to use the specified format by explicitly choosing it in the drop-down. For example, to force opening the test. hex file as binary, select the "Binary files (*)" format.

By default, if you select a shortcut in the Open File window, the shortcut's target will be opened. If you want to open a shortcut file, make sure to select the "Shortcut files" format from the format list.

## Using the Right-click Menu in the Windows Explorer

If you chose an option to integrate with Windows Explorer during Hex Editor Neo installation, theEdit with HHD Hex Editor Neocommand appears when you right click on the selected file(s) in the Windows Explorer. Select this option to open all selected files in the Hex Editor Neo.

```
Open
Print
Edit with Hex Editor
Extract files...
Extract Here
Extract to "msxml4-KB936181-enu)"
Add to archive..
Add to "msxml4-K8936181-enu.7z"
Open With...
```

This will launch the Hex Editor Neo (if it is not already running) and instruct it to open all the files you selected in the folder.
Note: Windows Explorer references the extension DLL that is used to implement the right-click menu command. If you try to uninstall the product right after using the command, it will fail to delete all installed files and will have to schedule a reboot to finish installation.

## Drag\&Drop

You may select a file or a number of files in Windows Explorer (or any other compatible application) and drag them to the running instance of the Hex Editor Neo. This will instruct the editor to open all dragged files.

## Specify Files to Open in Command Line

Launching the Hex Editor Neo with a list of files in its command line instructs the editor to open all of them
Remember that all file names that contain spaces must be enclosed in the double quotation marks. There is a limit enforced by the operating system to the maximum length of the command line. Generally, it should be less than 256 characters

Example:
PowerShell
.\HexFrame.exe c:\temp\rom.dat c:\windows\system32\calc.exe "c:\Program Files\Windows NT\dialer.exe"

Running this command will load the Hex Editor Neo and open all specified files.
Specifying the command line option /s tells the Hex Editor Neo to reuse the running Hex Editor Neo instance. If omitted, a new Hex Editor Neo instance will be launched.
The following command line switches are supported by Hex Editor Neo:

## /shared

Open the next file specified in the command line in shared mode
/readonly
Open the next file specified in the command line in read only mode.
Both Drag\&Drop and command-line methods will make the editor to automatically determine the file format.

## Scripting

Macros and scripts running in the editor may use the openFile method to open the file in the editor

## Opening Files in Shared Mode

There are several ways a file may be opened by the Hex Editor Neo in shared mode.
File Open Shared Command

Use the File » Open » Open Shared File... command to open the standard File Open window. Select a file or multiple files and click the Open button.
Files you select may be located either on the local computer, or on the remote computer. Make sure your connection to the remote host is fast enough before opening the remote file in the editor. Usually, 10 Mbit connection will be enough for most editing tasks.

```
Using the Right-click Menu in the Windows Explorer
```

If you chose an option to integrate with Windows Explorer during Hex Editor Neo installation, theEdit with HHD Hex Editor Neo (Shared Mode)command appears when you right click on the selected file(s) in the Windows Explorer. Select this option to open all selected files in the Hex Editor Neo.

Note that this option may be available only when you hold Shift key when bringing up the context menu.
This will launch the Hex Editor Neo (if it is not already running) and instruct it to open all the files you selected in the folder.
Note: Windows Explorer references the extension DLL that is used to implement the right-click menu command. If you try to uninstall the product right after using the command, it will fail to delete all installed files and will have to schedule a reboot to finish installation.

```
Specify Files to Open in Command Line
```

Launching the Hex Editor Neo with a list of files in its command line instructs the editor to open all of them.
Remember that all file names that contain spaces must be enclosed in the double quotation marks. There is a limit enforced by the operating system to the maximum length of the command line. Generally, it should be less than 256 characters. Remember to prepend each file you want to be opened in shared mode with a/shared switch.

Example:

PowerShell

Running this command will load the Hex Editor Neo and open all specified files in shared mode.
Specifying the command line option /s tells the Hex Editor Neo to reuse the running Hex Editor Neo instance. If omitted, a new Hex Editor Neo instance will be launched.

## Scripting

Macros and scripts running in the editor may use the openFile method to open the file in shared mode in the editor.

## Opening Encoded Hex Files

Hex Editor Neo supports decoding Intel HEX and Motorola S-Records files. You may decode and put them into any opened document using theEdit » Insert Hex... command or open as new documents using the File » Open » Open Hex... command.

| 98.80 Oen Hex |  | $\times$ |
| :---: | :---: | :---: |
| Insert from |  |  |
| Clipboard |  |  |
| O File: |  |  |
|  |  |  |
| Format |  |  |
| Intel HEX |  |  |
| Put the data at |  |  |
| - Start of the file Cursor location | $\square$ Insert mode |  |
|  | $\square$ Ignore data offset |  |
|  | OK | Cancel |

See the Edit » Insert Hex... topic for the window description.
This command provides additional options over generic File» Open » Open File..., which also allows opening the HEX files in the editor.
When the user saves changes made to the opened HEX file Hex Editor Neo uses original file format.

## Scripting

Macros and scripts running in the editor may use the openHexAsync method to open Intel HEX or Motorola S-Records files.

## Opening Volumes

To open a volume, execute the File » Open » Open Volume... command. Open Volume window appears:


For each detected volume, its mounting point (drive letter and/or mounting directory), label, volume type, volume's file system (if any), volume's serial number and volume's unique ID are provided. You can sort the list in ascending or descending order on any of the list's columns.

In addition, detailed information about the current volume's file system and free space is provided in the dialog
Select a volume from the list of all volumes and click the OK button to open it. Open as Read-Only option may be used to have the editor open a volume in read-only mode (default). Some volumes can only be opened in read-only mode.

Starting from version 6.12, Hex Editor Neo allows you to create a volume snapshot and open its snapshot in read-only mode. This allows you to "freeze" volume contents at a specific time point. Snapshot is automatically discarded as soon as volume is closed

## Volume Navigator Support

If supported volume is opened, Volume Navigator automatically starts parsing its structure. Volume Navigator supports the following operating systems: NTFS, FAT, FAT32, ExFAT and CDFS. It also allows the user to create a definition of a custom file system

## Scripting

Macros and scripts running in the editor may use the openVolume method to open the volume in the editor.

## Opening Disks

To open a physical disk, execute the File » Open » Open Physical Disk... command. The Open Disk window appears:


For each detected disk, its name (model name), size, list of mounted volumes, disk type and disk capabilities are displayed. You can sort the list in ascending or descending order on any of the list's columns.

Detailed disk information (disk geometry and partition table) is displayed for a selected physical disk.
Select a disk from the list of all disks and click the OK button to open it. Open as Read-Only option may be used to have the editor open a disk in read-only mode (default).

## Scripting

Macros and scripts running in the editor may use the openDisk method to open the physical disk in the editor.

## Opening Devices

Opening device feature can be used to open block-level devices, such as disks or volumes not recognized by the Opening Disks and Opening Volumes windows.


You must enter the full path of a device you want to open and specify its type. If you leave the type as Auto, Hex Editor Neo will query the device to see if it is a disk or a volume. You can also manually set the device type.

The Open as Read-Only option is used to prevent the underlying media from changes.

## Opening Processes

To open a process memory, execute the File » Open » Open Process... command. This command opens and brings the Processes Tool Window into focus.

## Creating New Documents

The Hex Editor Neo allows you to quickly create an empty document with a use of the File » New » New File command. Created document is virtual and is not backed up by any real file until you save it for the first time.

Empty document always has the size of zero bytes. You can use the wide range of editing commands to insert data to the file.
File » New » New Intel Hex and File » New » New Motorola S-Records commands allow you to create empty Intel Hex and Motorola S-Records encoded binary files.

## Scripting

Macros and scripts running in the editor may use the newDocument method to create new empty document in the editor.

## Closing Documents

Closing document in a Hex Editor Neo releases all references to the opened file, process, disk or volume, optionally saves all changes made to the document, frees all memory and disk space used to maintain the document state. The effect of closing a document will also be that the sharing restrictions set on the file at the time it was opened are cancelled, making it possible for other processes to open the file for writing.

Hex Editor Neo provides you with several ways of closing documents:

1. The document is automatically closed when all its editor windows are closed. See the topic for editor windows to find out the ways to close a window.
2. The File » Close menu command can be used to immediately close the document, regardless of the number of editor windows you have opened for it.
3. The File » Close All command is used to close all opened documents.
4. All opened documents are automatically closed when you exit the Hex Editor Neo.

Right before closing the document, the editor checks if it is in modified state. In this case, you are asked whether you want to save the changes, cancel them, or abort the close operation

The same prompt is displayed when you shut down your operating system, perform restart or log off. If you have any unsaved changes to any of the opened documents, you will be prompted whether you want to save them. This may delay the system shutdown process or even cancel it.

More information on saving changes made to documents can be found in Saving Documents section

## Scripting

Macros and scripts running in the editor may use the IDocument.close method to close the document and all associated editor windows.

## Saving Documents

All changes made to a document always reside in memory. In order to apply them, you have to save a document.
When the document has any modifications, the "*" symbol appears next to the document's name in each of its editor windows. Two of Hex Editor Neo commands, File " Save and File »Save As... are used to save documents. The first command applies changes to the document file itself, while the second one allows you to save a copy of the current version of the document, which is the version that includes all changes, to another file.

The File » Save command is not available for documents opened with Open as Read Only flag set on as well as for documents created with a File» New command. Being executed, this command behaves exactly as File »Save As... command.

## Save As Command

The Save As window appears when the user executes the File »Save As... command. The window has a formats drop-down which lists all formats the editor supports for the current document. By default, the document is saved in its original format, but the user may change that. For example, the user may choose to save the file in Intel Hex format.

## Save All Command

Use the File » Save All command to save all modified documents.

## Always Create Backups Option

There is an option on the General Settings tab, called "Always create backups" that governs the behavior of the File Save command. When it is on, the save operation does the following:

- Creates a temporary file in the same folder where the original file is located.
- Saves the entire copy of the current version of the file to the temporary file.
- Renames the original file, adding the .bak to its filename and renames the temporary file to the filename of the original file. Both these renames are performed as an atomic operation. If there already exists a file with a .bak extension, it is silently deleted.

When the "Always create backups" option is turned off (the default state), the editor determines if it is more efficient to write only the changes made to a file than to write an entire file. For example, you may open a 4GB file in the editor, modify several bytes in the middle of it and save the changes. It is obvious, that writing only a small number of changed bytes will be much faster than writing the full copy of the original file - that's why the editor prefers to have this option off by default.

If any error occurs during "optimized" file save operation, the file will be left in undefined state. It is recommended to switch this option ON if you have any vital information in the edited file.

This option also governs the behavior of Replace in Files function. If you execute the function in Replace all occurrences and save mode, the editor warns you to switch this option on, otherwise, you may lose some valuable information. With this option on, on the other hand, each modified file is backed up before overwriting.

This option does not affect streams. Hex Editor Neo never creates a backup for a modified stream. This behavior is by design and is governed by the operating system.

## Requirements

Usually, Hex Editor Neo creates a temporary file in the original file's folder, so it is required to have enough free space on the volume. Otherwise, Hex Editor Neo will not be able to save changes made to a file. Disk quota settings may also prevent the editor from saving changes.

When the "Always Create Backups" option is turned off and Hex Editor Neo determines that it is possible to do an effective save, it does not create a temporary file, and lowers its requirements on free volume space.

If you get a "not enough space" error message when trying to save a file, all your modifications are retained - you can free up the required space on the volume, or use the Save As... command to save the file on another volume.

## Scripting

Macros and scripts running in the editor may use the IDocument.saveAsync and IDocument.saveAsAsync methods to save changes made to the document.

## Working with Shared Files

Opening file in shared mode will allow other applications to continue working with a file, and moreover, to continue making changes to it.
After you open a file in a shared mode, you may update it in Hex Editor Neo and in any other application.
Working with a file opened in shared mode in Hex Editor Neo is exactly the same as working with any other type of document. The only difference is that Hex Editor Neo displays an overlay warning icon on top of file's icon in its user interface, reminding you that the file has actually been opened in shared mode.

When the file is updated in another application, Hex Editor Neo automatically detects the change and displays a Shared File Conflict Dialog It gives you options to discard or apply your changes, to discard and apply other application's changes, or both at the same time. It also allows you to close the file if you do not need it anymore.

This dialog is displayed only if Hex Editor Neo is a foreground application. If it is not, it flashes its icon on the task bar and its application window frame until you bring it to the front.

## Supporting Scenarios

Hex Editor Neo supports opening files in shared mode on any Windows operating system it supports (Windows 7 or later), any file system (FAT, FAT32, NTFS, ExtFS), and any kind of storage media (local hard disks, network shares, removable devices).

## Limitation

As opening a file in a shared mode is almost equivalent to copying a file, Hex Editor Neo warns you if you instruct it to open a file which is too large. It uses the disk system rating to guess the maximum supported size.

If the size of a file opened in shared mode exceeds the limit, a warning is displayed. You may still continue opening in shared mode if you agree to wait or instead open a file in normal, locked mode.

## Scripting

Macros and scripts running in the editor may use the openFile method to open the file in shared mode in the editor.

## Solving File Conflicts

Hex Editor Neo watches for changes in each file opened in shared mode. It is able to detect the following change types: deletion, modification and renaming.
By default, rename change detection is switched off, as it may interfere with a way several applications modify files. Imagine the following scenario:
You open a file A in Hex Editor Neo and some application B. You then make a change to a file in application B and save it. Application B follows this algorithm:

1. It creates a temporary file $T$ and writes the full copy of $A$ (with changes made) into it from its memory.
2. It renames the original file $A$ into file $C$.
3. It renames the temporary file $T$ into file $A$.
4. It deletes the file $C$

If rename change detection is ON, Hex Editor Neo catches it on the step 2, renames the document and then catches the delete notification on step 4, thus effectively missing actual file modification made on step 3. That is why this option is turned OFF by default. You may turn in on in the General Settings page using the option "Ignore renames of shared opened files".

## Rename Change Notification

When a shared file is renamed and "Ignore renames of shared opened files" option is ON, Hex Editor Neo automatically renames all openededitor windows for the file. No notifications are presented to the user.

## Delete Change Notification

When a shared file is deleted, Hex Editor Neo presents a window with two options:

## Discard all changes and close the file.

Automatically discard all changes you made to a file (if there are any) and close a file. Use this option if you intentionally deleted a file and are not going to work with it anymore.

## Continue working with a file.

Continue working with a file. Note that Hex Editor Neo stops watching for file modifications if you select this option. So, if another application creates a file with a same name in a same folder, Hex Editor Neo will not detect it.

## Modify Change Notification

When a shared file is modified, Hex Editor Neo presents one of two windows, depending on whether you have made modifications in the editor or not. Your further actions are described in the following table:

| Change type | There are no modifications in There are modifications in Hex Editor Neo Hex Editor Neo |  |
| :---: | :---: | :---: |
| File's size is unchanged | - Ignore changes <br> - Close the file <br> - Update the file | - Retain all changes and do not update the file <br> - Discard all changes and close the file <br> - Discard all changes and update the file <br> - Retain all changes and update the file (changes are merged into the new version) |
| File's size is changed | - Ignore changes <br> - Close the file <br> - Update the file | - Retain all changes and do not update the file <br> - Discard all changes and close the file <br> - Discard all changes and update the file <br> - Retain all changes and update the file (requires "Highlight changes in the file" option also be set to ON; changes are retained in old version) |

## The following table briefly describes each option:

## Ignore changes

Hex Editor Neo ignores any changes made by another application.

## Retain all changes and do not update the file

Hex Editor Neo ignores any changes made by another application. All changes made in Hex Editor Neo are retained.

## Close the file

Immediately close the file in Hex Editor Neo.

## Discard all changes and close the file

Discard any changes and close the file in Hex Editor Neo.

## Update the file

Update the file in Hex Editor Neo to reflect changes made by another application.

## Discard all changes and update the file

Discard all changes made in Hex Editor Neo and load changes made by another application.

## Retain all changes and update the file

Load changes made by another application but also retain local changes.
Note that if another application changes the shared file's size and you select the "Retain all changes and update the file" option, the "Highlight changes in the file" option must also be selected (it is done automatically).

## Highlighting Changes in the File

If you select one of the "update" options, an option to highlight changes in a file becomes available. It automatically performs a file comparison between the old and new versions of the shared file. Both algorithms, Simple Compare Algorithm and Diff Compare Algorithm are provided.

When you check this option, Hex Editor Neo renames the current document and opens a new version of the file. It then instructs theFile Compare module to compare them and highlight their differences. An option "Close older versions of the file" will instruct Hex Editor Neo to close older versions used in previous comparisons.

Note that older versions are not backed up by an actual file, but nevertheless are normal documents which can be edited, saved and so on.

Notes

Please note that Hex Editor Neo may be unable to detect a file change if too many files in the file's folder are modified at the same time.

## Working with Processes

## Process List Window

This tool window displays the interactive list of all running processes.

| Processes |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  | Search processes... |  |  |  |  |
| Name | Process ID Archit... | User | Private | Threads | Command line |
| msedge.exe | $9856 \times 64$ | COREI5... | 95.95 MB |  | "C:\|Program Files (x86) MMicrosoft|Edge Dev|Application\msedge.exe" --no- |
| msedge.exe | $17460 \times 64$ | COREI5... | 2.17 MB |  | "C:\|Program Files (x86) MMicrosoft|Edge DeviApplication\msedge.exe" --typt |
| msedge.exe | $20356 \times 64$ | COREI5... | 123.42 MB |  | "C:\|Program Files (x86) MMicrosoft|Edge DeviApplication\msedge.exe" --typr |
| msedge.exe | $18256 \times 64$ | COREIS... | 15.30 MB |  | "C:\|Program Files (x86) MMicrosoft|Edge DeviApplication\msedge.exe" --typr |
| msedge.exe | $12900 \times 64$ | COREI5... | 8.41 MB |  | "C:\|Program Files (x86)/Microsoft|Edge DeviApplication\msedge.exe" --typi |
| msedge.exe | $19824 \times 64$ | COREI5... | 17.39 MB |  | "C:\|Program Files (x86)\Microsoft|Edge DeviApplication\msedge.exe" --typi |
| msedge.exe | $22548 \times 64$ | COREIS... | 34.01 MB |  | "C:\|Program Files (x86) MMicrosoft|Edge DeviApplication\msedge.exe" --typt |
| msedge.exe | $12504 \times 64$ | COREIS... | 19.32 MB |  |  |
| msedge.exe | $11940 \times 64$ | COREIS... | 15.15 MB |  | "C:\|Program Files (x86)|Microsoft|Edge Dev|Application|msedge.exe" --typt |
| msedge.exe | $14512 \times 64$ | COREIS... | 60.04 MB |  | "C:\|Program Files (x86)\Microsoft\Edge Dev\Application\msedge.exe" --typs |

The following information is displayed for each running process:
Name
The name of the process (usually a name of its main executable file, with extension).
Process ID
Process identifier number. This number is unique among all currently running processes. The operating system may reuse the process ID as soon as the process ends.

## Architecture

Process architecture. The following architectures are supported: x86, x64 and arm64

## User

The name of the user, system or service account who launched the process.

## Private memory

The amount of "private" memory allocated by the process. This is the amount of total process memory minus the amount of mapped and shared memory.
Threads
The number of threads a process is currently running

## Command line

The full command line of the process.

## Sorting, Filtering and Tree View

The list may be sorted on each of its columns, either in ascending or in descending order. Click on the column header to change the sorting order. When sorting is switched off, Hex Editor Neo displays a "process tree", that is, it provides a parent-child relationship for running processes.

Use the "Search Processes..." filter box on a window toolbar to search for specific processes.

## Showing More Information

If Hex Editor Neo is running under non-privileged account it may be unable to query and display information about processes running under different user accounts or under system/service accounts. To display this information without the need to restart the application under privileged account, turn on the Show All Processes toolbar button. The elevation request will appear and, if completed successfully, Hex Editor Neo will begin showing more information about running processes.

Elevation consent is cached, you won't need to elevate again for other privileged tasks, until you switch the Show All Processes mode off.

## Terminating Processes

Hex Editor Neo allows you to terminate any running process (provided you have sufficient rights to do so) by executing the Terminate Process command from the context menu.

## Opening Processes

In order to retrieve more information about running process, see its memory layout, create and manage process snapshots and open process memory for viewing/editing, use the Open Process or Open Process Snapshot commands.

The former command opens the process window, while the second command also opens the process window and additionally automatically create a process snapshot
Press the Refresh button to refresh the list of running processes or manage automatic refresh using the Auto Refresh switch on the toolbar.

## Scripting

The user script may use the findProcess and findProcesses global methods to obtain information about running processes.

## Process Window

Process Window displays information about the running process.


If the opened process exits, process window is updated: process icon goes gray, "Status: Running" message is replaced with a process's exit code and "Running" snapshot is replaced with an "Exited" snapshot.

Once process exits, you are no longer able to view and use the "Running" snapshot, but may still use any created process snapshots.

## Snapshots List

Hex Editor Neo allows you to create any number of light-weight process memory snapshots. A process snapshot is a "view" into the process memory in a given point of time. There is always a virtual "Running" snapshot that references a live state of the running process's memory. If process exits, it is replaced with an empty "Exited" snapshot.

Use the Create Snapshot command to create new memory snapshots. Use the Delete Snapshot command to delete selected snapshots and Clear Snapshots command to delete all process snapshots.

If the process window is closed (or Hex Editor Neo exits), all process snapshots are automatically deleted.
Select a snapshot and use the Debug Snapshot command to launch an external debugger or Save Dump command to save a process snapshot as a memory dump file.
Select two snapshots and execute the Compare Snapshots command to compare the contents of snapshots memory. You cannot compare a snapshot with the "Running" snapshot.

## Memory Layout

This list shows the process memory layout. It supports two modes of operation: Blocks and Modules. Use the switch on the toolbar to select the display mode.
When a memory block or blocks are selected in either of the lists, the starting address of the selection and its size is displayed at the bottom the of the process window. Use the Open (Read-Only) and Open (Read/Write) commands to open the document for viewing or editing of the selected process memory block(s).

## Blocks

The following information is displayed for each memory block in the process address space:

## Address

Block starting address.

## Size

Size of the block.
Type
Type of block: Unknown, Private or Image.
State
Block allocation state: Free, Reserved or Committed.
Flags
Memory protection and access flags.

## Module

If the block is part of a mapped module, contains the module's name.
The list may be sorted on any column in either ascending or descending order. If the list is sorted on the Address column, multiple blocks may be selected, otherwise, only single block may be selected.

## Modules

The following information is displayed for each module mapped into the process's address space
Name
Module name.

```
Address
Module starting address Size
Module mapping size.
Path
Full path to the mapped module.
```

The list may be sorted on any column in either ascending or descending order. If the list is sorted on the Address column, multiple modules may be selected, otherwise, only single module may be selected.

## Integration with Disassembler

When single module is selected, the Run Disassembler command may be used to disassemble the selected mapped module.
The built-in Hex Editor Neo'sDisassembler component is used to disassemble the in-memory representation of a selected module. Please note that a module mapped into the process's address space may be different from its disk image, for example, if it was modified in memory.

## Memory Map

When the snapshot is selected, its "memory map" is displayed on the right of the process window. It displays the relative location and size of each allocated memory block.
When 64-bit version of Hex Editor Neo is running on Windows x64 or Windows ARM64 computer, a special logarithmic representation of the process address space is shown. Otherwise, nearly empty map of 128 TB address space of a running process would be impossible to use. Please note, however, that in this case relative size of displayed blocks does not directly correlate to relative size of real memory blocks.

## Scripting

Each process window is represented by a process window object in running scripts. Script may use the openProcess method to open a process and get a reference to an object. A process window object supports methods and properties to carry any task that is available in the process window user interface.

## Process Snapshots

Process Snapshot is a light-weight copy of a full process memory. It consists of all allocated memory pages, can be quickly created and discarded on demand and may outlive the parent process.

Hex Editor Neo provides an ability to create a memory snapshot using the commands in process window. Once created, the following options are available:

## WARNING

Creation of process snapshots requires Windows 8 or later operating system.

## Opening Snapshot Memory Regions

The user may open either an entire snapshot address space or only a sub-portion of it for viewing in the editor. Opening for read/write access is not available for obvious reasons.

Running scripts may use the IProcessSnapshot.openRegion method to execute this task.

## Comparing Snapshots

The user may compare the memory from two process snapshots using the built-in File Comparison module. In order to compare snapshots, they first need to be selected in the Snapshots list in the process window and Compare Snapshots command need to be executed. Comparing a snapshot with a running process is not allowed.

Running scripts may use the IProcessWindow.compareSnapshots method to compare snapshots
Hex Editor Neo uses an effective algorithm to find the memory that changed between the given snapshots.

## Launch External Debugger

An external debugger may be launched for a selected snapshot. Opening a snapshot in a debugger allows the user to investigate the state of all process threads at a given point in time, including their contexts and call stacks, their referenced variables and memory.

This option may be executed against the "Running" snapshot.
The "Include full memory when debugging snapshots" option in General Settings Page govern how Hex Editor Neo prepares information to be passed to a debugger. If this option is off (default), only directly referenced memory will be available for the debugger. This is usually enough for most debugging purposes and allows the debugger to be launched much faster and efficiently. When this option is on, the full snapshot memory will be available for the debugger, however, execution of the command will take longer.

## NOTE

The full process memory is always available for the debugger when it is launched for the "Running" snapshot, that is, live process. In this case, the process may get suspended when debugger is launched.

Hex Editor Neo is compatible with two debuggers: WinDBG (including "legacy" WinDBG and new WinDBG Preview, available from Microsoft Store) and Visual Studio. It can also use the debugger that is currently registered as system default JIT debugger. Use the "Preferred external debugger" option inGeneral Settings Page to select what debugger to use.

Running scripts may use the IProcessSnapshot.debug method to execute this task.

## Save Snapshot Dump

It is possible to save a dump file of the selected snapshot. This command also supports a "Running" snapshot. The "Include full memory when debugging snapshots" option in General Settings Page controls the amount of memory written to a dump file. It is recommended to leave that option off, unless the specific memory blocks are missing in the resulting dump file.

To execute this command, select the snapshot in the process window and choose the Save Dump option from the context menu.

The resulting dump file may be used to launch the debugger or to investigate the process address space on another computer.
Running scripts may use the IProcessSnapshot.saveDump method to execute this task.

## Editor Windows

Editor window is the main Hex Editor Neo element provided for file viewing and editing. These windows are used to display and edit the contents ofopened files and other documents. This section describes the editor windows in detail.

The new window is opened each time you open or create a new document. Additional windows for a document may also be opened using the Window » New Window command. Another way for opening additional editor windows for a document, window splitting is described later in this section.

## Basic Elements

Here's a typical editor window:


There are the following elements in each editor window:

## Title Bar

Title bar displays the full file name and a close button. Holding a mouse over the title bar displays a tooltip with a full path to the file. Clicking on a close button closes the current editor window. If this is a last window for a given document, the document is closed. In the latter case, you are asked if you want to save any changes made to the document, if there are any.
The Window » Close Window command may also be used to close an editor window.
Clicking on a title bar and dragging starts editor window placement customization.

## Split Anchor

Split anchor is used to split an editor window into two windows horizontally. Click and drag with a left mouse button to split a window. As a result of your operation, a new editor window is created. It is another view of original window's opened document. Each editor window has its own cursor position, number of columns and view type and its own selection.
After the window has been split, the split anchor disappears, but you can use the splitter bar between two windows to change their relative sizes. As soon as you move the splitter bar to window's top border, the top window is hidden and the split anchor appears again. Note, that the hidden window is not destroyed and all its properties are restored as soon as you split window again.
Hex Editor Neo command Window "Split is used to split the window without using a mouse. If the window has already been split, use this command to hide the top view.

## Scroll Bars

As most opened documents are large enough not to fit into their editor windows, scroll bars are used to scroll the file content's through the view, provided by the window. Horizontal scrolling is used to scroll the window's content horizontally, if the current window's size and layout does not allow all the information to be visible. Vertical scrolling is a fast and convenient way to navigate to any portion of the file.
Right-clicking on the scroll bar displays a shortcut menu with following commands (vertical/horizontal):

- Scroll Here - scrolls to the mouse position
- Top/Left Edge - scrolls to the beginning of the file/first column
- Bottom/Right Edge - scrolls to the end of the file/last column
- Page Up/Page Left - scrolls a few rows/columns backwards/to the left
- Page Down/Page Right - scrolls a few rows/columns forward/to the right
- Scroll Up/Scroll Left - scrolls one row/column backwards/to the left
- Scroll Down/Scroll Right - scrolls one row/column forward/to the right

The vertical scrollbar is disabled if the whole file is visible on the screen. Horizontal scrollbar is disabled if the window fits horizontally and all columns are visible.

## Columns' Numbers

To make a file contents more readable, editor windows display the number of each column on the first row. Columns are numbered in hexadecimal, decimal or octal numbers (depending on the way Address Area is displayed) with first column having number zero. The cursor's current column is highlighted if it is visible on a screen.

Each view can be configured to display any number of columns in a row. By default, 16 columns are displayed. The View " Columns command group can be used to change the number of columns. You can choose between standard 4, 8, 16 or 32 columns, specify your own number or select "Auto". In the latter case, the number of columns is determined automatically depending on the window's width.

When actual number of columns is a power of 2, smart indentation may be applied to a window. In this case, editor window inserts small gaps between columns in a way to greatly increase data readability.

NOTE
Hex Editor Neo supports any number of columns from 1 to 16384.

## Address Area

For convenience, the address of the first cell in a row is displayed at the beginning of each row. You can configure the address to be displayed as a hexadecimal, decimal or octal number.

The cursor's current row is highlighted and address of the first cell in a cursor row is temporary replaced by the cursor's address. You may switch this behavior off in the Tools » Settings... » General Tab.

## Navigation Area

Navigation area displays the main cursor offset. When you hover a mouse over the navigation area, it is replaced by a small window which allows you to enter the specific offset, store the current offset or copy it to the Clipboard. Navigation window's toolbar, as other tool window toolbars can be customized and more commands can be added to it.

## Data or Code Pane

This pane displays the document's content. Data is displayed in a number of cells, where the cell is a byte, word, double word, quad word, float or double. Cell type is individually configured for each editor window through the View » Display As command group.
More information on data pane is provided further in the documentation.
You can turn code pane on or off at any time using the View » Code Pane command.

## Text Pane

This optional pane provides a textual representation of document's content. In editing binary files, it is sometimes convenient to display data in both binary and textual form. This pane is allowed only for Bytes and Words grouping modes.

In word view types, document's data is displayed as an UNICODE (UTF-16) text while in byte view types the window's currentencoding type is used to display document's data. You can change the window's encodings through the shortcut menu, or through the View » Encodings menu.

You can turn text pane on or off at any time using the View » Text Pane command.

## Cursor

Cursor is a position where subsequent editing commands will take place. This includes typing, inserting or deleting data. The cursor is also used as an anchor point for keyboard selection.

More information on cursor movement and keyboard selection are provided in corresponding sections. Editor windows support multiple cursors. When multiple cursors are present, one of them is considered main, or default cursors and others are called alternate cursors. A main cursor can be changed with the Ctrl + Left Arrow and Ctrl

+ Right Arrow keyboard combinations.


## Scripting

Each editor window is represented by individual scripting object that implements the IDocumentView interface.

## Encodings

There are a lot of different character encodings that describe how characters of some specific alphabet are encoded in single or multi-byte codes. The editor window text pane is used to display the textual representation of the document. The text it displays may of course be interpreted according to one or another encoding.

Hex Editor Neo allows you to choose the editor window's encoding from a wide set of supported encodings. As long as an encoding is a property of individual editor window, you can set different encoding for each editor window, even if they represent the same document.

Below is a full list of supported encodings.
NOTE: Support for a specific encoding depends on installed Windows code pages and fonts. If required components cannot be found for a selected encoding, the "Encoding not supported" text is displayed instead of document's data. Typing in text pane is also disabled until you select another, supported encoding.

| Encoding | Encoding | Encoding |
| :---: | :---: | :---: |
| Default ANSI | Default OEM | UTF-8 |
| ANSI - Arabic | ANSI - Baltic | ANSI-Central European |
| ANSI - Cyrillic | ANSI-Greek | ANSI - Hebrew |
| ANSI - Latin I | ANSI - Turkish | Arabic - ASMO 449+, BCON V4 |
| Arabic - ASMO 708 | Arabic - Transparent Arabic | Arabic - Transparent ASMO |
| ISO 2022 Japanese JIS X 0201-1989 | ISO 2022 Japanese with halfwidth Katakana | ISO 2022 Japanese with no halfwidth Katakana |
| ISO 2022 Korean | ISO 2022 Simplified Chines | ISO 2022 Traditional Chinese |
| ISO 6937 Non-Spacing Accent | ISO 8859-1 Latin I | ISO 8859-15 Latin 9 |
| ISO 8859-2 Central Europe | ISO 8859-3 Latin 3 | ISO 8859-4 Baltic |
| ISO 8859-5 Cyrillic | ISO 8859-6 Arabic | ISO 8859-7 Greek |
| ISO 8859-8 Hebrew | ISO 8859-8 Hebrew | ISO 8859-9 Latin 5 |
| IBM EBCDIC - Arabic | IBM EBCDIC - Cyrillic (Russian) | IBM EBCDIC - Cyrillic (Serbian, Bulgarian) |
| IBM EBCDIC - Denmark/Norway | IBM EBCDIC - Denmark/Norway (20277 + Euro symbol) | IBM EBCDIC - Finland/Sweden |
| IBM EBCDIC - Finland/Sweden (20278 + Euro symbol) | IBM EBCDIC - France | IBM EBCDIC - France (20297 + Euro symbol) |
| IBM EBCDIC - Germany | IBM EBCDIC - Germany (20273 + Euro symbol) | IBM EBCDIC - Greek |
| IBM EBCDIC - Hebrew | IBM EBCDIC - Icelandic | IBM EBCDIC - Icelandic (20871 + Euro symbol) |
| IBM EBCDIC - International | IBM EBCDIC - International (500 + Euro symbol) | IBM EBCDIC - Italy |
| IBM EBCDIC - Italy (20280 + Euro symbol) | IBM EBCDIC - Japanese Katakana Extended | IBM EBCDIC - Korean Extended |
| IBM EBCDIC - Latin 1/Open System | IBM EBCDIC - Latin America/Spain | IBM EBCDIC - Latin America/Spain (20284 + Euro symbol) |
| IBM EBCDIC - Latin-1/Open System (1047 + Euro symbol) | IBM EBCDIC - Modern Greek | M EBCDIC - Multilingual/ROECE (Latin-2) |
| IBM EBCDIC - Thai | IBM EBCDIC - Turkish | IBM EBCDIC - Turkish (Latin-5) |
| IBM EBCDIC - U.S./Canada | IBM EBCDIC - U.S./Canada (037 + Euro symbol) | IBM EBCDIC - United Kingdom |
| IBM EBCDIC - United Kingdom (20285 + Euro symbol) | ISCII Assamese | ISCII Bengali |
| ISCII Devanagari | ISCII Gujarati | ISCII Kannada |
| ISCII Malayalam | ISCII Oriya | ISCII Punjabi |
| ISCII Tamil | ISCII Telugu | MAC - Arabic |
| MAC - Croatia | MAC - Cyrillic | MAC - Greek I |
| MAC - Hebrew | MAC - Icelandic | MAC - Japanese |
| MAC - Korean | MAC - Latin II | MAC - Roman |
| MAC - Romania | MAC - Simplified Chinese (GB 2312) | MAC - Thai |
| MAC - Traditional Chinese (Big5) | MAC - Turkish | MAC - Ukraine |
| OEM - Arabic | OEM - Baltic | OEM - Canadian-French |
| OEM - Cyrillic (primarily Russian) | OEM - Greek (formerly 437G) | OEM - Hebrew |
| OEM - Icelandic | OEM - Latin II | OEM - Modern Greek |
| OEM - Multilingual Latin I | OEM - Multilingual Latin I + Euro symbol | OEM - Nordic |
| OEM - Portuguese | OEM - Russian | OEM - Turkish |
| OEM - United States | Japanese (Katakana) Extended | Japanese (Latin) Extended and Japanese |
| JIS X 0208-1990 \& 0121-1990 | Korean (Johab) | Korean Extended and Korean |
| Simplified Chinese | Simplified Chinese (GB2312) | Simplified Chinese Extended and Simplified Chinese |
| Russian - KOI8-R | T. 61 | TCA - Taiwan |
| TeleText - Taiwan | Ukrainian (KOI8-U) | US/Canada and Japanese |
| US/Canada and Traditional Chinese | US-ASCII (7-bit) | Wang - Taiwan |
| CNS - Taiwan | Eten - Taiwan | EUC - Japanese |
| EUC - Korean | EUC - Simplified Chinese | EUC - Traditional Chinese |
| Europa 3 | HZ-GB2312 Simplified Chinese | IA5 German (7-bit) |
| IA5 IRV International Alphabet No. 5 (7-bit) | IA5 Norwegian (7-bit) | IA5 Swedish (7-bit) |
| IBM5550 - Taiwan |  |  |

## Working with Encodings

The current editor window's encoding is displayed on the status bar:

```
    00) File... WW Book... %o NTFS... Dilu
```

    ). 4.79 MB Hex bytes, 16 , Default ANSI OVR
    Text pane displays text data according to selected encoding. When you type new data on the keyboard (with text pane active), typed characters are processed according to selected encoding.

When the editor window is displaying data in Hex Words or Decimal Words view type, the UNICODE (UTF-16) encoding is automatically selected (as the text pane displays UNICODE data in these modes)

To change the current window's encoding, open the shortcut menu, select "Encoding" item and choose an encoding from the list. The list consists of "Default ANSI", "Default OEM", 5 recently used encodings and the "Other" item. Selecting the Other item opens a full list of supported encodings.

## UTF-8 Support

UTF-8 is the first (and only, for now) multi-byte encoding supported by the editor.
The editor provides the full support for UTF-8 encoding. It not only displays the text encoded in UTF-8, but also allows you to type new data in this encoding. When you type, entered characters are converted on-the-fly and a single entered character may occupy up to 4 bytes.

When a character occupies several bytes, a space character is displayed in all but the last cell (in Text View). The last cell displays the character itself.
UTF-8 encoding defines strict rules for encoding UNICODE characters into single, two, three or four bytes. If these rules are broken and Hex Editor Neo cannot decode the character, it displays the '?' character for each cell that contains invalid data.

All editor features, such as Find, Fill and so on are compatible with current editor window's encoding, and, therefore, are capable of working with UTF-8 as well

## Scripting

All encodings are available for scripts as Encodings enumeration. Use the IDocumentView.encoding property to query or set the editor window's encoding

## Byte Order

In computing, two basic number representation schemes are used: Little-Endian and Big-Endian. In little-endian scheme, multi-byte values are stored starting from least significant byte to most, and vice-versa in big-endian scheme

For example, the double word value $0 \times 12345678$ will be stored by little-endian computer as

## 78563412

And by big-endian computer as:
12345678
x86 and x64 processors use little-endian encoding, while some other processors from Motorola ${ }^{\circledR}$ and IBM ${ }^{\circledR}$ use big-endian encoding
Hex Editor Neo allows you to change the byte order for each editor window individually. Please note that any changes will be visible only in words, double words and quad words view types

Editor Layout settings page allows you to specify the default byte order for new editor windows.
To change a byte order for the opened window, use the commands available in the View "Byte Order menu. By default, the Ctrl+E key combination is bound to the View » Byte Order » Little-Endian command, while the Ctrl+Shift+E key combination is bound to the View » Byte Order » Big-Endian command

## Effect on Floating-Point Types

Floating-point standard (IEEE 754) does not define exact encoding of floating-point data types on little-endian and big-endian computers. According to the standard, encoding should not be affected by the change of byte order, although, floating-point type's bytes are actually swapped on several big-endian platforms.

Hex Editor Neo supports scenarios, where floating-point types are not affected by big-endian byte order, and where floating-point types are affected by the byte order change. The "Byte order change affects floating-point types" option on the General Settings page controls this. This option is ON by default.

## Data Inspector and Structure Viewer

Data Inspector and Structure Viewer modules also take the current window's byte order into account when they display and process data

## Scripting

Scripts running in the editor may query and change the editor window's byte order using the IDocumentView.bigEndian property.

## Simple Editing

## Cursor Movement and Navigation

Cursor is an important concept in any editor. It displays the position where the next editing command will take place.
Hex Editor Neo distinguishes between the navigation cursor and editing cursor. The cursor can be in any of these two types at any given time. Most of the time it is in navigation mode which is described in this section. See data modification section for more information on editing cursor.

As any editor window may have data pane and text pane, which essentially display the same data, cursor position applies to both panes simultaneously. To distinguish between the active and inactive pane, the cursor image in active pane is outlined, while in inactive pane it is not outlined. You can switch between data and text pane either by pressing the TAB key, or using your mouse.

When the window does not have keyboard focus, the cursor image is not outlined in both panes.

## "Unlimited" Editing Space

Hex Editor Neo utilizes the unique feature called "unlimited editing space". This means that you are not limited by the current file's size during editing. After the file data ends, the special "empty" data character is displayed. It consists of "." symbols and is usually painted with different color (see Color Schemes for more information). Cursor movement is therefore is not limited by the file size. You are allowed to move cursor at any position beyond the end of file and then start typing data or apply any editor command.

If new data inserted or any editor command applied after the end of the file, the corresponding number of zeroed cells is inserted to extend the size of the file. This operation has constant time complexity. It also does not consume memory or disk space so it can be considered a "cheap" operation.

For convenience, the vertical scroll bar is scaled to display the real current size of the edited document. This allows you to quickly position a cursor to a real position within the document. Several keyboard navigation keys also respect the real current document size as described below.

## Multiple Cursors

Hex Editor Neo supports multiple navigation and editing cursors. An additional cursor is added by holding the Alt key when left-clicking pressing on the cell. When multiple cursors are present, one of the cursors is considered "main", or "default". Use the Ctrl+Left Arrow and Ctrl+Right Arrow key combinations to switch between cursors.

Keyboard Movement

The cursor may be driven by keyboard keys and key combinations. These key combinations repeat those found in almost every text/binary editor and cannot be customized Left Arrow

Moves all cursors one cell to the left. If the current cell is first in a row, the cursor goes to the last cell in a previous row. If the current cell is the very first cell in a file, the cursor position is not changed.

## Right Arrow

Moves all cursors one cell to the right. If the current cell is last in a row, the cursor goes to the first cell in a next row. According to the "unlimited editing space" rule, the cursor movement is not restricted.
Up Arrow
Move all cursors one row up. If the current row is a very first row in a file, the cursor position is not changed.
Down Arrow
Move all cursors one row down. According to the "unlimited editing space" rule, the cursor movement is not restricted.
Home
Moves all cursors to the first cell in a their corresponding rows
End
Moves all cursors to the last cell in a their corresponding rows
Pgup
Moves all cursors several rows up. Number of rows to move is selected in such a way that the first visible row becomes last. If the resulting cursor position lies before the beginning of the file, the cursor is moved to the beginning (zero offset).

PgDn
Moves all cursors several rows down. Number of rows to move is selected in such a way that the last visible row becomes first. According to the "unlimited editing space" rule, the cursor movement is not restricted

Ctrl+Home
Collapses all alternate cursors and moves the cursor to the beginning of the file.
Ctrl+End
Collapses all alternate cursors and moves the cursor to the (real) end of the file.
tAB
Activates the text pane if data pane is active and data pane if text pane is active.
If the cursor moves out of the window as a result of keyboard action, the window is automatically scrolled so the cursor is visible again. Several alternate cursors may collapse as a result of a navigation operation.

## Navigating with Mouse

To scroll the contents of the window with a mouse, use the scroll bars or mouse wheel. Rotating the mouse wheel scrolls the contents of the window up or down beveral rows at a time. Scrolling before the beginning of the document is not allowed, scrolling down is always allowed according to the "unlimited editing space" rule.

It is possible to scroll the window in such a way that all cursors are out of the view using mouse navigation. To quickly scroll back to the cursor, press one of the keyboard navigation keys or start modifying data. Alternatively, left-click any visible cell to move the cursor to it.

## Navigating with Touch

To scroll the contents of the window, swipe the window up or down. The amount of rows the window scrolls depends on the speed of a swipe gesture.

## Cursor Movement with Mouse

Mouse can also take part in cursor movement.
Left clicking any cell (in any visible pane) moves the cursor to that cell. This action removes all alternate cursors. Hold the Alt key to add new alternate cursor (or remove one, if clicked on an alternate cursor).

Left-clicking on the cursor cell again enters the edit mode. In hexadecimal view types the actual position of this second click is used to move the caret to specific digit within the cell. In decimal or floating-point view types, the entire cell is always selected for editing.

## Scripting

All cursor movement actions can be used from scripts. IDocumentView.cursor property allows the script to control editor window cursor(s), including alternate cursors, cursor navigation and other related actions.

## Insert Mode

Editor windows operate in two modes: overwrite mode and insert mode. The mode is a property of individual editor window. That means that each opened window may be in a different mode.

To switch between modes use the Edit » Insert Modecommand (Ins key by default). The state of the command item denotes the current mode - pressed (or checked) means insert mode and unpressed (unchecked) means overwrite mode. Insert mode for the current editor window is also displayed on the status bar.
$39(57) \quad$ Size: $0 \times 002 f 1000(3,087,360): 2.94 \mathrm{MB} \quad$ IINS Hex bytes, 16

By default, each opened editor window is in overwrite mode.

## nfluence on Editor Command

Insert mode changes the behavior of several editor commands. Below is a list of commands whose behavior changes when insert mode is enabled:

- Typing - when insert mode is on, new data does not overwrite existing. Instead, the file size is increased, part of the file lying beyond the cursor position is moved and typed data is inserted at the cursor position.
- Insert File - in overwrite mode the file's content overwrites current file's content (possibly increasing file's size). In insert mode a file is inserted at the cursor position, moving remaining part forward. The file's size is always increased.
- Paste - in overwrite mode the Clipboard's content overwrites current file's content (possibly increasing file's size). In insert mode Clipboard's content is inserted at the
cursor position, moving remaining part forward. The file's size is always increased.
Other editor commands are not influenced by the insert mode. Insert mode fully supports multiple cursors feature.


## Scripting

Scripts running in the editor may query and change the editor window's insert mode using the IDocumentView.insertMode property.

## Data Modification

Direct data modification is a simplest editing action provided by the Hex Editor Neo.
First, we'll describe all supported view types in more detail.

## View Types

Hex Editor Neo supports a number of different view types. A view type is a property of individual editor window, therefore, you can have several editor windows (for the same or different documents) with different view types. Hex Editor Neo supports hexadecimal, decimal, octal and binary data representation. In addition, two floating-point formats are also supported. For some of these representations, it allows you to select data grouping, according to the following table:

| Representation | Bytes | Words | Double Words | Quad Words |
| :---: | :---: | :---: | :---: | :---: |
| Hexadecimal | 0 to $\mathrm{FF}_{16}$ | 0 to $\mathrm{FFFF}_{16}$ | 0 to FFFF, FFFF $_{16}$ | 0 to FFFF,FFFF,FFFF,FFFF 16 |
| Decimal ${ }^{1}$ | $\begin{aligned} & 0 \text { to } 255_{10} \text { (0 to } \\ & 2^{8-1} \text { ) } \end{aligned}$ | $\begin{aligned} & 0 \text { to } 65,535_{10} \text { (0 to } \\ & 2^{16-1} \text { ) } \end{aligned}$ | 0 to 4,294,967,295 ${ }_{10}$ (0 to $2^{32-1}$ ) | $\begin{aligned} & 0 \text { to } 18,446,744,073,709,551,61510 \text { ( } 0 \text { to } \\ & 2^{64-1} \text { ) } \end{aligned}$ |
| Octal | 0 to 3778 | 0 to 177,7778 | 0 to 37,777,777,7778 | 0 to 1,777,777,777,777,777,777,7778 |
| Binary | 0 to 28-1 (binary) | 0 to 216-1 (binary) | 0 to 232-1 (binary) | 0 to 264-1 (binary) |
| Float ${ }^{2}$ | N/A | N/A | Single-precision floating-point number (IEEE 754) | N/A |
| Double2 | N/A | N/A | N/A | Double-precision floating-point number (IEEE 754) |

You can change the editor window display type using the View » Display As command, and change grouping using View » Group By command, both of which are available through the popup menu.

## Modifying File Data

When you move the cursor in an editor window, you are working with navigation cursor. In order to be able to modify the document's data you need to switch to editing cursor. This can be achieved using one of the following ways:

- Execute the Edit» Edit Cell command (F2 by default). This will switch the cursor mode. For hexadecimal and octal view types, the caret is placed at the first cell's digit. For other view types, the entire cell is selected, like in standard edit control.
- Double-click on a cell to start editing it. For hexadecimal view types, the caret is placed at the first cell's digit. For other view types, the entire cell is selected, like in standard edit control.
- For hexadecimal and octal view types, click on a specific digit in the current cell. This switches into the edit mode and moves the caret to a digit you clicked
- Press a ' 0 ' - ' 9 ' key, 'A' - ' $F$ ' key (for hexadecimal view types) to start editing the current cell when data pane is active. Press any alpha-numeric key to change the current cell's value when text pane is active

Once in an edit mode, only the ' 0 ' - ' 9 ' keys and ' $A^{\prime}$ - ' $F$ ' keys (for hexadecimal view types) may be pressed if data pane is active and any alpha-numeric keys may be pressed if text pane is active. Navigation keys cancel the edit mode, except for the right and left keys, which moves the caret within the current or neighbor cells.

For hexadecimal view types, when last digit in a cell is modified, the cursor automatically moves to the next cell, continuing to stay in edit mode. The caret is moved to the next cell's first digit. Pressing Esc key cancels any modification made to the current cell. Pressing Enter key commits any modifications made to the current cell, moves the cursor to the next cell and cancels edit mode.

For decimal and floating-point view types, press Enter key to accept changes and move to the next cell or press Esc key to cancel any changes made to the cell. Arrow keys work only within the current cell.

For floating-point view types, you may use the sign symbol ('-'), floating-point symbol ('.') as well as exponent symbol ('e') to enter floating-point values. For example, the following floating-point values are correct

```
1.2
.
```

The last "scientific" form represents the $2.34 \cdot 10^{-23}$ number.
f multiple cursors are present in editor window, data modification applies to each cursor location. Current Insert Mode is also taken into account when calculating the resulting cursors positions

## Scripting

Scripts may use IDocumentView.typeChar and IDocumentView.editCell methods to modify document data

1. When "Digit Grouping" option is enabled (default), regional settings govern the formatting of the decimal numbers. For example, 1351680 decimal number will be displayed as 1,351,680 for standard U.S. regional settings.
2. Float is a single-precision floating-point number, encoded according to the IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754). Double is a double-precision floating-point number, defined in the same IEEE standard. These types correspond to the float and double C/C++ fundamental language types as well as floating-point types in most other programming languages.

Editor Windows List
Use the Window » Windows... command to access the window list.


Use the commands provided by this window to view the list of all opened editor windows，save changes made to documents，close one or more editor windows or activate specific editor window．

## Editor Windows Placement

Hex Editor Neo provides an extremely convenient way of organizing the work space when working with more than one document or with more than one editor window．
The workspace is divided into one or more frames．Each frame contains one or more editor window．

| $\square$ explorer．exe $\times$ 包 hh．exe |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 三 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 000000b 7 | 0001 |  |  | 04 | 05 | 0607 |  | 09 | Oa | 0b | Oc | Od | 0e |  |  |  |
| 00000040 | 0 l 1f | ba | Oe | 00 | b4 | 09 cd |  | b8 | 01 | 4 c | cd | 21 | 54 | 68 | ．．є．．r．H！ë．LH！Th |  |
| 00000050 | 6973 | 207 |  | 72 | 6f | 6772 | 61 | 6d | 20 | 63 | 61 | 6e | 6 e | 6 f | is program canno |  |
| 00000060 | 7420 | 626 |  | 20 | 72 | 75 6e | 20 | 69 | 6 e | 20 | 44 | $4 f$ | 53 | 20 | $t$ be run in dos |  |
| 00000070 | 6d 6f | 646 | 65 | 2 e | Od | Od Oa | 24 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | mode．．．．S． |  |
| 00000080 | 36 ad |  |  | 72 | cc | 2a 22 | 72 | cc | 2a | 22 | 72 | cc | 2a | 22 | 6．DqrM＊ $\mathrm{rm}^{*}$＂rm＊＂ |  |
| 00000090 | 7 b b4 | b9 |  | 7 P | cd | 2a 22 |  | be | 29 | 23 | 76 | cc | 2a |  |  |  |
| 000000a0 | al be | $2 f$ |  | 51 | cc | 2a 22 |  | cc | 2b | 22 | 96 | c5 | 2a | 22 | ys／\＃＠M＊＂rM＋＂－E＂ |  |
| 00000060 | a1 be | 2 b |  | 51 | cc | 2a 22 | a1 | be | 27 | 23 | 38 | cd | 2a | 22 |  |  |
| 000000c0 | al be | 2 e |  | 50 | cc | 2a 22 |  |  |  |  | 73 | cc | 2a |  |  |  |
| 000000d0 | a1 be | 2a |  | 73 | cc | 2a 22 |  | be | d5 | 22 | 73 | cc | 2a | 22 |  |  |
| 000000e0 | al be | 28 |  | 73 | cc | 2a 22 | 52 | 69 | 63 | 68 | 72 | cc | 2a | 22 | y̆s（\＃sm＊＂Richrm＊＂ |  |
| 000000fo | 0000 | 00 |  | 00 | 00 | 0000 |  | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |  |
| － |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ＊ |
| （2）HelpPane．exe $x$ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ＝ |
| 00000000 | 0001 |  |  | 04 | 05 | 0607 | 08 | 09 | 0a | 0b | Oc | Od | Oe | of |  |  |
| 00000000 | 4d 5a |  |  | 03 | 00 | 0000 | 04 | 00 | 00 | 00 | ff | ff | 00 | 00 |  |  |
| 00000010 | b8 00 |  |  | 00 | 00 | 0000 | 40 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | e．．．．．．．＠． |  |
| 00000020 | 0000 |  |  | 00 | 00 | 0000 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |  |  |
| 00000030 | 0000 |  |  | 00 | 00 | 0000 | 00 | 00 | 00 | 00 | e8 | 00 | 00 | 00 | ．．．．．．．．．．．．． |  |
| 00000040 | 0 e 1 f |  |  | 00 | b4 | 09 cd |  | b8 | 01 | 4 c | cd | 21 | 54 | 68 | ．．є．．r．H！ë．LH！Th |  |
| 00000050 | 6973 |  |  | 72 | 6 f | 6772 | 61 | 6d | 20 | 63 | 61 |  | 6 e | 6 f | is program canno |  |
| 00000060 | 7420 |  |  | 20 | 72 | 75 6e | 20 | 69 | 6e | 20 | 44 | 4 f | 53 | 20 | $t$ be run in dos |  |
| 00000070 | 6d 6f |  |  | 2e | Od | Od Oa | 24 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | mode．．．．\＄．．．．．．． |  |
| 00000080 | co 1f |  | 41 | 84 | 7 P | eb 12 | 84 | 7 e | eb | 12 | 84 | 7 e | eb | 12 |  |  |
| 00000090 | 8 d 06 |  |  | 86 | 7 e | eb 12 | 57 | Oc | ef | 13 | a 4 |  | eb | 12 |  |  |
| 000000a0 | 57 Oc | ee 1 |  | 8a | 7 | eb 12 | 57 | Oc | e8 | 13 | 81 | 7 e | eb | 12 | W．o．љนл．W．и．广～л． |  |
| 000000b0 | 57 Oc | ea 1 |  | 9d | 7e | eb 12 | 84 |  | ea | 12 | f2 | 7f | eb | 12 | W．к．кк～л．，，～к．т．л． |  |
| ¢ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | － |

Each editor window is represented by a tab in a frame．The document＇s file name is displayed on each tab．Click on a tab to switch to the window．If there are too much tabs in a frame that they cannot fit in it，extra tabs are not displayed and hidden windows may be accessed using the Window List button at the top－right corner of the frame．

When you create new editor window，it is placed in the first visible frame．To create a new frame，click on a tab and drag it away from other tabs．Use the same actions as in tool window customization to rearrange editor windows into different frames．

Windows may be freely moved between frames，and inter－frame splitter bars may be used to change frames＇relative sizes．When you close or move away the last frame＇s window，the frame is closed．

Hex Editor Neo organizes tool windows and editor windows into frames．There are two kinds of frames：view frame and tool window frame．Only tool windows may be placed inside tool window frames，while both editor windows and tool windows may be placed inside view frames．

## Selection

## Multiple Selection

Hex Editor Neo offers the unique feature called multiple selection．In most other editors you are only allowed to select a contiguous text or data，while Hex Editor Neo allows you to have several contiguous ranges（or blocks）in a selection．In addition，you are not limited in a number of such contiguous blocks，there can be as many blocks as you need．

| 00000060 | 7420 | 6265 | 2072 | 756 | 2069 | 6e 20 |  | 4 f | 53 | 20 | $t$ be run in dos |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 00000070 | 6d 6f | 6465 | 2e Od | Od Oa | 2400 | 0000 | 00 | 00 | 00 | 00 | mode．．．．\＄． |
| 00000080 | 36 ad | 4471 | 72 cc | 2a 22 | 72 cc | 2a 22 | 72 | cc | 2a | 22 | 6． $\mathrm{DqFrm}^{*}$＂r $\mathrm{M}^{*}$＂rM＊＊ |
| 00000090 | 7b b4 | b9 22 | 7 ccd | 2a 22 | a1 be | 2923 | 76 | cc | 2a | 22 |  |
| 000000a0 | a1 be | 2 f 23 | 51 cc | 2a 22 | 72 cc | 2622 | 96 | c5 | 2a | 22 | ys／\＃QM＊＂rM＋＂－E＂ |
| 000000b0 | al be | 2b 23 | 51 cc | 2a 22 | al be | 2723 | 38 | cd | 2a | 22 |  |
| 000000c0 | a1 be | 2e 23 | 50 cc | 2a 22 | a1 be | d7 22 | 73 | cc | 2 a | 22 |  |
| 000000d0 | al be | 2134 | 2142 | 3423 | al be | d5 22 | 73 | cc | 2a | 22 |  |
| 000000e0 | a1 be | 2823 | 73 cc | 2a 22 | 5269 | 6368 | 72 | cc | 2a | 22 | $\overline{\mathrm{y}}$（\＃SM\＃\＃＂Richrm＂${ }^{\text {c }}$ |
| 000000fo | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 00 | 00 | 00 | 00 |  |
| 00000100 | 0000 | 0000 | 0000 | 0000 | 5045 | 0000 | 64 | 86 | 08 | 00 | ．．PE．． $\mathrm{d} \dagger$ |

By default，the multiple selection feature is enabled．When you start selecting data，all previously selected data remains selected．
Most Hex Editor Neo commands work with a current selection．Among those commands are：

## Delete

Deletes the current selection．All selected ranges are removed from the document．Remaining data is shifted to＂eliminate＂gaps．File size is decreased．
Fill
Fill may be instructed to fill the current selection．All selected blocks are filled with a repeating pattern．In normal mode，filling of each block starts from the beginning of the pattern，while in＂transparent＂mode，filling of the next block continues from the end of the previous one．

## Cut，Copy，Merge and Cut，Merge and Copy

Clipboard operations always work with a current selection．

## Find，Find All，Replace，Replace All

Find and replace operations may be instructed to operate within a current selection．

This option, being enabled, makes Hex Editor Neo always drop current selection before selecting new block. You can still take advantage of the multiple selection using mouse.

## Selecting with Keyboard

When the shift key is pressed, all cursor movement results in a current selection being modified. The range between the original cursor position and resulting cursor position is added to, subtracted from or laid upon the current selection, depending on its original configuration.

If multiple cursors are present in the window, all of them start new multiple selection range.
When "Single Selection mode" option is on, you can only make a single selection with keyboard. See also Selecting with Mouse section for information how to make a multiple selection with enabled "Auto drop selection" option.

## Scripting

Keyboard selection actions may be emulated by scripts using the methods in ICursor interface.

## Selecting with Mouse

To select a data with a mouse, position the mouse cursor over the starting cell, press the left mouse button and drag the mouse to the target cell. See the table below to see the result of such operation. Pressing Ctrl or Alt key modifies the behavior of an operation. In the table, the range is data between the starting and target cell.

| Pressed Keys | "Single Selection mode" option is OFF | "Single Selection mode" option is ON |
| :--- | :--- | :--- |
| No keys pressed | The range is inverted | Current selection is dropped, range is added to selection |
| Ctrl is pressed | The range is added to current selection | The range is added to current selection |
| Alt is pressed | The range is removed from current selection | The range is removed from current selection |

## Selection Tool Window

Selection tool window is used to view and control the properties of the current selection. It consists of two parts, Information and Details.


## Information

This pane displays generic information about the current selection. The total number of bytes selected and total number of selection blocks are shown.

## Details

Selection details lists all selection blocks. For each block its ordinal number, block address and length are displayed. You can sort the list by any column (sorting is disabled if a list contains more than 500 thousands items), navigate to any block using mouse or keyboard. You can also use the list to select and delete individual block from the selection. All these commands are accessible via the shortcut menu.

## Working with Selection

The following commands are used to work with a selection:

## Select All

Selects the whole document.
Complexity: constant-time.

## Select None

Drops the current selection.
Complexity: constant-time.

## nvert Selection

Inverts the current selection. Selected blocks become unselected, and vice versa.
Complexity: constant-time.

## Select Range...

Use this command to add, subtract or overlap the given range with a current selection.
Complexity: from constant-time to linear-time, depending on the current selection.

## Select Modified

Drops the current selection and adds all modified document data to the current selection Complexity: linear-time, depending on number of document modifications.

## Save Selection..

Compresses and saves the current selection to a disk file.
Complexity: linear-time, depending on the number of blocks in a current selection.
Load Selection...
Loads and optionally merges the selection from disk file with a current one
Complexity: linear-time, depending on loaded selection complexity and current selection complexity (for merge operations).

## Copy Selection Ranges

Copy the information about the current multiple selection into the Clipboard. This includes the list of all selected ranges.

Complexity: linear-time, depending on the number of blocks in a current selection.

## Export Selection Ranges..

Save the information about the current multiple selection to the text file. This includes the list of all selected ranges
Complexity: linear-time, depending on the number of blocks in a current selection.

## Scripting

Scripts may use the following methods to work with selection objects:

- IDocumentView.saveSelectionAsync
- IDocumentView.loadSelectionAsync
- IDocumentView.copySelectionRangesAsync
- IDocumentView.exportSelectionRangesAsync

In addition, methods and properties of the ISelection interface are available to control the multiple selection object.

## Performance Considerations

Hex Editor Neo uses unique time- and space- efficient algorithm to work with a selection. Nevertheless, as a number of blocks in a selection grows, more disk space and processor time are required to work with it. This does not apply to "constant-time" selection commands, which always work in approximately constant time and do not depend on the number of blocks in a selection.

Most Hex Editor Neo commands, when instructed to work on the current selection, reveal linear-time complexity. The processing time then depends on the selection complexity, that is, the number of blocks in a selection.

Hex Editor Neo automatically starts utilizing disk space to back up complex selections. As a result, Hex Editor Neo consumes very little RAM when working with even most complex ones. Internal compression is used to minimize disk space usage.

Care must be taken while using keyboard selection with complex selections, as most keyboard selection commands operate in linear time. For example, if you have a selection with 20 million blocks and press Ctrl + Shift+End with a cursor positioned at offset 0 , then the time required to fulfill the operation will be proportional to a number of blocks in a selection. Select » Invert Selection command, on the other hand, will do the same in constant time, that is, immediately.

## Editor Commands

Pattern Window
The pattern window is used throughout the Hex Editor Neo to specify patterns to fill, insert, find, replace and so on.


The Type selector with Size selector let you specify the type of pattern you are going to enter:

| Size/Type BYTE | WORD | DWORD | QWORD |  |
| :--- | :--- | :--- | :--- | :--- |
| String <br> (Current | A pattern is specified as a <br> single-byte text string. <br> Encoding <br> Current window's encoding <br> is used. | N/A | N/A | N/A |
| UNICODE | N/A | A pattern is specified as a <br> single-byte text string. <br> Current window's encoding <br> is used. | N/A | N/A |
| String |  | 16-bit hexadecimal values | 32-bit hexadecimal values | 64-bit hexadecimal values |
|  |  | 16-bit decimal values | 32-bit decimal values | 64-bit decimal values |

All pattern types except for String and UNICODE strings are entered using standardeditor window, which is part of the pattern window. Keyboard navigation, selection and data modification operate exactly the same. A shorter version of shortcut menu is also provided for the window.

When in String or UNICODE string mode, you can select one of the previous strings by pressing the down-arrow combo-box button. For other types,Recent Patterns... button is used to access recent patterns or clear the current pattern.

Every window that contains a pattern window is resizable. You can change its size and it will be restored next time you open the same window. Each window also stores its own list of recent patterns and strings.

## Byte Order

Byte order in the pattern dialog automatically matches the one in the current editor window. Commands in the shortcut menu let you change the byte order.

## Regular Expressions

Several editor commands allow the regular expressions to be entered in "ASCII string" and "UNICODE string" modes. In this case, theRegular Expression checkbox and Sub-
match field appear. Sub-match field allows you to specify whether you want to match the entire expression (sub-match $=0$ ), or sub-expression ( sub-match $>\boldsymbol{0}$ ).
When regular expressions are used in single byte string mode, system default encoding is used

## Scripting

Patterns are represented by objects with IPattern interfaces in the scripting. Use one of the createPattern methods to create various patterns in scripts
Fill

The Fill command is used to fill the whole file or the current selection with a specified pattern.
Complexity: from constant-time to linear-time (depends on the number of blocks in a current selection).


Pattern Window is used to specify the pattern to fill. Other options are listed below:

## Fill whole file

The whole file will be filled with a pattern.

## Fill selection

Only the current selection will be filled with a pattern.

## Transparent fill

Each selection block will continue filling from the end of the previous block. If this option is off (default), each block is filled starting from the beginning of the pattern.

## Reverse pattern

The pattern is reversed before filling. Reversing is always performed on the byte level, regardless of the pattern type.
Note, that Find All command followed by Fill may be replaced with Replace All command, which will operate faster than these two commands.

## Scripting

Scripts may use the IDocumentView.fillAsync method to execute this command

## Insert

The Insert command is used to insert the specified pattern at a current location in the document. You specify the pattern and the total insert size. The pattern is then repeated until the block of given size is inserted into the document. The data beyond the cursor is shifted, effectively extending file size.

The Pattern Window is used to specify the pattern to insert. A pattern is inserted at the location of all cursors, if multiple cursors are preset in the editor window.
Complexity: constant-time


## Scripting

Scripts may use the IDocumentView.insert method to execute this command.

## Delete

Deletes the current selection. If there is no selection in an editor window, the cells under the multiple cursors are deleted.
Complexity: constant-time to linear-time (depends on the number of blocks in a current selection or the number of multiple cursors).


#### Abstract

After deleting data, remaining data is shifted to fill all gaps. The size of the file is then reduced to compensate for removed data.


## Scripting

Scripts may use the IDocumentView.clearAsync method to execute this command.

## Insert File

Inserts the contents of specified file at the current cursor position(s). The behavior of this command differs depending on the current Insert Mode.
If Insert Mode is off, the file's content overwrites the current document's content, possibly increasing file's size.
If Insert Mode is on, file is inserted at the current position. Current file's content is shifted to prepare space for the file being inserted. File size is always increased.
This command supports multiple cursors.
This command has a constant-time complexity.
Scripting

Scripts may use the IDocumentView.insertFile method to execute this command

## Insert Hex

Inserts the decoded contents of specified encoded file file at the current cursor position(s).


## Insert from

Select whether you are inserting Intel HEX or Motorola S-Records encoded data from Clipboard or from a text file. Click the Browse button to browse for the file or enter the path manually.

## Format

Select the format of your Intel HEX / Motorola S-Records encoded file (or Clipboard contents). Hex Editor Neo can sometime automatically discover the format.

## Put the data at

Select where you want to put decoded data and whether the editor should overwrite any existing data or make space for new Intel HEX or Motorola S-Records data. You may also use the Ignore data offset option to ignore original data offset specified in hex file.

## Scripting

Scripts may use the IDocumentView.insertHexAsync method to execute this command.

## Go to Offset

This command is used to move the cursor to the given position.


Enter the offset in the corresponding field. Structure Viewer's compatible expressions are supported. Use the ex prefix for hexadecimal numbers.
Specify how the resulting offset is interpreted by choosing the Absolute offset, Relative offset or Percent option.
In addition, you may select one of the bookmarks in the Bookmark list to jump to.
The resulting offset is displayed at the bottom of the window in hexadecimal and decimal formats.

## Selection

You may instruct the Hex Editor Neo to change the current selection when jumping to the specified address. You may select one of the following options:

## No change

The current selection is not changed by the operation.

## Replace selection

The current selection is discarded and new one is created. It includes all bytes from the current cursor position to the resulting position.

## Add to selection

The range from the current cursor position to the resulting position is added to the current selection.

## Remove from selection

The range from the current cursor position to the resulting position is subtracted from the current selection.

## Invert selection

The range from the current cursor position to the resulting position is inverted.

## Scripting

Various methods and properties of the ICursor interface allow scripts to emulate this command. In addition, IDocumentView.goto method may be used to navigate withing the editor window.

## Go to Offset (Process)

This command is used to jump to a given memory block within a currently edited process.


Select a memory block in the list or on the map and press the OK button.

## Change File Size

This command allows you to quickly change the size of the current document.

| 3 Change File Size |  |
| :---: | :---: |
| Specify new file size: | ок |
| 7.5 - GB |  |
| Resulting file size: | Cancel |
| $\begin{aligned} & \text { Hex: } \quad 0 \times 1 \mathrm{e} 0000000 \\ & \text { Decimal: } 8,053,063,680(7.50 \mathrm{~GB}) \end{aligned}$ | Help |
| File disk free space: 51.21 GB |  |
| You may use mathematic expression to enter new file size. See Help for more details. |  |

Complexity: constant-time.
Enter the new size (Structure Viewer's compatible expressions are supported) and select the size modifier (bytes, KB, MB or GB).
The resulting file size among with a free space on the drive from which you opened a file is displayed at the bottom of the window.
Memory or disk space is not consumed regardless of the resulting file size. Although, make sure you have enough free disk space if you are going to save the document. When the size of the file is increased as a result of this operation, a newly inserted area is filled with byte 00 .

## Scripting

Scripts may change the document's size by directly modifying the IDocument.size property.

## Encrypt

Encrypt command encrypts the current document or the current selection.
Complexity: linear-time.


## Algorithm

Provides you with a list of installed encryption algorithms. Cryptography API block and stream ciphers are supported by the Hex Editor Neo.

## Key length (bits)

Allows you to change the key length in bits for some algorithms. For others, this field is read-only.

## Password and Re-type password

Enter the encryption password here. You have to enter the same password into both fields. The password will then be required to decrypt the encrypted data.

## Do not hide password

Select to make password visible.

## Whole file/Selection

Select whether encryption takes place on the whole document or only its part.

## Scripting

Scripts may use the IDocumentView.encryptAsync method to execute this command.

## Decrypt

Decrypt command decrypts the current document or the current selection.
Complexity: linear-time.

| P Decrypt |  |  |  |
| :---: | :---: | :---: | :---: |
| Algorithm: | Advanced Encryption Standard 256-bit |  | ок |
| Key length: | $256 \div$ bits |  | Cancel |
| Password: | $\cdots$ |  | Help |
| Re-type password: | -•• |  |  |
|  | $\square$ Do not hide password | O Whole file Selection |  |
| Status: | OK |  |  |

## Algorithm

Provides you with a list of installed encryption algorithms. Cryptography API block and stream ciphers are supported by the Hex Editor Neo.

## Key length (bits)

Allows you to change the key length in bits for some algorithms. For others, this field is read-only

## Password and Re-type password

Enter the encryption password here. You have to enter the same password into both fields. The password must match the one used during encryption.
Do not hide password
Select to make password visible.

## Whole file/Selection

Select whether decryption takes place on the whole document or only its part.

## Scripting

Scripts may use the IDocumentView.decryptAsync method to execute this command.

## Modify Bits

Modify Bits command allows you to modify individual bits of the cell under the cursor.

| 01 Modify Bits $\times$ |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| 1st byte: | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | Set All To Zero |
| 2nd byte: | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | Set All To One |
| 3rd byte: | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | Setalto |
| 4th byte: | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | Invert |
|  |  |  |  |  |  |  |  |  | Revert |
| Binary representation: |  |  |  |  |  |  |  |  |  |
| 01110100011110000110010101110100 |  |  |  |  |  |  |  |  |  |
| Value: 0x74786574 |  |  |  |  |  |  |  |  | ок |
| $\bigcirc$ Decimal |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | Cancel |
| Octal |  |  |  |  |  |  |  |  | Help |

There are several checkboxes at the top of the dialog for each bit in the cell. Click on the check box to change its state. In addition, use Set All to Zero, Set All to One, Invert and Revert buttons to edit cell's bits.

Binary representation field allows you to edit bits directly. Position the cursor to the bit you want to modify and press 0 or 1 keyboard button.

## Scripting

IDocumentView.modifyBits method allows scripts to execute this command.

## Find and Replace

## Find \& Find All

Find Window
The Find Window contains a Pattern Window where you specify the search pattern.


Find Window options are described below:

## Search whole file

Search for a pattern in a whole document.
Search selection

Limit the search to the current selection. This option is disabled if no selection is currently active.

## Search up

Search backward

## Search down

Search forward

## Ignore case

Ignore case during searching.
See the Find and Find All sections for more information on pattern searching.

```
NOTE
```

Searching for regular expression backwards is not supported.

Find
The Find command is used to locate specific pattern in a file. The Find Window is used to specify the pattern to search for, as well as a number of additional options Complexity: linear-time.

## Find Operations

When the pattern is found, the cursor is moved to the beginning of a match. To continue searching, execute the Find »Find Next or Find »Find Previous commands. The direction of find depends on whether the Search up or Search down option is selected in a Find Window:

| Operation | Search down | Search up |
| :--- | :--- | :--- |
| Find Next | Forward | Backwards |
| Find Previous | Backwards | Forward |

If pattern is not found in the remaining part of the document, the message box is displayed and find is terminated.

## Regular Expressions

The Find command fully supports regular expressions. To search using regular expressions, select either "ASCII string (char[])" or "UNICODE string (wchar_t[])" pattern type, enter the regular expression, make sure the Regular expression checkbox is checked and enter the sub-expression number you want to search for. Sub-expression 0 represents the expression itself.

If the match is found, the cursor is moved to the beginning of the matched expression or sub-expression. When you use the Find Next command to find the next match, the search is started from the next cell, not the end of the matched expression.

## NOTE

Take the following limitations into account:

- Searching with regular expressions backwards is not supported.
- Searching with regular expressions within a selection (either single-range, or multiple) is not supported.


## cripting

Scripts are provided with the following methods to execute the find operations:

- IDocumentView.findAsync
- IDocumentView.findRegExpAsync
- IDocumentView.findNextAsync
- IDocumentView.findPrevAsync


## Find Al

This extremely powerful command locates all occurrences of specified pattern. The Find Window is used to specify the pattern to search for, as well as a number of additional options.

Complexity: linear time (depends on the file's size and resulting number of non-contiguous matches).

## Find All Results

The result of the Find All operation is a multiple selection, which describes all pattern occurrences. Selection Details window may be used to browse results. In addition, message box with a total number of occurrences found is displayed when searching finishes

If the pattern has not been found in a file, the message box is displayed.

## Regular Expression

The Find All command fully supports regular expressions. To search using regular expressions, select either "ASCII string (char[])" or "UNICODE string (wchar_t[])" pattern type, enter the regular expression, make sure the Regular expression checkbox is checked and enter the sub-expression number you want to search for. Sub-expression 0 represents the expression itself.

NOTE
Searching with regular expressions within a selection (either single-range, or multiple) is not supported.

## Scripting

Scripts are provided with the following methods to execute the find operations:

- IDocumentView.findAllAsync
- IDocumentView.findAllRegExpAsync


## Replace \& Replace AI

## Find \& Replace Window

The Find \& Replace Window contains two Pattern Windows where you specify the pattern to search and the pattern to replace.


Find \& Replace Window options are described below:

## Search whole file

Search for a pattern in a whole document.

## Search selection

Limit the search to the current selection. This option is disabled if no selection is currently active.

## Search up

Search backwards

## Search down

Search forward

## Ignore case

Ignore case during searching.
See the Replace and Replace All sections for more information on pattern searching and replacing.

## Replace

The Replace command is used to locate a specific pattern in a file and replace it with another pattern. The Find \& Replace Window is used to specify both patterns, as well as a number of additional options. It is not required that the size of the search and replace patterns match, in addition, the replace pattern may be empty. In the latter case, found pattern occurrences are removed from the document.

Complexity: linear-time.

## Find \& Replace Operations

When you press the Replace button for the first time, Hex Editor Neo starts searching for a pattern. When pattern is located, thecursor is moved to the beginning of the pattern and a Find \& Replace Window is displayed again. Now you have a choice of pressing the Find Next button to skip this match and search for another, or pressing the Replace button to replace this match and continue searching.

The direction of the next search is governed by the Search up or Search down option.
When there are no more matches, the message box is displayed.

## Regular Expressions

The Replace command fully supports regular expressions. To search using regular expressions, select either "ANSI string" or "UNICODE String" pattern type, enter the regular expression, make sure the Regular expression checkbox is checked and enter the sub-expression number you want to search for. Sub-expression $\theta$ represents the expression itself.

If the match is found, the cursor is moved to the beginning of the matched expression or sub-expression. When you use the Find Next command to find the next match, the search is started from the next cell, not the end of the matched expression. If you press the Replace button, the matched expression or sub-expression is replaced with a replace pattern.

A replace pattern may contain special characters. To tell Hex Editor Neo to treat a replace pattern specially, make sure its Regular expression checkbox is checked. If it is, the following restrictions apply:

1. The type of the replace pattern automatically matches the type of the search pattern (encoded or UNICODE string).
2. You are restricted to use sub-expression 0 for a search pattern (that is, the whole expression).

If you enable a regular expression mode for a replace pattern, but do not use any of the special characters, Hex Editor Neo will automatically turn this mode off when performing replace in order to achieve greater performance.

See the Replace Pattern Syntax topic for further information on supported syntax.

## NOTE

Take the following limitations into account:

- Searching with regular expressions backwards is not supported.
- Searching with regular expressions within a selection (either single-range, or multiple) is not supported.

Scripts may use the IDocumentView.replaceAsync and IDocumentView.replaceRegExpAsync methods to execute this operation.
Replace All
This extremely powerful command can be used to locate all occurrences of one pattern and replace them with another pattern. The Find \& Replace Window is used to specify both patterns, as well as a number of additional options. It is not required that the size of the search and replace patterns matches, in addition, the replace pattern may be empty. In the latter case, found pattern occurrences are removed from the document.

Complexity: linear-time (depends on the file's size and resulting number of noncontiguous matches).

## Replace All Results

After all occurrences of the search pattern are found and replaced, the message box with a total number of replacements is displayed. If there were no matches, the message box with "No matches" text is displayed.

The Replace All command is always faster than a series of commands, such as Find All and Fill, or Find All and Delete. The latter command sequence is easily replaced with a Replace All command with an empty replace pattern.

Replace All command, as most other commands, creates an operation in the operation history.

## Regular Expressions

The Replace All command fully supports regular expressions. To search using regular expressions, select either "ASCII string (char[])" or "UNICODE string (wchar_t[])" pattern type, enter the regular expression, make sure the Regular expression checkbox is checked and enter the sub-expression number you want to search for. Sub-expression 0 represents the expression itself.

A replace pattern may contain special characters. To tell Hex Editor Neo to treat a replace pattern specially, make sure its Regular expression checkbox is checked. If it is, the following restrictions apply:

1. The type of the replace pattern automatically matches the type of the search pattern (encoded or UNICODE string).
2. You are restricted to use sub-expression $\theta$ for a search pattern (that is, the whole expression).

If you enable a regular expression mode for a replace pattern, but do not use any of the special characters, Hex Editor Neo will automatically turn this mode off when performing replace in order to achieve greater performance.

See the Replace Pattern Syntax topic for further information on supported syntax.

NOTE
Searching with regular expressions within a selection (either single-range, or multiple) is not supported.

## Scripting

Scripts may use the IDocumentView.replaceAllAsync and IDocumentView.replaceAllRegExpAsync methods to execute this operation.

## Find in Files \& Replace in Files

Find in Files and Replace in Files are the two powerful Hex Editor Neo commands that allow you to perform batch find and replace operations. They can be instructed to operate on all files in a folder or the whole folder tree.

The Find in Files Window is used to enter the find and/or replace patterns and specify the list of folders to search in. You can also narrow the search by entering the mask (for example, *.txt) to search only in files that match the mask. Several masks may be specified (separate them with a semicolon). Below is an example that can be used to search within $\mathrm{C} / \mathrm{C}++$ source files:
*.c;*.cpp;*.cxx;*.cc;*.h;*.hpp;*.hxx;*.idl
All matched documents are displayed in the Find in Files Tool Window. In some operation modes, matched files are immediately opened in the editor, while in others, you may open them by double-clicking on the file's item in a result list.

## Scalability

The implementation of Find in Files and Replace in Files commands scales well, that is, it works faster on multi-core or multi-processor computers. Several dedicated threads of execution are launched on such computers and perform searching and replacing in parallel.

Note that the real performance boost is achieved only if you have a fast disk system or access files over a fast network connection.

## Safety

Replace in Files function, operating in Replace all occurrences and save mode may potentially be harmful to your data. Hex Editor Neo automatically detects if you are trying to execute this operation and have Always create backups Option turned off. It then warns you about the possible data loses and offers several actions to continue:

- Hex Editor Neo offers you to enable the "Always create backups" option and continue the operation.
- It also offers you to switch to safer Replace all occurrences and open mode, in which all modifications are stored in memory and you decide when and whether to save them.
- It then offers you to continue the operation, if you understand its consequences.
- And finally, it offers you to cancel operation.

If you have "Always create backup" option turned on, then each modified file is first backed up before modification.

## Regular Expressions

Both Find in Files and Replace in Files functions fully support regular expressions.
Find in Files Window

Find in Files Window is used to specify the search pattern, replace pattern (optional) and other options required by Find in Files and Replace in Files commands.


Window options are described below:

## Find and replace mode

Check to enable the Replace in Files mode, uncheck to enable Find in Files mode.
Search in
Enter the list of folders (separated with semicolon) to search in or press the browse (...) button to display the Folder List Window, where you can manage the list.
File types
The list of file masks to search for. Each mask is separated with a semicolon.

## Include sub-folders

Sub-folders of each folder in a folder list are automatically included into the folder list.

## lgnore case

Case is ignored during pattern searching.

## Display errors

File opening or saving errors are displayed in the output.

## Include streams

NTFS alternate data streams are included in the searching/replacing. Please note that this option includes all streams of any matched file in a search, there is no way to filter streams, only files

## Mode

Operation mode. Two operation modes are available for Find in Files command and two operation modes are available for Replace in Files command:

| Command | Mode | Description |
| :--- | :--- | :--- |
| Find in Files | Find at least one <br> occurrence | Each file that matches specified mask(s) is searched for a pattern. Once the pattern is found, its <br> offset is recorded and file is included in the result list. Find is aborted for this file. |
|  | Find all occurrences <br> Each file that matches the mask(s) is searched for a pattern. The file that contains at least one <br> pattern occurrence is opened in the Hex Editor Neo and all pattern occurrences are added to a <br> multiple selection. |  |
| Replace in |  |  |
| Files | Replace all occurrences <br> and save | Each file that matches specified mask(s) is searched for a pattern. Each pattern occurrence is <br> replaced by the replace pattern. The file is then saved and closed. |
| Replace all occurrences |  |  |
| and open |  |  | | Each file that matches specified mask(s) is searched for a pattern. Each pattern occurrence is |
| :--- |
| replaced by the replace pattern. The file is then opened in the editor. You can then review the |
| changes and either cancel them or commit (by saving the document). |

## Folder List Window

Folder List Window is used to manage the folder list for Find in Files and Replace in Files functions

| $\square$ Find in Files Folder List |  | $\times$ |
| :---: | :---: | :---: |
| Enter the path to the folder or press the browse button: |  |  |
|  | ... | Add |
| Folder list: |  |  |
| c:\|windows\} |  | ок |
|  |  | Cancel |
|  |  | Move Up |
|  |  | Move Down |
|  |  | Remove |
|  |  | Remove All |
| $\square$ |  | Help |

To remove a folder or folders from a list, select them with mouse or keyboard and press the Remove button. To add new folder, enter its full path or press the Browse (...) button. Then press the Add button.

To clear the folder list, press the Remove All button
To change folders order, select one folder and press Move Up or Move Down button.

## Find in Files Tool Window

This tool window is used to display the Find in Files and Replace in Files function results. During the function execution, all matched files are listed in the window.


Depending on the operation mode, slightly different information is displayed.

## Find at least one occurrence

Each matched file's full path is displayed. To open a file and navigate to located pattern, double click it or select it and press the Open File button on the toolbar.

## Find all occurrences

Each matched file's full path and number of pattern occurrences are displayed. A file is opened in the editor. Open File command is disabled (as the file is already opened).

## Replace all occurrences and save

Each file where any replacements are made is added to the result list. The full file's path along with a number of replacements is displayed. Double-clicking on the file opens it in the editor.

## Replace all occurrences and open

Each file where any replacements are made is added to the result list and opened in the editor. The full file's path along with a number of replacements is displayed. The Tools» Find in Files» Open File command is disabled.

During operation, the last item in a list displays an operation progress. The following information is displayed:
p \% completed. N of T files in F folders have been checked. $M$ matched file(s) found. S of TS processed.
Where
p
A percentage of the operation completed. It is always zero until the total number of files that match criteria (folder list and mask) is determined.
N
A number of files processed so far.
T
A total number of files to process. This is an amount of all files that match the mask and folder list filters. If Hex Editor Neo is still scanning folders, the "(...)" is added after the current total number of files. The percentage $p$ is always zero in this case. Folder scanning and pattern searching/replacing are always performed concurrently.
F
A total number of folders to process.
M
A total number of matched files.
s
A number of bytes processed.
TS
A total number of bytes to process.
To clear all items in a list, execute the Tools» Find in Files » Clear All command.
To cancel the current operation, execute the Tools »Find in Files » Stop Searching command.

## Find in Files

The Find in Files command is used to search a list of files for a specified pattern. You specify the list of folders and file mask that determine the list of files and you enter a pattern to search. The Find in Files Window is used to provide all this information.


Find in Files function operates in one of two modes: Find at least one occurrence and Find all occurrences.

## Find at Least One Occurrence

In this mode, all candidate files are searched for a first pattern occurrence. If it is found, the file is added to a result list and searching continues with a next file.
Double-clicking the file in a result list opens the file and navigates to the located position.

In this mode, all occurrences of the pattern are located. If there are any, the file is opened in the editor and selection displays all matched locations. The file's path is also added to the result list.

## Regular Expressions

The Find in Files command fully supports regular expressions. To search using regular expressions, select either "ASCII string (char[])" or "UNICODE string (wchar_t[])" pattern type, enter the regular expression, make sure the Regular expression checkbox is checked and enter the sub-expression number you want to search for. Sub-expression $\theta$ represents the expression itself.

## Replace in Files

The Replace in Files command is used to search a list of files for a specified pattern and replace it with another pattern. You specify the list of folders and file mask that determine the list of files and you enter both patterns to search in a Find in Files Window. The "Find and replace mode" switch must be enabled for this command


Replace in Files function operates in one of two modes: Replace all occurrences and save and Replace all occurrences and open.

## Replace All Occurrences and Save

In this mode, all candidate files are searched for a search pattern. All occurrences are replaced with a replace pattern. If the replace pattern is empty, all matches are removed from the file. Any matched file is added to the results list. All modifications made to a file are immediately saved. The Always create backups Option is used to determine whether to make backup copies of modified files.

Double-clicking the file in a result list opens the file.
Replace All Occurrences and Open

In this mode, all candidate files are searched for a search pattern. All found occurrences are replaced with a replace pattern. Files then are opened in the editor. Operation history for an opened file contains a Replace All command. You can then continue editing the document, save changes, or close the document without saving changes.

## Regular Expressions

The Replace in Files command fully supports regular expressions. To search using regular expressions, select either "ASCII string (char[])" or "UNICODE string (wchar_t[])" pattern type, enter the regular expression, make sure the Regular expression checkbox is checked and enter the sub-expression number you want to search for. Sub-expression $\theta$ represents the expression itself. Replace pattern cannot be a regular expression.

## Regular Expressions

Regular expressions provide a concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters. Regular expressions are written in a formal language. The syntax used by the Hex Editor Neo is essentially the syntax standardized byECMAScript, with minor changes in support of internationalization. The following section briefly describes the syntax.

Regular expressions are supported by the following Hex Editor Neo commands and modules:

- Find
- Find All
- Replace
- Replace All
- Find in Files
- Replace in Files
- Pattern Coloring
- Pattern Statistics


## Capturing Sub-expressions

Every Hex Editor Neo function that works with regular expressions allows you to specify what sub-expression you want to capture. Sub-expression zero represents the entire expression, sub-expression 1 and greater represent corresponding sub-expressions.

If you specify the sub-expression that is greater than the total number of sub-expressions, the error message is displayed.

## Usage Tips and Performance Considerations

Using regular expressions affect performance and memory usage. Always prefer using normal pattern searching functions whenever possible. Replace All command with regular expressions may need as much as 3 times more memory compared to a simple (pattern based) Replace All command.

The following limitations are present in the current version of the Hex Editor Neo:

- Searching for a regular expression backwards is not supported
- Searching within selection (either single or multiple) is not supported.
- Pattern Coloring will sometimes fail to highlight a complex match if it starts long before the visible area and/or ends long after the visible area.
- Zero-matching regular expressions are forbidden

Future versions of the Hex Editor Neo will eliminate some of these limitations, while others are by design and cannot be eliminated.

## Regular Expressions Syntax

A regular expression is a pattern of text that consists of ordinary characters (for example, letters a through z) and special characters, known as metacharacters. The pattern describes one or more strings to match when searching text. The following table contains the complete list of metacharacters and their behavior in the context of regular expressions (regular expressions syntax):

| 1 | Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, n matches the character n . In matches a newline character. The sequence \I matches \and |
| :---: | :---: |
| ( matches (. |  |
| $\wedge$ | Matches the position at the beginning of the input string. Also matches the position following $\backslash \mathrm{n}$ or $\backslash \mathrm{r}$. |
| \$ | Matches the position at the end of the input string. Also matches the position preceding ln or $\backslash \mathrm{r}$. |
| * | Matches the preceding character or sub-expression zero or more times. For example, zo* matches $z$ and $z o o . *$ is equivalent to $\{0$,$\} .$ |
| + | Matches the preceding character or sub-expression one or more times. For example, zo+ matches zo and zoo, but not $z$. + is equivalent to $\{1$,$\} .$ |
| ? | Matches the preceding character or sub-expression zero or one time. For example, do(es)? matches the do in do or does. ? is equivalent to $\{0,1\}$. |
| \{n\} | n is a non-negative integer. Matches exactly n times. For example, $o\{2\}$ does not match the $o$ in Bob, but matches the two o's in food. |
| \{n, \} | n is a non-negative integer. Matches at least n times. For example, $o\{2$,$\} does not match the o$ in Bob and matches all the o's in foooood. o $\{1$,$\} is equivalent to 0+$. $o\{0$,$\} is equivalent to \circ^{*}$. |
| $\{\mathrm{n}, \mathrm{m}\}$ | $m$ and $n$ are non-negative integers, where $n<=m$. Matches at least $n$ and at most $m$ times. For example, $o\{1,3\}$ matches the first three 0 's in fooooood. $0\{0,1\}$ is equivalent to 0 ?. Note that you cannot put a space between the comma and the numbers. |
| ? | When this character immediately follows any of the other quantifiers $(*,+, ?,\{n\},\{n\},,\{n, m\})$, the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string $0000,0+$ ? matches a single 0 , while o+ matches all os. |
| . | Matches any single character. |
| (pattern) | A sub-expression that matches pattern and captures the match. To match parentheses characters (), use $\backslash$ ( or $\backslash$ ) . |
| (?:pattern) | A sub-expression that matches pattern but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character \|. For example, industr(?:y|ies) is a more economical expression than industry|industries. |
| (? p attern) | A sub-expression that performs a positive lookahead search, which matches the string at any point where a string matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example Windows (? $=95\|98\| \mathrm{NT} \mid 2000$ ) matches Windows in Windows 2000 but not Windows in Windows 3.1. Look-aheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead. |
| (?!pattern) | A sub-expression that performs a negative lookahead search, which matches the search string at any point where a string not matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example Windows (?!95\|98|NT|2000) matches Windows in Windows 3.1 but does not match Windows in Windows 2000. Look-aheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead. |
| (?>pattern) | Independent sub-expression, match pattern and turn off backtracking. |
| (? $<=$ pattern) | Positive look-behind assertion, match if after pattern but don't include pattern in the match. (pattern must be constantwidth). |
| (? 3 ! pattern) | Negative look-behind assertion, match if not after pattern. (pattern must be constant-width). |
| (?i:pattern) | Match pattern disregarding case. Allows to temporary turn off case sensitiveness during pattern matching. |
| (?\$name) | Reference a named regular expression class. A class must be defined in the Regular Expressions Settings, otherwise, the error will be generated. |
| $x \mid y$ | Matches either x or y . For example, $\mathrm{z} \mid$ food matches z or food. (z\|f)ood matches zood or food. |
| [xyz] | A character set. Matches any one of the enclosed characters. For example, [abc] matches the a in plain. |
| [^xyz] | A negative character set. Matches any character not enclosed. For example, [^abc] matches the p in plain. |
| [a-z] | A range of characters. Matches any character in the specified range. For example, [a-z] matches any lowercase alphabetic character in the range a through $z$. |
| [^a-z] | A negative range characters. Matches any character not in the specified range. For example, [^a-z] matches any character not in the range a through z . |
| 1 b | Matches a word boundary, that is, the position between a word and a space. For example, er $\backslash \mathrm{b}$ matches the er in never but not the er in verb. |
| \B | Matches a non-word boundary. er $\backslash \mathrm{B}$ matches the er in verb but not the er in never. |
| lcx | Matches the control character indicated by x . For example, \cm matches a Control- M or carriage return character. The value of $x$ must be in the range of $A-Z$ or $a-z$. If not, $c$ is assumed to be a literal $c$ character. |
| ld | Matches a digit character. Equivalent to [0-9]. |
| ID | Matches a non-digit character. Equivalent to [^^-9]. |
| If | Matches a form-feed character. Equivalent to \xec and \cL. |
| In | Matches a newline character. Equivalent to \x@a and \cJ. |
| Ir | Matches a carriage return character. Equivalent to \xөd and \cm. |
| Is | Matches any white space character including space, tab, form-feed, and so on. Equivalent to [ $\backslash \mathrm{f} \backslash \mathrm{n} \backslash \mathrm{r} \backslash \mathrm{t} \backslash \mathrm{v}$ ]. |
| Is | Matches any non-white space character. Equivalent to [^ \f\n\r\t\v]. |
| It | Matches a tab character. Equivalent to \xөe and \cI. |
| Iv | Matches a vertical tab character. Equivalent to $\backslash \times 0 \mathrm{~b}$ and $\backslash \mathrm{ck}$. |
| Iw | Matches any word character including underscore. Equivalent to [A-Za-ze-9_]. |
| Iw | Matches any non-word character. Equivalent to [^A-Za-z0-9_] . |
| Ixn | Matches $n$, where $n$ is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, \x41 matches A. \x041 is equivalent to $\backslash \times 04 \& 1$. Allows ASCII codes to be used in regular expressions. |
| \num | Matches num, where num is a positive integer. A reference back to captured matches. For example, (.) 11 matches two consecutive identical characters. |


| In | Identifies either an octal escape value or a back-reference. If n is preceded by at least n captured sub-expressions, n is a back-reference. Otherwise, n is an octal escape value if n is an octal digit ( $0-7$ ). |
| :---: | :---: |
| \nm | Identifies either an octal escape value or a back-reference. If $\backslash \mathrm{nm}$ is preceded by at least nm captured sub-expressions, nm is a back-reference. If $\backslash \mathrm{nm}$ is preceded by at least n captures, n is a back-reference followed by literal m . If neither of the preceding conditions exists, nm matches octal escape value nm when n and m are octal digits (0-7). |
| \nml | Matches octal escape value nml when n is an octal digit ( $0-3$ ) and m and 1 are octal digits ( $0-7$ ). |
| \un | Matches $n$, where $n$ is a Unicode character expressed as four hexadecimal digits. For example, \u00A9 matches the copyright symbol (©). |

## Replace Pattern Syntax

Hex Editor Neo allows you to use special characters in replace pattern when performing commands Replace and Replace All in regular expressions mode.
The main advantage of using regular expressions during pattern matching operations is the ability to match different patterns with a single expression. This function is natively cooperated with an ability to refer to matched pattern in replace pattern. Hex Editor Neo allows you to perform such referrals.
All characters in a replace pattern are treated "as is", except for the following character combinations:

| \$\& | Refers to the whole matched expression. |
| :---: | :---: |
| \$n | Where n is [1..9]: Refers to numbered sub-expression of matched expression. |
| \$\$ | Means a single \$ character. |
| \xnn | Where n is [0..9] or [a..f] or [A..F]: A single hexadecimal code byte. Can be used only if performing search/replace in single-byte string mode (ASCII or encoded). |
| \xnnnn | Where n is $[0 . .9$ or $[\mathrm{a} . . \mathrm{f}]$ or [A..F]: A single hexadecimal code word. Can be used only if performing search/replace in UNICODE mode. |

## Examples

This section lists a few regular expressions examples. The description section describes the results of the Find All command used with a given regular expression. \w+

Match all identifiers in a programming language source file.
(.) $\backslash 1+$

Match any sequence of repeated characters, which length is at least two characters.
[ $\times \times 00 \backslash \times 01 \backslash \times 02$ ]
Match 00, 01 or 02 bytes in the document.
\w+(?=\s*\*)
Match all C++ class names in the document which are followed by a * character. * character and any optional space characters before it are not matched.
(?<=http://)[\w|.]+?\.(com|net|org)
Match all domain name portions of URLs.
$\backslash d\{1,3\} \backslash . \backslash d\{1,3\} \backslash . \backslash d\{1,3\} \backslash . \ d\{1,3\} \backslash$
Match all IPv4 addresses
$\wedge$
Match "\{" characters at the beginning of the line.

## Named Classes

Hex Editor Neo allows you to define any number of named regular expression classes. Each class is a regular expression with a name assigned to it. A defined class may later be referenced in the regular expression (and in another named class) using the following syntax:

## (?\$name)

## Recursive Grammars

The concept of named classes is so powerful, that it even allows you to define complex recursive grammars. This documentation provides one example of how this feature may be used. See the Expression Demo section for more information.

## Creating and Managing Classes

The Regular Expressions Settings is used to create new named classes and edit existing ones.

| 앙 Settings: Regular Expressions |  | $\times$ |
| :---: | :---: | :---: |
| Search settings | You can create reusable regular expression classes on this page. Every class should be a valid regular expression and may reference other classes. |  |
| Visual Themes | Class Name Regular Expression | Add... |
| User Interface |  | Edit. |
| Keyboard |  | Delete |
| Commands |  |  |
| Network \& Support |  | Delete All |
| Beta |  |  |
| Crash Reporting Proxy |  |  |
| Binary EditorRegular ExpressionsTasksEditorData Annotations |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Structure Viewer General Structure Editor |  |  |
|  |  |  |
|  |  |  |
| Other General |  |  |
| Settings | Global regular expression options:'.' (Dot) metacharacter does not match null character (' 10 ')' $\because$ ' (Dot) metacharacter does not match newline character (' $\mid n$ ')Non-escaped white-space is not significant |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | OK Close | Help |

## Add Button

Press the button to create a new named class. A class must have the unique name

## Edit Button

Press the button to edit a currently selected class. You can change a class name and a regular expression. If you change a class name, a new name must be unique

## Delete Button

Press the button to delete a currently selected class.

## Delete All Button

Press the button to delete all classes.
If you accidentally made a mistake, you may press the Cancel button to cancel all changes. The Cancel button cancels the changes even if the Apply button has been pressed.

## Regular Expression Syntax Errors

Syntax errors in entered regular expressions are not checked until the expression is actually used in one of the find/replace operations.

## Example

This section shows you how the named regular expression classes may be used, for example, to match standard mathematical expressions, like $2+3 *(7-9)$ or $2+3 * 8-11$. First, we need to define a number of regular expression classes:
group
$\backslash((? \$$ expr $) \backslash)$
factor
\d +1 (?\$group)
term
(?\$factor)( \* (?\$factor)| / (?\$factor))*
(?\$term) ( \+ (?\$term)| - (?\$term))*

Now, if you execute the Find All command in regular expression mode and enter the
(?\$expr)
as regular expression to search for, all mathematical expressions in the document get selected.

## Clipboard

## Supported Formats

Hex Editor Neo supports two clipboard formats: internal format and Windows Explorer format
Hex Editor Neo uses the internal format to exchange data between its documents and uses the Windows Explorer format to exchange files and data with Windows Explorer. When you put a selection into the Clipboard, it is automatically placed in both supported formats.

## Internal Format

This format allows you to exchange data between two documents in the Hex Editor Neo as well as between several locations within the same document. Data placed to the Clipboard in one running Hex Editor Neo instance cannot be pasted into another. To exchange data between two documents, open them both in a single Hex Editor Neo instance.

Internal format fully supports multiple selection.

## Windows Explorer Format

Hex Editor Neo supports exchanging data with Windows Explorer.

## Hex Editor Neo's Document Data to File

You can paste the copied document's data into the file in Windows Explorer. Select the data you want to copy (multiple selection is fully supported) and put it into the Clipboard (for example, execute the Edit » Copy command). Open a folder in Windows Explorer where you want the data to be pasted. Execute the Paste command. Copied data will be pasted into the new file, named "HHD Hex Editor Neo 7 Copied Data".

## File Data to Hex Editor Neo's Document

You can also paste a file copied to the Clipboard from Windows Explorer (or any other compatible application) into the current cursor positions in a document (multiple cursors are supported for paste operation).

## Text

You can paste a text copied from some other application into the Hex Editor Neo. See the Pasting Text section for detailed feature description. Copy \& Export Tool Window, on the other hand, provides a number of tools to encode binary document data and place it into the Clipboard in textual form.

## Scripting

Scripts may use the following methods to work with the Clipboard

- IDocumentView.cutAsync
- IDocumentView.copyAsync
- IDocumentView.pasteAsync
- IDocumentView.cutAsync


## Copying, Cutting and Merge Modifier

Several commands exist in the Hex Editor Neo that can be used to put data into the Clipboard. They always operate on the currentselection unless it is empty. In the latter case, all Clipboard operations work on the cell under the cursor. Hex Editor Neo's Clipboard operations fully support multiple selection.

The behavior of Clipboard commands differ:

## Copy Command

The selected data is placed into the Clipboard.
Complexity: from constant-time to linear-time (depends on the number of selection blocks).

## Cut Command

The selected data is placed into the Clipboard and then removed from the document, as if you executed the Delete command.
Complexity: from constant-time to linear-time (depends on the number of selection blocks). The performance of the Cut operation is better than of the Copy operation followed by Delete operation.

## Merge and Copy

All selection blocks are merged together and then the resulting contiguous block is placed into the Clipboard.
Complexity: from constant-time to linear-time (depends on the number of selection blocks).

## Merge and Cut

All selection blocks are merged together and then the resulting contiguous block is placed into the Clipboard. The selected data is then removed from the document, as if you executed the Delete command.
Complexity: from constant-time to linear-time (depends on the number of selection blocks). The performance of the Cut operation is better than of the Copy operation followed by Delete operation.

## mportant Considerations

The file whose data is put into the Clipboard is referenced until something else is placed into the Clipboard (by any application), running instance of the Hex Editor Neo is closed, or referenced document is saved. In all these cases, data is silently removed from the Clipboard.

While the file is referenced, it cannot be deleted or modified outside the Hex Editor Neo.
Third-party Clipboard extension tools may prevent the document from being unreferenced at the proper time leading to an inability to modify the file outside the Hex Editor Neo until you close it.

## Advanced Copy \& Export

Hex Editor Neo has an Advanced Copy \& Export module, which may be used to convert binary data into text, using one of a large number of supported formats. Starting from version 4.92, Hex Editor Neo also automatically puts data into Clipboard in one of those formats (depending on the currenteditor window's settings) when you execute the Edit » Copy or Edit » Cut commands.

## Limitations

The current version of the Hex Editor Neo does not support creating patches if the document contains data pasted from other documents. On the other hand, if the document contains only data pasted from itself, the patch can be created.

## Scripting

Scripts may use the following methods:

- IDocumentView.cutAsync
- IDocumentView.copyAsync


## Pasting Data

This section describes pasting file (or binary) data into Hex Editor Neo documents. To learn how to paste text data, see the Pasting Text section.
You can paste data into the Hex Editor Neo if it is in one of the supported formats. To paste data from the Clipboard, you need to have an editor window active. Position the cursor(s) to the offset where you want the data to be placed and execute the Edit » Paste command. Multiple Cursors are supported by this command.

The current Insert Mode governs the behavior of the Paste command. If it is on, file's data beyond the cursor position is shifted to free required space, in which Clipboard data is placed. If Insert Mode is off, file's data is overwritten with data from the Clipboard.

If the multiple selection existed at a time you executed the Copy or Cut commands, additional gaps are inserted into the document. If Insert Mode is on, those gaps are cumulative.

## Paste Command Complexity

If Clipboard contains a file copied from the Windows Explorer, the paste command operates in constant-time, that is, immediately. If data has been copied or cut from the same or another Hex Editor Neo's document, the complexity is linear-time, depending on the number of blocks in the selection.

## Scripting

Scripts may use the IDocumentView.pasteAsync method to execute this operation

## Pasting Text

Hex Editor Neo supports pasting text from Clipboard. This section describes supported formats and usage strategies:

## Pasting Raw Tex

Any plain text placed into the Clipboard may be pasted into the Hex Editor Neo's document. In order to paste the text, perform the following steps:

1. Open (or create) the document you want to paste text to
2. Make sure the required encoding is selected.
3. Navigate to the position where you want to paste text.
4. Check the Insert Mode switch. Turn it OFF to overwrite existing document's data with Clipboard's data, or ON to make room for Clipboard's data
5. Make sure the text pane is visible and active (the cursor image has a frame around it),
6. Execute the Edit» Paste command.

## Pasting Formatted Numbers

In addition to providing the ability to paste the text "as it is", Hex Editor Neo also supports parsing Clipboard's text in order to extract a sequence of decimal or hexadecimal numbers.

Make sure the code pane is active before pasting data. If the text pane is active, Clipboard contents will be pasted "as is", as described in the previous section. As always, the Insert Mode switch is used to specify whether you want to overwrite or insert data.

The current view type dictates the type of integers parsed from the input text. If current view type is a hexadecimal view type, integers are assumed to be in base 16; otherwise, they are assumed to be in base 10 . Use the " $0 x$ " prefix to temporary change number's base to 16 (it affects only the number immediately following the prefix).

Numbers may be separated with one or more separators. Hex Editor Neo treats all non-digit (or non-hexadecimal digit) characters as separators, including line feed and carriage return characters. If there are no separators, current editor window's grouping mode is used to separate them.

Multiple separators are allowed. Parsing continues to the end of the input text.
Pasting Formatted Numbers feature is not supported for Float and Double view types.

## Examples

The following text pasted into the editor window, currently in Hex Bytes view type...
$1234 \mathrm{ax5}$
a 7 f 12, ff; 12
results in the following sequence (displayed in hex) to be inserted into the document:
$0102034 a 050 a 7 f 12$ ff 12
The same text pasted into the Decimal Bytes window results in the following sequence (displayed in decimal):
1234571212
Parsing ignored all invalid characters, treating them as separators. "0x" prefix may be used to temporary switch to hexadecimal mode:
0x34 23 0x5a 0xaa 17
Which results in the following sequence (displayed in decimal)
$52 \quad 23 \quad 90170 \quad 17$

The following text can be pasted into the editor window, which is currently in Hex Words mode:
7a20, 5, ffff, AE01
12c 150017 FF

## Scripting

Scripts may use the IDocumentView.pasteAsync method to execute this operation

## Advanced Copy \& Export

n addition to standard Clipboard support, Hex Editor Neo provides advanced copy/export feature.
This feature allows you to convert selected data (multiple selection is fully supported) into one of supported format and either place it into the Clipboard or write to a file
All currently supported formats generate textual data that can directly be pasted into or opened by nearly every application. Three formats, Raw Text, Formatted Data and Encoded Data are provided, each of which offer a rich set of configurable options.

## Copy \& Export Tool Window

This tool window is used to configure the feature and start data conversion.


A window is divided into two parts: at the top, two panes, Copy to Clipboard and Export are displayed. Only one pane may be opened at a given time. The active pane determines the action to be held by the Hex Editor Neo: if the Copy to Clipboard pane is opened, data will be placed into the Clipboard, if Export pane is opened, data will be exported to a given file.

Copy to Clipboard pane contains a single configurable option: a Merge switch. It becomes active when there is a multiple selection and allows you to merge all selected blocks during data conversion.

Export pane allows you to specify the full path to the file you want the data to be written to. The Append switch allows you to append new data to an existing file and Merge switch, as already described, allows you to merge all selected blocks during data conversion.

Second part of this tool window contains one pane for each supported data format. Only one pane may be opened at a given time. Opened pane determines the format to be used for data conversion. Subsequent sections describe each format in greater detail.

## Raw Text Format

This data format is used to copy the contents of the text pane of the editor window, "as you see it". In fact, selected document's data is interpreted as textual data and placed into the Clipboard (or exported) in the given encoding.

The format has a single configurable parameter, Encoding, which can be used to tell Hex Editor Neo how to export document's data.
NOTE: Support for a specific encoding depends on installed Windows code pages and fonts. If required components cannot be found for a selected encoding, resulting output will be empty.

The following table lists all "special" encodings:

| Encoding | Notes |
| :---: | :---: |
| None | Document's data is placed into the Clipboard (or file) "as is". No processing is performed. Selection is assumed to contain ANSI text. |
| Current | The current editor window's encoding is used to convert data. This option is the default. |
| System default | Selected data is assumed to contain ANSI text in current system encoding. It is converted to UTF-16. |
| OEM | Selected data is assumed to contain ANSI text in current system OEM encoding. It is converted to UTF-16. |
| UTF-7 | Selected data is assumed to contain multi-byte UNICODE text encoded in UTF-7 encoding. It is converted to UTF-16. |
| UTF-8 | Selected data is assumed to contain multi-byte UNICODE text encoded in UTF-8 encoding. It is converted to UTF-16. |
| Unicode (UTF16) | Document's data is placed into the Clipboard (or file) "as is". No processing is performed. Selection is assumed to contain UNICODE UTF-16 text. |

When selected data is not too complex, it is automatically put into the Clipboard in this format when user executes the Edit » Copy or Edit» Cut and cursor is in the text pane. All following encodings assume the text to be a single-byte or multi-byte DBCS text and convert it to UTF-16.

[^0]- IBM EBCDIC - Arabic
- IBM EBCDIC - Cyrillic (Russian)
- IBM EBCDIC - Cyrillic (Serbian, Bulgarian)
- IBM EBCDIC - Denmark/Norway (20277 + Euro symbol)
- IBM EBCDIC - Denmark/Norway
- IBM EBCDIC - Finland/Sweden (20278 + Euro symbol)
- IBM EBCDIC - Finland/Sweden
- IBM EBCDIC - France (20297 + Euro symbol)
- IBM EBCDIC - France
- IBM EBCDIC - Germany (20273 + Euro symbol)
- IBM EBCDIC - Germany
- IBM EBCDIC - Greek
- IBM EBCDIC - Hebrew
- IBM EBCDIC - Icelandic (20871 + Euro symbol)
- IBM EBCDIC - Icelandic
- IBM EBCDIC - International (500 + Euro symbol)
- IBM EBCDIC - International
- IBM EBCDIC - Italy (20280 + Euro symbol)
- IBM EBCDIC - Italy
- IBM EBCDIC - Japanese Katakana Extended
- IBM EBCDIC - Korean Extended
- IBM EBCDIC - Latin 1/Open System
- IBM EBCDIC - Latin America/Spain (20284 + Euro symbol
- IBM EBCDIC - Latin America/Spain
- IBM EBCDIC - Latin-1/Open System (1047 + Euro symbol)
- IBM EBCDIC - Modern Greek
- IBM EBCDIC - Multilingual/ROECE (Latin-2)
- IBM EBCDIC - Thai
- IBM EBCDIC - Turkish (Latin-5)
- IBM EBCDIC - Turkish
- IBM EBCDIC - U.S./Canada (037 + Euro symbol)
- IBM EBCDIC - U.S./Canada
- IBM EBCDIC - United Kingdom (20285 + Euro symbol)
- IBM EBCDIC - United Kingdom
- IBM5550 - Taiwan
- ISCII Assames
- ISCII Bengali
- ISCII Devanagari
- ISCII Gujarati
- ISCII Kannada
- ISCII Malayalam
- ISCII Oriya
- ISCII Punjabi
- ISCII Tamil
- ISCII Telugu
- ISO 2022 Japanese JIS X 0201-1989
- ISO 2022 Japanese with halfwidth Katakana
- ISO 2022 Japanese with no halfwidth Katakana
- ISO 2022 Korean
- ISO 2022 Simplified Chinese
- ISO 2022 Traditional Chinese
- ISO 6937 Non-Spacing Accent
- ISO 8859-1 Latin I
- ISO 8859-15 Latin 9
- ISO 8859-2 Central Europe
- ISO 8859-3 Latin 3
- ISO 8859-4 Baltic
- ISO 8859-5 Cyrillic
- ISO 8859-6 Arabic
- ISO 8859-7 Greek
- ISO 8859-8 Hebrew
- ISO 8859-8 Hebrew
- ISO 8859-9 Latin 5
- Japanese (Katakana) Extended
- Japanese (Latin) Extended and Japanese
- JIS X 0208-1990 \& 0121-1990
- Korean (Johab)
- Korean Extended and Korean
- MAC - Arabic
- MAC - Croatia
- MAC - Cyrillic
- MAC - Greek I
- MAC - Hebrew
- MAC - Icelandic
- MAC - Japanese
- MAC - Korean
- MAC - Latin II
- MAC - Roman
- MAC - Romania
- MAC - Simplified Chinese (GB 2312)
- MAC - Thai
- MAC - Traditional Chinese (Big5)
- MAC - Turkish
- MAC - Ukraine
- OEM - Arabic
- OEM - Baltic
- OEM - Canadian-French
- OEM - Cyrillic (primarily Russian)
- OEM - Greek (formerly 437G)

[^1]
## Formatted Data Format

This powerful format allows you to convert a binary data to a sequence of numbers, encoded according to the selected type. You select a type of a value, whether base 16 or base 10 is used to represent them, the number of values in a row and output type

When selected data is not too complex, it is automatically put into the Clipboard in this format when user executes the Edit »Copy or Edit » Cut and cursor is in the code pane. The Elements combo among with Group combo provide you with a choice of number type and grouping mode:

| Group/Elements BYTE | WORD | DWORD | QWORD |  |
| :--- | :--- | :--- | :--- | :--- |
| Hex | 8-bit <br> hexadecimal <br> values | 16-bit <br> hexadecimal <br> values | 32-bit hexadecimal values | 64-bit hexadecimal values |
| Decimal | 8-bit decimal <br> values | 16-bit <br> decimal <br> values | 32-bit decimal values | 64-bit decimal values |
| Octal | 8-bit octal <br> values | 16-bit octal <br> values | 32-bit octal values | 64-bit octal values |
| Binary | 8-bit binary <br> values | 16-bit binary <br> values | 32-bit binary values | 64-bit binary values |
| Float | N/A | N/A | Single-precision floating-point values <br> (encoded according to IEEE 754 standard). | N/A |
| Double | N/A | N/A | N/A | Double-precision floating-point values <br> (encoded according to IEEE 754 standard). |

When Elements or Group are set to "Default", their values are taken from the current editor window.
Columns combo lets you specify the number of values in a row. If "Default" is selected, it is taken from the current editor window.
Addresses switch may be used to include row addresses in the output. Text pane switch includes the text pane in the output as well. Text pane is included only if the element type size is one or two bytes. In the latter case, Unicode text appears in the text pane. The text is separated from value list by the tabulation character ( $\backslash \mathrm{t}$ exe9). Both these options are available only for "Space-separated text" and "Comma-separated text" output formats.

Use Type combo to select an output format:

## Without separators

Select data is encoded into a text stream:
6fa864000001c349444154384fbd934f4893611cc7dfcaa16188985887898c31294407319d2df12082d0452112ea344f1578110c092

## Space-separated text

Selected data is encoded into space-separated value list. Below is a sample text (Addresses and Text pane options are turned OFF):
$6173000009617564696 f 73697 a 650041$
$\begin{array}{llllllllllll}72 & 6 f & f 6 & 60 & 00 & 00 & 00 & 00 & 08 & 68 & 61 & 73 \\ 64 & 41 & 75 & 64 & 69 \\ 6 f & 01 & 00 & 0 a & 61 & 75 & 64 & 69 & 6 f & 64 & 65 & 6 c \\ 61 & 79 & 00\end{array}$

Below is a sample text (Addresses and Text pane options are turned ON)
00000140: $6173000009617564696 f 73697 a 650041$ as....audiosize.A
$00000150: 726 \mathrm{f}$
f6
00
00000150: 72 6f f6 6000 00 00 000868617341756469 ro. .......hasAudi
00000160: 6f 010100 0a 61756469 6f 64656 c 617900 o.....audiodelay.

## Comma-separated text

Selected data is encoded into comma-separated value list. Below is a sample text (Addresses and Text pane options are turned OFF)
$61,73,00,00,09,61,75,64,69,6 f, 73,69,7 a, 65,00,41$
$72,6 f, ~ f 6,60,00,00,00,00,08,68,61,73,41,75,64,69$
$72,6 f, 66,60,00,00,00,00,08,68,61,73,41,75,64,69$
$6 f, 01,01,00,0 a, 61,75,64,69,6 f, 64,65,6 c, 61,79,00$
Below is a sample text (Addresses and Text pane options are turned ON)

00000140: 61, 73, 00, 00, 09, 61, 75, 64, 69, 6f, 73, 69, 7a, 65, 00, 41 as...audiosize.A
00000150: 72, 6f, $66,60,60,60,60,60,68,68,61,73,41,75,64,69$ ro. ......hasAudi
00000160: 6f, 01, 01, 00, 0a, 61, 75, 64, 69, 6f, 64, 65, 6c, 61, 79, 00 o.....audiodelay.

## C/C++ array

Selected data is converted into C/C++ array initialization:
C++
c++ unsigned char b1[] = \{


\};
or
C++
unsigned short b1[] = \{
29537, $24841,25717,28521,26995,25978,16640,28530,24822,0,0,26632,29537,30017,26980$,
367, 1, 24842, 25717, 28521, 25956, 24940, 121
\};

## Java array

Selected data is converted into Java array initialization:
byte b1[] = \{
0x61, $0 \times 73,0 \times 00,0 \times 00,0 \times 09,0 \times 61,0 \times 75,0 \times 64,0 \times 69,0 \times 6 f, 0 \times 73,0 \times 69,0 \times 7 a, 0 \times 65,0 \times 00,0 \times 41$, 0x72, 0x6f, 0xf6, 0x60, 0x00, 0x00, 0x00, 0x00, 0x08, 0x68, 0x61, 0x73, 0x41, 0x75, 0x64, 0x69, 0x6f, 0x01, 0x01, 0x00, 0x0a, 0x61, 0x75, 0x64, 0x69, 0x6f, 0x64, 0x65, 0x6c, 0x61, 0x79, 0x00
\};
or
ushort b1[] = \{
29537, 0, 24841, 25717, 28521, 26995, 25978, 16640, 28530, 24822, 0, 0, 26632, 29537, 30017, 26980, 367, 1, 24842, 25717, 28521, 25956, 24940, 121
\};

## JavaScript array

Selected data is converted into Javascript array initialization:

## JavaScript

var b1 = new Array (
0x61, $0 \times 73,0 \times 00,0 \times 00,0 \times 09,0 \times 61,0 \times 75,0 \times 64,0 \times 69,0 \times 6 f, 0 \times 73,0 \times 69,0 \times 7 a, 0 \times 65,0 \times 00,0 \times 41$,
0x6f, $0 \times 01$, ex 01, ex00, $0 \times 0 \mathrm{a}$, 0x61, $0 \times 75,0 \times 04,0 \times 08,0 \times 68,0 \times 61,0 \times 73,0 \times 41,0 \times 75,0 \times 64,0 \times 69$,
);
or

JavaScript
var b1 = new Array (
29537, 0, 24841, 25717, 28521, 26995, 25978, 16640, 28530, 24822, 0, 0, 26632, 29537, 30017, 26980,
367, 1, 24842, 25717, 28521, 25956, 24940, 121
);

## Delphi array

Selected data is converted into Delphi array initialization:
$\mathrm{b} 1=\operatorname{array}[1 . .48]$ of Byte $=($
0x61, 0x73, 0x00, 0x00, 0x09, 0x61, 0x75, 0x64, 0x69, 0x6f, 0x73, 0x69, 0x7a, 0x65, 0x00, 0x41,
$0 x 72,0 x 6 f, 0 x f 6, ~ 0 x 60, ~ 0 x 00, ~ 0 x 00, ~ 0 x 00, ~ 0 x 00, ~ 0 x 08, ~ 0 x 68, ~ 0 x 61, ~ 0 x 73, ~ 0 x 41, ~ 0 x 75, ~ 0 x 64, ~ 0 x 69$, 0x6f, $0 \times 01,0 \times 01,0 \times 00,0 \times 0 a, 0 \times 61,0 x 75,0 x 64,0 \times 69,0 \times 6 f, 0 \times 64,0 \times 65,0 \times 6 c, 0 \times 61,0 \times 79,0 \times 00$
);
or
$\mathrm{b} 1=\operatorname{array}[1 . .24]$ of Word $=($
29537, 0, 24841, 25717, 28521, 26995, 25978, 16640, 28530, 24822, 0, 0, 26632, 29537, 30017, 26980,
367, 1, 24842, 25717, 28521, 25956, 24940, 121
);

## C\# .NET array

Selected data is converted into C\# array initialization:

```
C#
    Byte [] b1 = new Byte [] {
    0x61, 0x73, 0\times00, 0x00, 0x09, 0x61, 0x75, 0x64, 0x69, 0x6f, 0x73, 0x69, 0x7a, 0x65, 0\times00, 0\times41,
    0x6f, 0x01, 0x01, 0x00, 0x0a, 0x61, 0x75, 0x64, 0x69, 0x6f, 0x64, 0x65, 0x6c, 0x61, 0x79, 0x00
};
or
C#
UInt16 [] b1 = new UInt16 [] {
    29537, 0, 24841, 25717, 28521, 26995, 25978, 16640, 28530, 24822, 0, 0, 26632, 29537, 30017, 26980,
    367, 1, 24842, 25717, 28521, 25956, 24940, 121
};
```


## Visual Basic .NET array

Selected data is converted into Visual Basic array initialization:
Dim b1() As Byte = \{ _
61h, 73h, 00h, 00h, 09h, 61h, 75h, 64h, 69h, 6fh, 73h, 69h, 7ah, 65h, 00h, 41h,
72h, 6fh, $66 \mathrm{~h}, 60 \mathrm{~h}, 00 \mathrm{~h}, 00 \mathrm{~h}, 00 \mathrm{~h}, 00 \mathrm{~h}, 08 \mathrm{~h}, 68 \mathrm{~h}, 61 \mathrm{~h}, 73 \mathrm{~h}, 41 \mathrm{~h}, 75 \mathrm{~h}, 64 \mathrm{~h}, 69 \mathrm{~h}$, -
$6 \mathrm{fh}, 01 \mathrm{~h}, 01 \mathrm{~h}, 00 \mathrm{~h}, 0 \mathrm{~h}, 61 \mathrm{~h}, 75 \mathrm{~h}, 64 \mathrm{~h}, 69 \mathrm{~h}, 6 \mathrm{fh}, 64 \mathrm{~h}, 65 \mathrm{~h}, 6 \mathrm{ch}, 61 \mathrm{~h}, 79 \mathrm{~h}, 00 \mathrm{~h}$
\}
or

Dim b1() As UInt16 = \{
29537, 0, 24841, 25717, 28521, 26995, 25978, 16640, 28530, 24822, 0, 0, 26632, 29537, 30017, 26980, _
$367,1,24842,25717,28521,25956,24940,121$
\}
PHP array
Selected data is converted into PHP array initialization:
var $\$ \mathrm{~b} 1=$ array (
0x61, 0x73, 0x00, 0x00, 0x09, 0x61, 0x75, 0x64, 0x69, 0x6f, 0x73, 0x69, 0x7a, 0x65, 0x00, 0x41, $0 \times 72,0 x 6 f, 0 x f 6,0 \times 60,0 x 00,0 \times 00,0 \times 00,0 \times 00,0 \times 08,0 \times 68,0 \times 61,0 x 73,0 \times 41,0 \times 75,0 \times 64,0 \times 69$, 0x6f, $0 \times 01, ~ 0 x \theta 1, ~ 0 \times 00, ~ 0 x \theta a, ~ 0 x 61, ~ 0 x 75, ~ 0 x 64, ~ 0 x 69, ~ 0 x 6 f, ~ 0 x 64, ~ 0 x 65, ~ 0 x 6 c, ~ 0 x 61, ~ 0 x 79, ~ 0 x 00$
);

## Assembler array

Selected data is converted into Assembler language array initialization:



## Encoded Data Format

This format converts selected data into the following formats: Base64, UUencode, Quoted-Printable, Intel HEX and Motorola S-Records.
Its single parameter Algorithm lets you choose the format to be used.
Below is an example of a data encoded in various formats.

## Base64:

YXMAAAlhdWRpb3NpemUAQXJv9mAAAAAACGhhceF1ZGlvAQEACmF1ZGlvZGVsYXkA

## UUencode:

```
begin 666 c:\file.bin
M87,``EA=61I;W-I>F4`07)0]F`....."&AA<T%U9&EO` 0$` "F%U9&EO9&5L
#870
end
```


## Quoted-Printable:

as $=00=00$ audiosize= $=00$ Aro $=F 6^{\circ}=00=00=00=00=08$ hasAudio $=01=01=00=0$ Aau $=$
diodelay=00

## Intel HEX:

```
10000000CFBEC96422B97C49A6963FA3E62F8FA331
080010000100000032000000B5
:00000001FF
```


## Motorola S-Records

```
S1130000CFBEC96422B97C49A6963FA3E62F8FA32D
```

S10B00100100000032000000B1

## NTFS Streams

NTFS (a native file system for all "NT-based" operating systems) has a little-known and usually underestimated feature, called alternate data streams. Each file and directory on NTFS-formatted volume may have an unlimited number of data streams. Each stream may be of any size, provided there is enough free space on the volume. Every file on the volume always contains at least one stream, but may also contain other streams. Unlike the first, default stream, which is unnamed, other file streams have names, which follow the same rules, as defined for naming files and folders on NTFS volume

By convention, to refer to a specific named stream within a file, add the colon followed by a stream name to a full file name. For example, if we have a named stream AltStream in a file $c: \ t e m p \backslash f i l e . b i n, ~ t h e n ~ i t s ~ f u l l ~ n a m e ~ i s ~$

## PowerShell

c: \temp \file.bin:AltStream
This named stream can almost be considered as a separate file, which has its own attributes, such as size, sparse-ness and so on. At the same time, it shares several attributes, such as security descriptor, with its "parent" file

In addition, the system automatically copies or moves all file's streams each time a file is copied or moved.
Support for alternate data streams is quite limited in standard OS tools such as Windows Explorer, Command Prompt, Powershell or Windows Terminal. Those tools may fail to correctly report the presence of streams or their sizes.

NTFS's alternate data streams is not a widely used feature, although, it is slowly becoming more popular. Several common usage scenarios are provided below:

- Several Web browsers create a small (usually 26 bytes long) stream called "Zone.Identifier" in each file downloaded from the "outside" world. This file contains a zone identifier, naming the original zone a file came from. This stream is later used by Windows Explorer, for example, to warn you that you are about to launch a downloaded file. As soon as you open file's properties and click the Unblock button, the stream is deleted.
- Several image viewing utilities are known to create a named stream and save a thumbnail version of the image inside each catalogued image file on your disk.
- Several anti-virus products store the information about last scan time and results in each scanned file so they can subsequently quickly access it without the burden of maintaining complex database.
- There have been reports of several malware and viruses that use the alternate data streams NTFS feature. Although the Windows Explorer and Windows Command Prompt are unaware of streams, Win32 API supports streams quite well. For example, a program or a script may start any named data stream for execution. Thus, a zerosized file may actually contain quite a big executable in one of its streams, or the executable may be disguised as a simple harmless text file. Luckily, most anti-virus products today are capable of scanning NTFS streams and locating viruses and malware in them. Hex Editor Neo can also be used to locate named streams on your computer.


## Streams Support in Hex Editor Neo

Hex Editor Neo provides a rich toolset to work with NTFS alternate data streams. Most of the tools are available through the NTFS Streams Tool Window.
The editor automatically detects and displays all named streams of each opened file. It allows you to open any stream for viewing or editing, delete a stream or create a new stream. It also implements a Find Streams function, which allows you to locate files, satisfying a given criteria, that contain one or more named streams of data. The result window then allows you to open them in the editor, or delete them.

File Attributes Tool Window displays the total number of streams in a file, as well as three file size values: the size of the main, unnamed stream - this is a size reported by Windows Explorer and most other programs; the size of all named streams; and the total size occupied by a file, that is, a sum of two previous values.

Find in Files supports searching and replacing a pattern in named data streams.

## NTFS Streams Tool Window

NTFS Streams tool window displays a list of streams. It operates in two modes. The first and default mode is to display all streams of the active document. The window automatically updates as you switch between opened documents.


Execute the NTFS Streams»Create Stream...command to create a new stream. You will be asked to provide a stream name. A file cannot have two streams with the same name. A stream name must also be a valid file name, that is, it cannot contain the following characters: /, : , ?, *, ", <, >, |

To rename a stream, select it and execute the NTFS Streams » Rename Stream...command. Note that the operating system lacks support for renaming streams, so the Hex Editor Neo emulates renaming by copying the whole stream. It can take a long time for large streams. In this case a progress bar is displayed and you may use the NTFS Streams » Cancel Current Operation command to cancel stream renaming.

To delete a stream (or several streams), select it (or them) in the list and execute the NTFS Streams "Delete Stream(s)...command. Please note that the delete operation cannot be undone.

To open stream (or several streams) in the editor, select them and execute the NTFS Streams » Open Stream(s) command. All selected streams are opened in the editor. You may edit them as you edit normal files. The Always Create Backups Option does not affect streams - backup is never created for a saved modified stream.

If the current document in the editor is a stream and not a file, NTFS Streams Tool Window displays a notification message, because NTFS does not allow streams to have sub-streams. In this case, you may use the NTFS Streams » Open Parent File command to open stream's parent file in the editor.

## Find Stream Mode

The "Find Stream" mode is described in detail in the following section.

## Searching for Streams

Hex Editor Neo provides you with a powerful feature that allows you to locate all files (according to a given criteria) that have alternate data streams in them. It can also be used to quickly calculate the total space occupied by streams on your disk. Found streams may then be opened in the editor or deleted.


Execute the Tools » Find Streams... command to start searching for streams.
Provide a folder or a list of folders (separated by semicolon) in the Search In box. You may also press the Browse button to open up the Folder List Window where you can visually select folders.

Use the File types box to specify the mask. You may specify several masks, separated by semicolons. This mask is used only to filter files, not streams.
You may limit a search by defining the lower bound for a stream size, either in bytes, KBytes, MBytes or GBytes in the Larger than box.
Exclude box lets you enter the regular expression used to exclude located streams. The predefined value (Zone.Identifier)|(encryptable)|(favicon) excludes streams Zone.Identifier, encryptable and favicon, as the system widely uses them and their size is small.

Include sub-folders option may be used to search deep into the selected folders.
When you finish configuring searching options, click the OK button.

## Search Process and Results

As soon as you start searching for streams, the NTFS Streams Tool Windowenters the search mode. In this mode it stops displaying streams of the active document and starts displaying streams satisfying your criteria. The last line in the list always tells you how much streams were found, how much files and folders scanned and total streams size. You may abort searching by executing the NTFS Streams » Cancel Current Operation command.

When the search is finished, all matched streams are displayed in the list. You may sort the list by clicking on any column.
To open a stream (or several streams) in the editor, select it (or them) and execute the NTFS Streams » Open Stream(s) command. All selected streams are opened in the editor. You may edit them as you edit normal files.

To delete a stream (or several streams), select it (or them) in the list and execute the NTFS Streams » Delete Stream(s)...command. Please note that the delete operation cannot be undone.

To exit a search mode execute the NTFS Streams » Exit Find Streams Mode command.

## Operation History

The Hex Editor Neo widely uses the concept of operation history. Operation history is a series of operations performed on thedocument. The history is a document's property and is therefore shared by all document's editor windows.

An operation is a command executed on the document that caused any modifications to it. Below is a list of all commands that lead to document modification and thus always form an operation:

- Typing (or Write)
- Fill
- Insert
- Delete
- Insert File
- Insert Hex
- Change File Size
- Replace
- Replace All
- Cut
- Merge and Cut
- Paste
- Encrypt
- Decrypt
- Bitwise Operations
- Arithmetic Operations
- Shift Operations
- Case Change Operations
- Reverse Operations

Other commands (such as Find and Copy) do not modify the document's data and do not form an operation
The current document's operation history is always displayed in the History Tool Window. In addition, Edit » Undo and Edit » Redo commands can be used to navigate through the operations.

## Organization

As you issue more and more commands on the document, they are added to its operation history, forming an operation list. There is always a "current" operation in the list. The current operation is the last operation performed on the document. At the very top of the list there is an "Open" or "New" operation in most cases. The "New" operation is used if the document was created and "Open" operation is used if the document was opened.

If the History » Clear History command were used on the document, the root of the list may not be a "New" or "Open" operation.
You can change the current operation using one of the following methods:

- Execute another operation. The operation is added to the operation list and becomes a current operation.
- Execute the Edit » Undo command. The operation, which is previous to the current one (if there is one), becomes a current operation.
- Execute the Edit » Redo command. The operation, which is next to the current one (if there is one), becomes a current operation.
- Select any operation in an operation tree in the History Tool Window. Selected operation becomes current.

When the current operation is not a last executed operation, all operations executed AFTER the current are grayed in theHistory Tool Window. They are then called phantom operations, or operation tail. All these operations are stored in a history, but are not currently affecting the document. Although, they can always be applied again to the document, if you change the current operation.

## Branches

Like in most other editors, the sequence of operations you perform on a document form an operation history. Imagine you create a new document and perform several operations. This is a resulting operation history:

Now, you decide to undo the last operation and execute the Edit » Undo command. This is how the history list looks like now:

```
O}\mathrm{ Open
    Write (3 items)
    Write 0\times11 at 0\times82
    Write 0\times33 at 0\times84
```

The last operation is still displayed, but this time in gray color. It is still available for you, but is not currently affecting a document. If you execute the Edit » Redo command, the operation becomes active again.

But what if you execute a new operation, say, delete a byte at offset $0 x 1 f$ ? In most other editors, the grayed operation is removed and you no longer can access it.
Hex Editor Neo introduces a concept of history branches. Think of an operation history as a tree; then a branch is a full path from the tree's root to a tree's leaf. The number of branches equals the number of leafs. When new operation is added to an operation history, one of the following occurs (providing the Auto Create Branches switch is ON):

- If the current operation is a last one and there are no grayed (undone) operations, the new operation is added to the end of the list and becomes active. This is a standard behavior found in most editors.
- If the current operation is not a last in a list and there are grayed operations following it, a new branch is created. Then, depending on the pinned state of the current branch, either the new operation is put into the newly created branch, or the grayed tail is moved into the newly created branch.

Branches are displayed as tabs at the top of the History Tool Window.

* Default p Branch 1

Additionally, a special icon is added to every tree's fork. See the History Tool Window section for more information.
You can perform the following actions with branches:

## Switch to a branch

To switch to a branch, click its tab.

## Rename a branch

To rename a branch, double click its tab.

## Delete a branch

To delete a branch, right-click it to bring up a shortcut menu and select a "Delete Branch" option.

## Pin or unpin a branch

To pin or unpin a branch, click the pin icon. See below for a description of pinned branches.

## Pinned Branches

When a new branch is created, the tail of the current branch is moved into a newly created branch, the new operation is added to the current branch and it still remains a current one. This is a default behavior of unpinned branch.

When a branch is pinned, a new operation is added to a newly created branch which in turn becomes a current branch. This allows you, for example, to assign a name to an important branch and make sure it stays "untouched" whenever you fork a new branch from it.

## History Tool Window

History Tool Window displays the operation history of the current document. The current document is a document whose editor window is currently active.

## History



- Default

Write (9 items)
Write $0 \times 12$ at $0 \times 1322$
Write 0x32 at 0x1323
Write $0 \times 13$ at $0 \times 1324$
Write 0x12 at 0x132
Write 0x31 at $0 \times 132$
Write 0×24 at 0x1327
Write $0 \times 23$ at $0 \times 1328$
Write 0x42 at 0x132
Write $0 \times 3 \mathrm{a}$ at $0 \times 132$

- Delete ( 6 items) Delete $0 \times 13 \mathrm{c} 3$
Delete $0 \times 13 \mathrm{C3}$
- Delete $0 \times 13 \mathrm{c} 3$
- Delete $0 \times 13 \mathrm{xc3}$

Delete $0 \times 13 \mathrm{c}$
Delete 0×13c

There is a list of branches at the top of the window. Each branch is represented by a tab which holds its name and its pinned state
Operations are displayed in a list in ascending order: older operations are at the top and newer operations are at the bottom. Similar operations (operations of the same type) can be grouped, which is a default behavior. The History " Group Same Operations switch is used to turn grouping on or off. Each operation is displayed with an image and operation name. Operation parameters are also displayed. Grouped items are marked with a special overlay icon. To switch to a given operation, click on it in a list. All document windows are immediately updated to reflect changes. The selected operation becomes a current operation. All operations, issued after the current operation in the same branch become phantom operations and are displayed in gray color. They currently do not affect the document's state.

If a displayed operation is a fork, a special "arrow" icon is displayed (as illustrated on a picture above). You can click on an icon to bring up a list of all branches (except the current one), that are forked from the given operation.

## Operation Tree Actions

In addition to History Commands, several actions may be carried directly on a tree using mouse or keyboard.

- Use arrow keys or mouse to navigate operations in a tree
- Use the mouse to switch to branches (click on tabs).
- Pin or unpin a branch by clicking on its pin icon.
- Press Del button to execute the Purge Tail command
- Right-click the mouse button or press the Menu key to bring up the shortcut menu.
- When on fork, press the spacebar to display a list of branches growing from this fork. Alternatively, click on a arrow with a mouse button or find one in a shortcut menu.


## History Commands

Several Hex Editor Neo commands are related to document's history.

## Edit » Undo

The last executed operation is undone and becomes the phantom operation. The previous operation becomes a current operation. This command is disabled if there is no previous operation.

Complexity: constant-time.

## Edit » Redo

The first operation in operation tail becomes a current operation. If there are no phantom operations, this command is disabled.
Complexity: constant-time.

## History » Clear History

All operations in a history are deleted. The document state is NOT changed, all modifications are "merged" into the single operation, which becomes an operation list root

Complexity: linear-time (depends on the number of operations in history and their complexity).

## History » Purge...

History Purge provides you with three ways of purging an operation history:

- Purge redo tail - all phantom operations (or operation tail) are dropped and removed
- Purge branches - all branches except the current one are dropped and removed.
- Purge all - remove all operations except the very first one. In most cases (if Clear History command was not used), this root operation is "Open" or "New".

Complexity: constant-time. See the Purging History.

## History » Group Same Operations switch

Turns automatic grouping on or off. Automatic grouping groups operations of the same type into one operation in a list. You can expand this operation to see each operation that forms a group.

## History » Auto Create Branches switch

When this switch is on (default), new branches are automatically created if there are any phantom operations at a time you execute a new operation. When this switch is off, phantom operations are silently deleted and new operation is added to the current branch.

## History » Save Operation History..

The entire operation history is compressed and saved to a disk file. It then may be loaded and applied to the same document. See the Saving History section for more information

Complexity: linear-time.

## History » Load Operation History..

Loads a saved operation history from a disk file. Several checks are performed to make sure the history is loaded for the same document. See the Loading History section for more information.
Complexity: linear-time.
History » Full Operation History Tree...
Displays a complete operation history tree. See the Operation History Tree Window topic for more information.

## Purging History

When the History » Purge... command is executed, the following window appears:

| V14. Purge History |  |
| :---: | :---: |
| Select operation to perform: | ок |
| O Purge redo tail | Cancel |
| Clear History | Help |
| Opurge all |  |
| Delete all undone operations. You won't be able to execute a Redo command. <br> Warning: History purge operation cannot be undone |  |

You may select the way you want to purge a history.

- Purge redo tail - all phantom operations (or operation tail) are dropped and removed.
- Purge branches - all branches except the current one are dropped and removed.
- Clear history - all operations in a history are deleted. The document state is NOT changed, all modifications are "merged" into the single operation, which becomes an operation list root.
- Purge all - remove all operations except the very first one. In most cases (if Clear History command was not used), this root operation is "Open" or "New".

Purge History operation cannot be undone.

## Saving and Loading History

## Saving History

The entire operation history tree, complete with all operations and branches may be compressed and saved to a disk file. Later, this operation history may be applied to the same document.

The complexity of this operation depends linearly on the number of operations in a history (in all branches).
Hex Editor Neo automatically stores a document's name and size in a history file.

## Loading History

This command allows you to restore the previously saved operation history tree. Saved history must be loaded for the same document it was created from.
If there is non-empty history in the current document when this command is invoked, the following window appears:

| Load History |  |
| :---: | :---: |
| This document already contains a non-empty history. Select an action to continue: |  |
| O Drop current history and replace it with a loaded one. | ок |
| Retain current history. Optionally add the following prefix to loaded history branches: | Cancel |
| Prefix: | Help |

Select the Drop current history and replace it with a loaded one option to drop the current history and replace it with a one you are loading.
Select the Save current history option if you want to retain the current history. In this case, you may enter an optional prefix to be added to the name of each loaded branch to easily distinguish them from each other.

Operation History Tree Window


The purpose of this window is to display the entire document's operation history. Each operation is displayed as a box with operation's icon and short description. A tooltip with more detailed information is displayed when you hover a mouse pointer over the box.

The window works in two modes: full mode and forks only mode. The Forks Only command is used to switch between modes.

## Full Mode

In this mode, entire operation history tree is displayed. As operation history grows larger, the tree gets more and more complex. It is recommended to switch to the forks only mode in this situation.

## Forks Only Mode

In forks only mode only tree's root, "leafs" and forks are displayed. Thus, all history branches are clearly visible, but the tree stays compact. This mode may be useful for displaying complex history trees

## Navigation

As operation history tree grows large, it becomes impossible to display the entire tree in the window. Use the scroll bars, keyboard keys or mouse to scroll the contents of the window to locate the required part.

Use the Zoom In, Zoom Out or mouse wheel to change the zoom level. The Fit to Screen command may be used to automatically adjust the zoom level to try to completely fit the tree into the window. Please note however that there are minimum and maximum zoom level limits.

You can use a keyboard navigation keys and mouse to select any operation in the tree. The underlying document's state is immediately switched to the selected operation. Press the Enter key, or double-click the operation box to close the window and switch to the selected operation

Press the Esc key to cancel the changes and close a window.

## Legend

Hex Editor Neo uses different colors and line widths to display the current branch, other (than current) branches as well as undone operations in the current branch. You can change the colors using the Select Colors... command.

## Shortcut Menu

Right-click the mouse button on any operation box to bring up the operation's shortcut menu. The following commands are available:

## Switch To

Switch to the given operation and close the window

## Purge Branches

Purge all history branches besides the one holding the given operation. Note that any branch originating below this operation is not purged.

## Purge Tail

Purge all operations below the given one.

## File Comparison

The powerful File Comparison feature may be used to compare any two files using simple comparison algorithm or difference comparison algorithm to locate all changes made to the first selected file in order to turn it into the second file. The feature may also be invoked from the Process Window to compare memory of two process snapshots.

When comparison operation completes, the brief overview is displayed to the user and then full comparison results are listed in the File Comparison Tool Window as well as highlighted in the editor windows. The user may filter and sort the results, click on them to immediately jump to the corresponding location in source documents as well as convert comparison results to multiple selection object.

Colors used by the file comparison feature may be configured using the Themes setting page.
To compare files execute the Tools » Compare Files... command.

## Compare Files Window

This dialog is used to configure the file comparison feature and to start comparing files.

|  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| First file: | CilWindowslexplorer.exe |  | $\checkmark$ | ... | ок |
| Second file: | CilWindowslexplorer.exe |  | $\checkmark$ | ... | Cancel |
| Comparison method: | Simple comparison algorithm | $\checkmark$ | Normal |  | Reset |
| Align documents: | Align horizontally | $\checkmark$ | Colors.. |  | Help |
|  | $\square$ Synchronize windows |  | Swap |  |  |

## First file, second file

Select the first and the second files to compare. All currently opened documents as well as last recently used documents are listed in the drop-down. In addition, you can enter the full path manually, or use the ... button to browse for it. After pressing the browse button, select one or two files. If you select a single file, it is used either as a first file, or second file. If you select two files, they will be used as first file and second file correspondingly.

## Comparison method

Select either the Simple comparison algorithm, Difference algorithm or Difference algorithm (alternate).

## Normal/Precise

The quality/performance level of the difference algorithm.

## Align documents

Select the automatic alignment of compared documents.

## Synchronize windows

Check to synchronize cursor movement in compared documents (applicable only to the simple comparison algorithm).

## Colors button

Press to change colors used to highlight different areas.

## File Comparison Tool Window

This tool window contains the results of the file comparison operation.
For each block, its type, its size and offset in both documents is provided. The block is colored according to its color in the editor window. You can navigate the blocks using the keyboard or mouse, convert the blocks to selection, switch number display between decimal and hexadecimal values.
"Combine mode" is used to display blocks from both documents in the list. When "combine mode" is off, only blocks from the current document are displayed. Switch the documents to view blocks for another one

Click on the column to sort the list by this column. Click again to reverse sorting order.
Click on the Configure Filter... button to configure result filters.

## Comparison Results Filters Window

Result filters are a number of rules which are applied to filter the comparison results. Each rule consists of a column name, operation, value and action:

## Column Name

One of the names of currently displayed columns in the File Comparison Tool Window.

## Operation

One of the following values:

- equal to
- less than
- greater than
- less than or equal to
- greater than or equal to

Value
Operation value, depends on the chosen column.
Action
One of the following values:

- Include
- Exclude

For each comparison result, each rule is applied one by one. If all rules result in include action and none of the rules result in exclude action, then the result is retained, otherwise, it is discarded.

## Operation

Click the Add Filter Rule button to create new rule, Edit Filter Rule button to edit the selected rule. Press the Remove Filter Rule button to delete selected rule.

## Comparison Algorithms

## Simple Comparison Algorithm

This algorithm compares file data byte-by-byte. It uses three types of blocks:

- Matched
- Different
- Not Found

If two bytes are the same in both files - they will be marked as Matched. Otherwise, they are marked as Different. If one file is larger than another, its excess part is marked as Not Found.

This algorithm is convenient if two files are almost equal and there were no data inserted or removed from them.

## Difference Algorithm

This algorithm searches for matched, modified, inserted and removed blocks in both files:

- Matched blocks are blocks of equal data in both files.
- Inserted blocks are blocks of data inserted into the first file. They are marked only in the second file as they are missing in the first one.
- Removed blocks are blocks of data removed from the first file. They are marked only in the first file as they are missing in the second one.
- Modified blocks are blocks of completely different data.

This algorithm has a quality/performance tuning option, specified in the Compare Files Window. If set to "Normal", it may operate faster, but less accurately, and if set to "Precise", it may operate slower (sometimes much slower), but will be able to find more matches.

Starting from version 7.21, there is a second implementation of a difference algorithm (called "Difference Algorithm (alternate)"), which should provide more accurate results for closely related files.

## Patches

A patch is a difference between two states of a document. In most cases the size of this difference is much smaller than the size of the file itself. Hex Editor Neo supports creating a special patch file, which contains all modifications made to an opened document. The patch can subsequently be distributed and applied to the same file resulting in the same modifications being made to a file.

Hex Editor Neo does not support applying patches in its framework. Instead, the separate application, PatchApply.exe is provided for this purpose. Alternatively, Patch API provides several other methods of applying patches.

## Distribution

Patch files created by the Hex Editor Neo may be freely distributed without any limitations. This also applies to self-installing patch files.

## Limitations

Currently, Hex Editor Neo does not allow you to create a patch if a file contains data, pasted from other documents. If a file contains data pasted from itself, the patch may be created.

## Creating Patches

To create a patch, you must have an opened and modified document. Execute the File » Create Patch... command.

| Create Patch | $\times$ |
| :---: | :---: |
| Patch file name: | ОК |
| マ ... |  |
| Patch creation options | Cancel |
| Save original file name | Help |
| $\square$ Calculate and save file hash (slower) |  |
| $\square$ Remove references to other files (may create larger patch file) |  |
| Patch packaging options |  |
| $\square$ Self-installing patch |  |
| O Target x86 platform |  |
| Target 664 platorm |  |

## Patch file name

Enter the full path name to the patch file or use the Browse (...) button. Hex Editor Neo automatically appends the .hexpatch file extension if you omit it. If Self-installing patch option is turned on, .exe extension is appended instead.

## Save original file name

Original file's name is saved in the patch file and then compared with a name of file the patch is applied on. If this option is turned off, file name comparison is skipped.

## Calculate and save file hash

A hash value is calculated for the whole file. Later, when the patch is applied, the hash value is calculated for the target file and compared to this value.

## Remove references to other files

This option is not currently implemented.

## Self-installing patch

The patch file is packaged inside the executable file. It can subsequently be used without the PatchApply.exe application. You may specify whether you want to target any platform ("Target x86 platform") or x64 platform only ("Target x64 platform"),

After you press the OK button, Hex Editor Neo starts creating a patch. This may take a while

## Applying Patches

A separate application, the PatchApply.exe utility, is used to apply patches created by the Hex Editor Neo. The patch may also be packaged inside the executable file. In this case, nothing else (except the target file, of course) is required to apply the patch.

Patch API may also be used to apply the patch.
Care must be taken to make sure the patch is applied to exactly the same file. Hex Editor Neo provides three simple mechanisms to validate the file's identity:

1. Hex Editor Neo always compares the size of the source and target files. If they differ, patch cannot be applied. This option cannot be disabled by the user.
2. Hex Editor Neo may store and compare the name of the source and target files. You may disable this option.
3. Hex Editor Neo may compute the hash value for both files and compare it. You may disable this option. Hash calculation may significantly slow down patch creation/applying but provides quite a good degree of confidence.

## PatchApply.exe

The Apply Patch utility is launched from the Hex Editor Neo's group in the Start menu.


Hex Editor Neo always installs two versions of the utility. One is a native x86 application and another is a native x64 application. Only x86-platform version may be used on x86 platform and any version may be used on x64 platform (but x64-platform version is preferable).

## Path to file to patch

The full path to the target file you are going to apply patch on. You may enter the path manually or click on the Browse (...) button.

## Path to the patch file

The full path to the patch file. You may enter the path manually or click on the Browse (...) button.

## This patch file

This switch is active if the patch file is encapsulated in the executable (self-installing patch).

## Do not validate file name

File name validation is disabled. This option takes no effect unless "Save original file name" option was active during patch creation.

## Do not compute and compare file hash

Skip calculation and validation of file's hash value. This option take no effect unless "Calculate and save file hash" option was active during patch creation.

## Do not create backup

Skip creation of the backup copy of the target file. Use this option with care as you may lose your data if wrong patch is applied to a file, or patch is applied to a wrong file.

## Command Line Interface

PatchApply.exe utility also provides the command-line interface
Powershell
PatchApply.exe [<target file>] [/patch <patch file>] [/noui] [/silent] [/skipfilename] [/skiphashcheck] [/skipbackup] [/help | /?]
where
target file
full path to the file being patched
patch file
full path to patch file (omit for self-installing patches)
/noui
do not display user interface (except progress and status)
/silent
do not display progress and status
/skipfilename
do not validate target file name
/skiphashcheck
do not calculate and compare file hash
/skipbackup
do not create target file backup
/help or /?
display this information

## Patch API

Patch API provides the programming interface to the Hex Editor Neo's patch application functions. Patch API may not be used to create a patch, only to apply it.
Hex Editor Neo's installer provides an option to install the Patch API on your computer.
More information on the API is found in separate documentation file, installed with a Patch API.

## Pattern Coloring

Hex Editor Neo provides a feature called "Pattern Coloring" that can be used to automatically highlight specific patterns in an edited document. You define a pattern and specify the coloring applied to it in an editor window. Pattern Coloring fully supports regular expressions.
"Range Rule" type also allows you to specify a range in a document and coloring to apply to this range.
A Pattern Coloring Tool Window is used to define one or more patterns and coloring rules. For each defined pattern (or range), background color, foreground (text) color, outline color and rounded edges flag are defined. It is recommended to specify semi-transparent background colors. In this case, if several modules apply coloring to the same editor's cell, it is still possible to differentiate them.

## Pattern Coloring Tool Window

Pattern Coloring Tool Window is used to define a set of rules that associate specific patterns or data ranges with coloring rules.
Pattern Coloring
(द) 4 C) Cx C
\# Pattern

- $2\{0 \times 0 \mathrm{Od}, 0 \times 0 \mathrm{a}\}$ (BYTE)
- $\{0 \times 0 \mathrm{a}, 0 \times 0 \mathrm{O}\}$ (BTE)

The Tools » Pattern Coloring »Enable Pattern Coloring switch is used to quickly disable or enable application of all defined coloring rules.

## Creating and Editing Rules

Use the Tools » Pattern Coloring » Add New Rule command to define a new pattern coloring rule.


You define a pattern using the Pattern Window, which is part of this dialog. You also specify whether to ignore case while matching the pattern. You can then specify the foreground and background colors, outline color and Round edges flag. In addition, description may be specified for a pattern. It is then displayed in a tooltip when you hover the mouse over highlighted pattern in an editor window.

Instead of entering a pattern to match in a document, you may create a range rule. For this rule, you enter a range by specifying its starting offset and size in either hexadecimal or decimal formats. The coloring you specify is then applied to the given range in all editor windows.

To edit a rule, double-click it in a list, or bring up the shortcut menu and choose the "Modify Rule..." option.
To remove a rule, select it in a list, bring up the shortcut menu and choose the "Remove" option

## Rule Schemes

All defined rules comprise the pattern coloring scheme. A scheme may be saved to a disk file and later loaded from it. To do it, execute Tools » Pattern Coloring » Save Scheme... or Tools» Pattern Coloring » Load Scheme... correspondingly. To delete all currently defined rules, execute the Tools» Pattern Coloring » Clear Coloring Scheme command

Hex Editor Neo automatically records all defined rules and restores them next time you launch the application.

## Regular Expressions

The Pattern Coloring fully supports regular expressions. To search using regular expressions, select either "ASCII string (char[])" or "UNICODE string (wchar_t[])" pattern type, enter the regular expression, make sure the "Regular expression" checkbox is checked and enter the sub-expression number you want to search for. Sub-expression " 0 " represents the expression itself.

Take the following limitation into account

- Complex regular expressions that start long before a visible area and/or end long after a visible are may not be highlighted.


## Structure Viewer

Structure Viewer interprets document's data according to some structure definition. A structure is an ordered set of fields, each of which has defined type. A type may be a built-in type or another structure.

Structure Viewer allows you to bind structure definition to a given position within a file and analyze file's data according to a structure definition. In addition, you may use parsed fields to modify data in a document.

This section describes features provided by the Structure Viewer and how to control it
Starting from version 7.01, the user may import Kaitai structure definitions, in addition to built-in format.
See also the What's new in Structure Viewer section for more information on latest changes.

## Support for Kaitai Struct

Starting from version 7.01, Structure Viewer supports special kind of definition files, called Kaitai Struct.
Kaitai definition files may be added to the library as standard structure definition files. They are automatically parsed and any defined structures are added to the library. The user may then bind those structures as usual

The built-in Structure Editor provides full support for Kaitai files, including syntax coloring and validation.

## What's New in Structure Viewer

## What's New in Structure Viewer (version 7.25

Version 7.25 adds support for the following built-in integer types:

- long long
- signed long long
- unsigned long long
_SVC_VER Compiler version macro value has been changed to $0 \times 702$.


## What's New in Structure Viewer (version 7.09)

Version 7.09 adds support for synchronization of the local Structure Definition Library with online Github repository.
Users can contribute by submitting corrections to existing format definitions or new definitions for well-known file formats.

## What's New in Structure Viewer (version 7.01)

Version 7.01 adds support for importing Kaitai structure definition files directly to the Structure Library.
using keyword may now be used to define type aliases, in addition to typedef. typedef continues to be supported.
_SVC_VER Compiler version macro value has been changed to $0 \times 701$ to illustrate editor upgrade to major version 7

## What's New in Structure Viewer (version 6.31)

Starting from version 6.31, a new JavaScript engine is used by Structure Viewer. This allows writing JavaScript ES2015 code
This is also a breaking change: the document object is no longer exposed directly to the structure being bound. Only subset of methods and properties are now available. See the file document object topic for more information.
_SVC_VER compiler version macro value has been changed to $0 \times 401$ (major version: 4 , minor version: 1 ), which indicates a major upgrade. The major version change is related to breaking change in support for the file document object and the version of JavaScript engine.

What's New in Structure Viewer (version 6.25)

## New Language Features

## SVC_VER compiler version macro

Compiler version in 6.01 has a value of $0 \times 307$ (major version: 3, minor version: 7), which indicates minor upgrade.

## New directives

New directives are supported: \$revert_to, \$shift_by and \$remove_to. New field attribute exact_only.

## What's New in Structure Viewer (version 6.01)

This is a minor upgrade to Structure Viewer. This release also introduces the new built-in structure definition file editor.

## New Language Features

## _SVC_VER compiler version macro

Compiler version in 6.01 has a value of $0 \times 306$ (major version: 3 , minor version: 6), which indicates minor upgrade.

## New built-in value type

New built-in value type ByteArray is now supported.

## New built-in functions

This release adds the following new functions: array, length, element, subarray, substring and find

## Structure Library revised

Structure Library is now a separate tool window.

## Fixed bugs

## Various function scope bugs fixed

Invalid scope rules were used when executing native functions

## Incorrect if statement branch optimization

The editor could incorrectly optimize if statement branch in some cases.

## Incorrect language grammar

Structure definition language grammar was fixed: it sometimes erroneously prohibited otherwise legitimate constructs.

## What's New in Structure Viewer (version 5.12)

This is a minor upgrade to Structure Viewer. In addition to several bug fixes

## New Language Features

## SVC_VER compiler version macro

Compiler version in 5.12 has a value of $0 \times 305$ (major version: 3, minor version: 5), which indicates minor upgrade
New predefined macros
This version adds the following predefined macros: _SVC_X86, _SVC_X64, _SVC_POINTER_SIZE, _SVC_POINTER_SIZE_REAL.

## Optimization of core algorithms

Optimization of core algorithms improved binding time by about 10\%. This is more noticeable for complex structures, where a lot of computations and other expression evaluation occurs during binding

## Integer type modifiers

It is now possible to use little_endian and big_endian to change integer's type default byte ordering. This functionality complements the \#pragma byte_order directive and has priority over it.

## New type [display] attribut

This release introduces the first type attribute: display. Previous versions automatically generated values for collapsed items. Values of first 5 type's fields in curly braces is displayed by default. Version 5.12 allows you to override this behavior for a specific type. You provide an expression, whose evaluated value is used. This expression is evaluated in current type's scope and has access to all its fields

## New built-in functions

New functions — is_valid() Function and evaluate_if() Function have been added to a list of supported built-in functions.

## Limited support for reference

Version 5.12 adds support for taking references of fields. References may be used to pass fields (including arrays and complex type fields) as arguments to native functions or to compare two objects. A new built-in function ref is used to get a field's reference.

## New method in Parser object

This release adds the IParser.add_coloring_scheme method to the IParser interface.

## Error removing non-existing files from the library

It was impossible to remove a non-existing structure definition file from the library.

## Invalid operation

In some rare cases Structure Viewer could read garbage from the document instead of real data.

## Errors updating Output window

Several inconsistences with Structure Errors tool window and present/absent definition files have been fixed.

## Errors in native functions implementation

Assignment statement did not work in native functions, which also resulted in inability to use for statement in them.

## Incorrect scope for array indexing operator

Array indexing operator used incorrect scope while evaluating the index expression, which could led to bugs or "ID not found" exceptions.
Crash after removing file from the library
Application could crash after removing structure definition file from the library (after it stopped watching its directory for changes, that is, after the last file from that directory was removed).

## Crash when trying to remove used file from the library

Application crashed when you tried to remove used structure definition file from the library.

## Very long string handling issue

Structure Viewer could crash when it tried to display a very long string (longer than 65536 characters).

## What's New in Structure Viewer (version 5.01)

This is a major upgrade to Structure Viewer. This module has almost been rewritten from scratch, taking advantage of new compiler, parallel processing (on multi-processor and multi-core computers), new features and a major performance boost.

Hex Editor Neo does not perform pre-compilation anymore. The speed of compilation is now 10-100 times faster and is performed by multiple cores. Structure Viewer initialization no longer slows down application start up.

New version improves the binding time and reduces memory consumption of bound structures.

## New Language Features

## SVC_VER compiler version macro

Compiler version in 5.01 has a value of $0 \times 300$ (major version: 3, minor version: 0), which indicates major upgrade

## Breaking change

break operator now always exits the current scope, even if not used inside a loop or switch statement.

## Breaking change

When defining constants, the expression at the right side must evaluate to a constant value at compile time. Compare this to variables

## Breaking change

Visual Basic Scripting is not supported anymore. You can create external functions on Javascript only.

## Alternative syntax for specifying enumeration's base type

New syntax (the one described in $\mathrm{C}++11$ ) for specifying enumeration's base type may be used. The old one is still supported.

## Update to \#pragma byte_order behavior

This pragma may now be used at any scope. It changes the byte ordering immediately (effective for next fields if used inside a user-defined type, for example).

## Non-fatal \$assert directive is supported

This release finally makes use of non-fatal \$assert directive possible (it was documented, but not supported in previous releases).

## Updating of array elements

Previous versions allowed you to declare variable arrays. This version adds support for modifying individual array elements after they have been declared (and possibly initialized).

## New attributes

In addition to noindex attribute, the following attributes introduced:

- noautohide - allows you to turn off automatic hiding of empty fields.
- onread - specify the expression to be evaluated each time the value is read.
- format - specify the format string to be used when displaying the field's value.
- description - specify the user-friendly description of the field
- color_scheme - specify the color scheme for the field.


## New built-in functions

The following new built-in functions were introduced: bool, decimal, hex, octal, binary, integer_convert and format.

## Support for native functions

In addition to existing support for using JavaScript functions (either inline or defined in external files), new Structure Viewer allows you to define native functions. This also introduces the return statement.

## Automatic optimization of sub-expressions

Previous versions of Structure Viewer could automatically optimize constant expressions. New version adds automatic optimization to sub-expressions as well.

## New directive

New \$alert directive allows you to evaluate an expression at bind time and display the result in a message box.

## New UI Features

## Support for color schemes

Combined with new color_scheme attribute, allows the customer to apply different color schemes to individual fields in a bound structure.

## Structure Library dialog improvements.

Various improvements of the Structure Library dialog.

## External structure editor support

Added support for external structure editor.

## improved error detection and reporting

This version dramatically improves syntax error detection and reporting. The editor is now capable of detecting more errors and provides more accurate position for an error. It is also now has so-called "expectation points" in its grammar. For example, in the following code fragment:

C++
public struct A
int $a[5 ;$

A "syntax error" with message "] expected" is displayed. A cursor points to the ';' character.

## Change to bind error behavior

Now, when bind error occurs, all items that have already been bound are left in the Structure Viewer. Exception item is added to the end of the list of bound items.

## Structure Errors tool window

New tool window displays all compilation and binding errors. You can double-click any entry in a list to display the position of the error.

## What's New in Structure Viewer (version 4.96)

Hex Editor Neo is a constantly developing application. Structure Viewer in version 4.96 adds support for the following:

## SVC VER compiler version macro

Compiler version in 4.96 has a value of $0 \times 206$ (major version: 2, minor version: 6).
Incorrect bit field values editing fixed : This version fixes the error in editing bit field values.

## Regular Expressions in File-Scheme Association

Structure Viewer now uses regular expressions to associate a file and a scheme.

## What's New in Structure Viewer (version 4.92)

Hex Editor Neo is a constantly developing application. Structure Viewer in version 4.92 adds support for the following:

## _SVC_VER compiler version macro

Compiler version in 4.92 has a value of $0 \times 205$ (major version: 2 , minor version: 5 ).

## New statements

New statements are now supported: while, for and do...while.
In-line javascript code
Javascript code may now be specified in the same file, without using the \#pragma script directive. A new javascript keyword is used for this.

## Changes in \$print directive operation

Before this version, \$print directive always converted its argument to string. Now it preserves the argument's type and uses Structure Viewer's visualization engine to display it. It will now appear exactly as any other parsed field. And do not remember, that \$print directive not only displays its value, it also introduces new field into the scope.

## New visualization mode for pointers

Hex Editor Neo now also prints computed address for a pointer as well as "peeks" inside the pointed type (as it does for structures and arrays).

## What's New in Structure Viewer (version 4.71)

Hex Editor Neo is a constantly developing application. Structure Viewer in version 4.71 adds support for the following:

## SVC_VER compiler version macro

Starting from the current version, Structure Viewer defines a macro that holds a current compiler version. Compiler version in 4.71 has a value of $0 \times 200$ (major version: 2, minor version: 0 ).

## New built-in types

The parser now natively recognizes three additional types: bool (one-byte boolean type), string (null-terminated single-byte character string) and wstring (null-terminated two-byte character string).

## Dynamic typing

When evaluating expressions, dynamic typing rules are used. For example, adding two integers produces an integer, but adding a floating-point number and an integer number produces a floating-point number. The same rules apply to other types and other operators.
Note that this feature introduces a breaking change: the syntax for defining constants has changed.

## Variables

You can now define variables, and change their values at a later time. Variable's type is polymorphic, that is, it may be changed by assigning a value of another type.

## Constant and Variable Arrays

You can now define constant or variable arrays and reference their elements in expressions:
const Months[]=\{
"January",
"March",
April",
"May",
"June",
"August",
"August",
"October"
"November",
"December"
\};
public struct A
short month
\$print("Month as string",Months[month])
\};

## Scripting support

You may now call functions written in Javascript or VBScript from the Structure Viewerexpressions. In addition, a number of internal functions are provided. Structure Viewer makes two objects, parser and document, available to the running script.
functions.js:

```
JavaScript
    function ScriptAdd(a,b)
{
}
function IsModified()
return document.Modified;
structure.h:
#pragma script("functions.js")
struct A
{
        if (IsModified())
        $print("warning","The document has been modified!")
    char array[ScriptAdd(10,8)]; // equivalent to char array[10+8];
};
```


## Statements

Hex Editor Neo's parser now understands the if statement and switch statement.

## "Infinite" Arrays

An array may now be declared as having an infinite (unknown) number of elements. Combined with a new \$break_array directive, this allows declaring arrays for which the number of elements is not known beforehand.
\#pragma byte_order
You may now instruct the parser how your multi-byte values are packed.

## Array optimization

Arrays of simple integer types are now optimized by the parser, reducing the binding time and memory consumption. In addition, you may refer to character arrays (arrays of type char and wchar_t) in expressions. In this case, the contents of an array is represented as a string:

```
C++
struct A
    char TagName[10];
    if (TagName == "#Default")
        // do something
};
```


## hidden: and visible: directives

You may now use "hidden:" and "visible:" directives to change the visibility of a field in a user-defined type. The visibility only affects the displaying of the field, otherwise it remains visible when referenced in expressions.

## public and private user-defined types.

A user-defined type may now be marked as private or public: private types are not displayed in the Bind Structure Dialog.

## \$bind directive

\$bind directive allows a structure definition to bind another structure.

## current offset built-in variable

You may now use the current_offset pseudo variable in expressions. It will always equal to the offset the next field will be bound to.
array_index built-in variable
This variable evaluates to a current array index when used in an user-defined type. For example:

```
C++
struct B;
struct A
    B b[10]; // array of 10 B's
};
struct B
int b;
    if (array_index == 3) // fourth element contains another integer
        int c;
};
```


## "Synchronize Position" mode

A new "Synchronize Position" mode allows you to easily match the location of a bound field anywhere in the document.

## Overall performance enhancements and several bug fixes.

Structure Viewer can now easily handle extremely complex structures and arrays with billions and billions of items. Navigating such complex structure is very simple and fast.

## Structure Library Tool Window

A structure is defined in a text file (that usually has .h extension). A file may contain one or more structure definitions and may also reference other files with an \#include directive. The syntax for structure definition is defined in the Language Reference section.

A Structure Viewer Library is used to manage a list of structure files from which a list of defined structures is taken. To access the Structure Library, open the Structure Library tool window by executing the View » Tool Windows » Structure Library command.

| Structure Library |
| :---: |
|  |
| ¢－目pal．h |
| 由－目pcx．h |
| 本－目pe．h |
| ¢自 pic．h |
| （自prg．h |
| 由自pssd．h |
| ¢－自rar．h |
| 目 stddefs．h |
| －目 strings．h |
| 由－目tga．h |
| （目tiff．h |
| ©－目 vhd．h |
| 由目wav．h |
| ¢－目qoi．h |
| －由－自zip．h |
| －－Structures |
| 㿣 RIFFHeaders |
| 國 BITMAPFILEHEADER |
| －CAB＿File |
| －CDA |
| －CDFS |
| －VolumeTest |
| －CHMHeader |
| 㬂 EmfHeader |
| 6 moc Ein |

The library window displays a list of managed files and a list of structure definitions，taken from them．They are grouped under the Files and Structures folders（or sections） correspondingly．

Double－click the file to open its contents in the built－in editor．Double－click the structure name to open its definition file in the built－in editor．Cursor is automatically moved to the structure definition location．

Hex Editor Neo watches for modifications of bound structure files added to the Structure Library．If the file is changed outside the Hex Editor Neo，you are offered to reload and re－scan it．

Starting from version 7．01，Structure Viewer allows Kaitai definition files to be added to the library．Added files are listed in the Files section as well as parsed structure definitions are listed in the Structures section．

## Library Management

Use the Tools » Structure Viewer » Library » Add Structure File（s）．．．command to add an existing structure definition files to the library．A file is compiled and a list of error messages is displayed in Output Window．

Use the File » New » New Structure File or File » New » New Kaitai Fileto create new structure definition file or Kaitai definition file correspondingly．File » Open » Open Structure File．．．command can be used to open external structure definition files or Kaitai definition files in the built－in editor．

Opened structure definition file may be added to the library using the Tools » Structure Viewer » Structure Library » Add Current Structure File command．
Highlight the file in the library window and execute the Tools» Structure Viewer » Library » Remove Structure File to remove a file from the library．
Tools » Structure Viewer » Library » Open Structures Folder command opens the configured structure definition folder in Windows Explorer
Most of these commands are also accessible via the window＇s context menu．

## Compilation

Hex Editor Neo compiles all structure definition files on each launch．It also automatically re－compiles definition files when they are modified using built－in editor or by external program unless the Tools»Structure Viewer » Library »Auto Recompile option is turned OFF．You may manually force the editor to recompile all structure definition files by executing the Tools» Structure Viewer » Library » Compile command．

Compilation errors are displayed in the Output Window．You can double－click on the error to open the built－in editor and navigate to the place the error is generated．

## Structure Editor

Starting from version 6．01，Hex Editor Neo includes built－in Structure Editor．Use this editor to modify any existing structure definition or to create new definitions．The editor has built－in syntax coloring and you may customize used colors as described in Structure Editor Coloring section．Starting from version 7.01 ，the built－in editor may also be used to create or edit Kaitai definition files．

Built－in structure editor allows you to view and edit structure definition files．To open a file，use one of the following：

## Structure Library Tool Window

Double－click on any file or structure in a list to automatically open it in the editor．

## File » New » New Structure File or File » New » New Kaitai File

Use this command to create a new structure definition file or Kaitai definition file and open it in the editor．The file is not added to the library until you save and explicitly add it using Tools» Structure Viewer » Structure Library » Add Current File command．

## File » Open » Open Structure File

Use this command to open an external structure definition file or Kaitai definition file（that is，file that has not yet been added to the library）．The file can be added to the library using Tools » Structure Viewer » Structure Library » Add Current File command．

Editor uses syntax coloring（which you may customize as described in the Structure Editor Coloring section）and provides advanced editing capabilities．
Find

| O Find |  |  |
| :---: | :---: | :---: |
| Find what：PACKEI | $\checkmark$ | Eind |
| Options | Find within | Cancel |
| Match whole word only | O Whole window | Help |
| $\square$ Match case | Selection |  |

Use the Edit »Find．．．command to bring up the Find window．Enter the pattern to search，configure options and press the Find button．Then use the Edit＂Find Next command to jump to the next pattern occurrence．

Replace

| 2. Replace $x$ |  |  |
| :---: | :---: | :---: |
| Find what: var | $\checkmark$ | Eind Next |
| Replace with: | $\checkmark$ | Replace |
| Options | Find within | Replace All |
| $\square$ Match whole word only | O Whole window | Cancel |
| $\square$ Match case | Selection | Help |

Use the Edit » Replace... command to bring up the Replace window. Enter the source and replacement patterns, configure options and then press the Replace or Replace All button.

## Go to Line

| 莖 Go to Line |  |  |
| :---: | :---: | :---: |
| Line number 1-357 |  |  |
| 1 |  |  |
| ок | Cancel | Help |

Use the Edit » Go to Line command to bring up the Go to Line window. Enter the line number and press OK button.

## Structure Editor Coloring



This window allows you to configure fonts and colors used by Structure Editor. You can change the appearance of the following categories:

## Comments

Comments are standard C/C++ comments sections, enclosed between /* and */ or one-line commentaries started with //

## Functions

Built-in functions.

## Identifiers

Identifiers, such as field names, variable names and structure names

## Keywords

Structure definition language keywords, such as statements and types
Numbers
All numeric values.

## Operators

All supported operators.

## Plain tex

Other un-categorized text.

## Pragmas

Pragma declarations.

## Preprocessor

Preprocessor directives.
Strings
String literals.

## Structure Binding

In order to be able to use any structure defined in the Structure Library, you must bind it to the document's data.
To bind a structure, execute the Tools » Structure Viewer » Bind... command. This brings up the Bind Structure window:


Select the structure you want to bind from the Type list. You can also press the Browse (...) button to go to the Structure Library where you can add new source files, for example. Give a bound structure a name - this will allow you to differentiate between several bound structures of the same type and to reference structure's fields.

Specify the address of the structure in a file. Address may be specified as hexadecimal or decimal value. You may select one of the following bind types which change the meaning of the address value:

## Absolute address

The structure is bound to the absolute address. The structure remains at the given address regardless of any actions performed on the document.

## Relative address

The structure is bound to the given address. The structure is then moved each time you insert or delete data before its start address. This does not relate to complex editing commands, such as Delete (with multiple selection), Replace or Replace All.

Cursor : The structure is bound to the cursor's location and moves with it.
Alternatively, you can enter the address of a structure as an expression. An expression is allowed to use constants, enumerations (defined in the global scope of any of added source files) and fields of already bound structures.

You also specify two coloring rules (one for the whole structure and another for a structure's field). The first coloring is applied when a bound structure is visible in an editor window. When you select a structure's field in a Structure Viewer Tool Window, the second coloring rule is applied to each cell the field occupies in an editor window.

## Structure Viewer Tool Window

Structure Viewer Tool Window displays structures bound to the current document.


For each bound structure, its name, type, binding address and coloring sample are displayed. Click on the plus button to expand the structure
A checkbox on the left to the bound structure controls whether it and the currently selected field are highlighted in the editor window.
After the structure is expanded, all its fields appear on a screen:


For every field of an expanded structure, you see the name, value, address and field type. As the structure may contain fields whose types are also structures, you may continue expanding them until only built-in types are visible.

The data displayed in a "Value" column is not read-only. You may edit any field by double-clicking on it or executing the Edit »Edit Cell command. When you finish editing a field's value, press the Enter key to submit the changes. The changes are then propagated directly to the current document.

## Managing Bound Structures

To bind a new structure, execute the Tools »Structure Viewer »Bind... command. To change the bound structure's properties, double-click it in the list, or select the Edit... option from the shortcut menu.

You may copy the structure's field or the whole structure into the Clipboard using the Copy to Clipboard or Copy All to Clipboard shortcut menu options. Structure may also be exported into the CSV file.

NOTE
Structure viewer limits the amount of data that you can place into the Clipboard. Export to CSV, on the other hand, is capable of processing the entire structure tree, regardless of its complexity and size, provided you have enough free disk space.

To unbind (delete) a bound structure, select it and press the Del key. You may also select the Unbind option from the shortcut menu.
Execute the Tools » Structure Viewer » Unbind All command to remove all bound structures.
The Tools» Structure Viewer » Display Hex Values switch is used to switch between displaying data in decimal or hexadecimal formats.
The Synchronize Position toolbar command tells the Structure Viewer to synchronize cursor movement between the current editor window and the bound structure.
Autorefresh option (on the toolbar) controls whether the Structure Viewer automatically re-binds all currently bound structures whenever a document changes. It is recommended to switch this option off if you have a complex structure bound.

## Working with Schemes

All bound structures may be saved as a scheme. To save a scheme, execute the Tools » Structure Viewer »Save Scheme.... Specify the scheme's name and click the OK button.

All defined schemes are then listed in a scheme combo box.


To load a given scheme, select it in a list. To remove a current scheme execute the Tools » Structure Viewer » Remove Scheme command

## Structure Viewer Settings Page

The Structure Viewer Settings Page allows you to specify the list of include directories and structure-file associations.


## Public Repository Synchronization

HHD Software maintains an online public repository with structure definition files. The editor can automatically synchronize with that public repository.
Press the Reset button to restore the default structure definition files, schemes and associations.
Include Directories

The list of include directories is managed in the left part of the window. Use the Add, Remove and Remove All buttons to manage the list. Include directories are scanned by the \#include directive when it searches for included structure files.

## File Associations

File Associations let you associate a Structure Viewer Scheme with a specific file using regular expressions. Next time a file which path matches the regular expression is opened in the Hex Editor Neo, the given structure scheme is automatically loaded. File associations are managed in the right part of the window.

Use the Add, Remove and Remove All buttons to manage associations.
Language Reference

Hex Editor Neo supports advanced structure definition syntax. It is based on the Standard C type definition syntax and extends it in a number of ways.
While Standard C only allows defining static types, that is, data structures which size and memory allocation is strictly defined at compile time, Hex Editor Neo extends the syntax to allow definition of dynamic types. The following code illustrates this:

```
C++
struct StaticType
    int Array[128];
};
struct DynamicType
    int ArraySize;
    // invalid in Standard C (non const expression), valid in Hex Editor Neo
    int Array[ArraySize / Sizeof(int)]
};
```

The DynamicType structure definition is valid in Hex Editor Neo. The size of the structure is determined at the time it is bound to the data and the number of elements in array array varies.

## Workflow

Hex Editor Neo compiles all protocol definition files in its library on each application start. A preprocessor is run on each source file. It is responsible for the following tasks:

- Elimination of all comments from the source file
- Macro evaluation.
- Processing of conditional compilation blocks
- Processing of all \#include directives and building the dependency graph.

Multiple files are pre-processed and compiled in parallel if Hex Editor Neo is running on multi-processor and/or multi-core computer.

## Tokenization

Source files are tokenized according to standard C language rules: there must be one or more space characters between tokens if they cannot be distinguished without spaces and there may optionally be one or more spaces between tokens if they are distinguishable without spaces.

Space is a space character (' '), tabulation ('\t'), newline (' n ' ) or comment.

C++
int a; $\quad$ // valid, space is used to separate tokens "int" and "a"
inta; $\quad$ // invalid, compiler cannot distinguish between "int" and "a" and parses it as "inta"
int // continued on the next line.
a // still continued...
; // valid, newline is valid space character
a \& b b // valid, space is used to separate "a" and "\&" and "\&" and "b"
b

Space must not be used inside a keyword, such as built-in type int or operator $\& \&$
C++
in t a; // invalid, "int" must not contain space
a || b || invalid, space inside keyword (operator "||")

## Comments

Two standard C-style comments are supported: single-line and multi-line comments. Single-line comment starts with // character sequence and continues to the end of the current line. Multi-line comments must be started with /* sequence and terminated with */ sequence.

## C++ <br> // Single-line comment

/* multi-line
(continued here)
comment */

Multi-line comments may be used "in-place"
C++
struct /* comment */ A
int /* another comment */ a;

Comments are completely ignored and removed from the document prior to compiling it

## Preprocessor

Preprocessor is a special compiler that is run each time the application compiles the protocol definition file. It executes before compilation of the source file and prepares a source file for compilation.

Hex Editor Neo provides a fully C99-compliant preprocessor which supports the following directives:
\#include
Performs a physical inclusion of the contents of another source file into the current file.

## \#pragma once

Prevents a file from being included multiple times.
\#define, \#undef
Allows defining preprocessing constants and macros.
\#if, \#ifdef, \#ifndef, \#else, \#elif, \#endif, defined() operator.
Provides support for conditional compilation.

```
#error
Unconditionally stops source file compilation
```


## \#include directive

```
\#include directive physically includes the contents of a specified file into the current file.
Syntax:
```

```
C++
```

C++
or
C++}\mathrm{ \#include<filename>

```

When used in its first form, the referenced file is searched in the same folder as the current file
For example, let us have following two files:
file1.h
\(\frac{\text { C+ }+ \text { struct } A}{}\)
\(\{\)
\};
And file2.h:
\#++
struct B
A a;
\};
If there was no \#include directive in file2.h, you would not be able to add this file to a Structure Library, as it uses an undefined type A. But preprocessor, which runs on the file before it is compiled, transforms the file into the following:
\(\frac{\text { C++ }}{\text { struct A }}\)
\{truct A
\};
struct \(B\)
A a;
\};
That is, it physically inserts the contents of file1.h into file2.h, thus, making file2.h compilable. See also the \#pragma once directive.
```

Using Absolute and Relative Paths

```

Both syntax forms, form 1 and form 2 allow you to specify absolute or relative paths, for example
\(\stackrel{C+}{1 /}\)
// will use an absolute path
\#include "c:\Projects\definitions.h"
// includes "definitions.h" file, located in "inc" sibling
\#include ". . \(\backslash\) incldefinitions. \(h^{\prime}\)
// includes "definitions.h" file, located in "lib" subdirectory
// of one of standard include paths.
\#include <lib\definitions.h>

\section*{\#pragma once Directive}
\#pragma once directive prevents the file from being included multiple times.
Syntax:
C++
\#pragma once

Let fileA.h be \#include-ed into fileB.h and into fileC.h. If you now write a fileD.h with the following two lines:
C++
\#include "FileB.h
\#include "Filec.h"
"fileA.h" will get included twice. In order to prevent this, put a following line into the "fileA.h" file:

C++
\#pragma once

\section*{\#define Directive}
\#define directive lets you define preprocessor-time constants and macros.
Syntax:
```

C++}\mathrm{ \#define id [token-string]
or
\#define id([id, [id, [...]]) [token-string]

```

A macro declaration must end on the same line. If you need to continue on the next line, finish a line with a backslash character:
```

C++
\#define MY_STRING "This is a start of the long " \
"string" // the end of macro declaration

```

\section*{Defining Constants}

In its first form, \#define creates a preprocessor-time constants. For example:

C++

This declaration will instruct the preprocessor to scan the source file and replace all occurrences of MAX_LENGTH with 5 . The preprocessor is smart enough to only perform a replace when MAX_LENGTH is used as an identifier, that is, it will transform the following code fragment:
```

C++
struct A
{
int array[MAX LENGTH]
const int MaxLength = MAX LENGTH
\$assert(MaxLength == MAX_LENGTH, "Error with MAX_LENGTH")
};
into the following:
C++
struct A
int array[5];
const int MaxLength = 5;
\$assert(MaxLength == 5, "Error with MAX_LENGTH")
};

```

As you see, substitution had not occurred in comment and in string. All other occurrences have been replaced.
A constant may be more complex, for example:
\#define MAX_LENGTH ( \(2+5\) )

Note the use of parenthesis in order to prevent operator precedence errors.
Defined constants may refer to constants defined before:
```

C++
\#define MINUTE (60 * SECOND)

```

The following form
\#define SOME
will only define a constant, without assigning it a particular value. If occurred in a text, it gets removed from it. However, you may successfully test such constant within the \#ifdef directive or using a defined() operator:
```

\#define INCLUDE_STRUCT_A
\#ifdef INCLUDE_STRUCT_A
struct A
{ //.
};

```

By convention, macros and preprocessor constants are named with uppercase letters.

\section*{Defining Macros}

The second form of the directive is used to define macros:
\(\frac{\text { C++ }}{\text { \#define } \operatorname{ADD}(x, y)((x)+(y))}\)

The preprocessor will not only perform a replace, but also will perform a parameter substitution:
```

C++}\operatorname{ADD(7,8) // will be replaced with ((7) + (8))
ADD(n,-1) // will be replaced with ((n) + (-1))

```

\section*{Macros may contain any number of parameters:}
\#define \(\operatorname{MAX}(a, b)((a)>(b)\) ? (a) : (b))

Variadic Macros

Hex Editor Neo supports variadic macros. To use variadic macros, the ellipsis may be specified as the final formal argument in a macro definition, and the replacement identifier _ VA_ARGS_ may be used in the definition to insert the extra arguments. _ VA_ARGS_ is replaced by all of the arguments that match the ellipsis, including commas between them.
\#undef Directive

An \#undef directive is used to remove the macro definition.
Syntax

C++

Example:
\#++ \#define MAX_LENGTH 5
struct A
. int array[MAX_LENGTH];
\};
\#undef MAX_LENGTH
struct \(B\)
(/ compile-time error, "MAX_LENGTH" identifier not found
// (preprocessor did not replace it with "5")
int array[MAX_LENGTH];
\};

\section*{\#error Directive}
\#error directive stops compilation of the current source file.
Syntax:
C++
\#error error-message-string
error-message-string is displayed to the user.

Preprocessor Operators

\section*{\# Stringizing Operator}

The number-sign or "stringizing" operator \# converts macro parameters to string literals without expanding the parameter definition. It is used only with macros that take arguments. If it precedes a formal parameter in the macro definition, the actual argument passed by the macro invocation is enclosed in quotation marks and treated as a string literal. The string literal then replaces each occurrence of a combination of the stringizing operator and formal parameter within the macro definition.

White space preceding the first token of the actual argument and following the last token of the actual argument is ignored. Any white space between the tokens in the actual argument is reduced to a single white space in the resulting string literal. Thus, if a comment occurs between two tokens in the actual argument, it is reduced to a single white space. The resulting string literal is automatically concatenated with any adjacent string literals from which it is separated only by white space.

Further, if a character contained in the argument usually requires an escape sequence when used in a string literal (for example, the quotation mark (") or backslash () character), the necessary escape backslash is automatically inserted before the character.
\(\frac{\text { C++ }}{\text { \#define ASSERT(condition, message) \$assert(condition, message ": " \#condition) }}\)
// ...
struct A
\{ int a
ASSERT(a > 5, "a must be greater than 5");
// The previous line will be replaced with \$assert(a > 5, "a must be greater than \(5^{\prime \prime}\) ": " "a > 5")
\};
\#\# Token-Pasting Operator

The double-number-sign or "token-pasting" operator \#\#, which is sometimes called the "merging" operator, is used in both object-like and function-like macros. It permits separate tokens to be joined into a single token and therefore cannot be the first or last token in the macro definition.

If a formal parameter in a macro definition is preceded or followed by the token-pasting operator, the formal parameter is immediately replaced by the un-expanded actual argument. Macro expansion is not performed on the argument prior to replacement.

Then, each occurrence of the token-pasting operator in token-string is removed, and the tokens preceding and following it are concatenated. The resulting token must be a valid token. If it is, the token is scanned for possible replacement if it represents a macro name. The identifier represents the name by which the concatenated tokens will be known in the program before replacement. Each token represents a token defined elsewhere, either within the program or on the compiler command line. White space preceding or following the operator is optional.

This example illustrates use of token-pasting operator

C++
\#define BIT_FIELD(type, n) type field\#\#n:n
// ...
\(\{\)
BIT_FIELD(unsigned,5); // will be replaced with: unsigned field5:5
BIT_FIELD(unsigned,7); // will be replaced with: unsigned field \(7: 7\);
\};

The \#if directive, with the \#elif, \#else, and \#endif directives, controls compilation of portions of a source file. If the expression you write (after the \#if) has a nonzero value, the line group immediately following the \#if directive is retained in the translation unit.
```

C++
text
[ \#elif expression
text
[\#elif expression
text
[
\#else
text

# \#endif

```

Each \#if directive in a source file must be matched by a closing \#endif directive. Any number of \#elif directives can appear between the \#if and \#endif directives, but at most one \#else directive is allowed. The \#else directive, if present, must be the last directive before \#endif.

The \#if, \#elif, \#else, and \#endif directives can nest in the text portions of other \#if directives. Each nested \#else, \#elif, or \#endif directive belongs to the closest preceding \#if directive

All conditional-compilation directives, such as \#if and \#ifdef, must be matched with closing \#endif directives prior to the end of file; otherwise, an error message is generated. When conditional-compilation directives are contained in include files, they must satisfy the same conditions: There must be no unmatched conditionalcompilation directives at the end of the include file.

Macro replacement is performed within the part of the command line that follows an \#elif command, so a macro call can be used in the expression.
The preprocessor selects one of the given occurrences of text for further processing. A block specified in text can be any sequence of text. It can occupy more than one line. Usually text is program text that has meaning to the compiler or the preprocessor.

The preprocessor processes the selected text and passes it to the compiler. If text contains preprocessor directives, the preprocessor carries out those directives. Only text blocks selected by the preprocessor are compiled

The preprocessor selects a single text item by evaluating the constant expression following each \#if or \#elif directive until it finds a true (nonzero) constant expression. It selects all text (including other preprocessor directives beginning with \#) up to its associated \#elif, \#else, or \#endif.

If all occurrences of constant-expression are false, or if no \#elif directives appear, the preprocessor selects the text block after the \#else clause. If the \#else clause is omitted and all instances of constant-expression in the \#if block are false, no text block is selected.

The constant-expression is an integer constant expression with these additional restrictions:
- Expressions must have integral type and can include only integer constants, character constants, and the defined() operator
- The expression cannot use sizeof() operator.

The preprocessor operator defined can be used in special constant expressions, as shown by the following syntax:

\section*{\(\frac{\text { C++ }}{\text { defined(identifier) }}\)}
defined identifier

This constant expression is considered true (nonzero) if the identifier is currently defined; otherwise, the condition is false (0). An identifier defined as empty text is considered defined. The defined directive can be used in an \#if and an \#elif directive, but nowhere else
```

C++
\#define VARIANT 2
// ..
struct A
{\mp@code{if VARIANT == 1}
char a;
Char a;;
short a;
\#elif VARIANT ==
int a;
long a
\#endif
};

```
ifdef, \#ifndef

The \#ifdef and \#ifndef directives perform the same task as the \#if directive when it is used with defined(identifier).
```

C++
\#ifdef identifien
\#ifndef identifie
// equivalent to
entifier)
\#if !defined(identifier)

```

You can use the \#ifdef and \#ifndef directives anywhere \#if can be used. The \#ifdef identifier statement is equivalent to \#if 1 when identifier has been defined, and it is equivalent to \#if 0 when identifier has not been defined or has been undefined with the \#undef directive. These directives check only for the presence or absence of identifiers defined with \#define.

\section*{Predefined Macros}

Hex Editor Neo defines several macros, called predefined macros. These macros are available for each structure definition file:
\begin{tabular}{ll} 
Macro & Description \\
\hline _SVC_VER & \begin{tabular}{l} 
An integer that specifies the current compiler's version. Low 8 bits represent the minor version, high 8 bits \\
represent major version. For example, version 3.05 is represented as \(0 \times 305\).
\end{tabular} \\
\hline _SVC_X86 & Defined for 32-bit version of Hex Editor Neo. Undefined for 64-bit version. \\
\hline _SVC_X64 & Defined for 64-bit version of Hex Editor Neo. Undefined for 32-bit version. \\
\hline _SVC_POINTER_SIZE & \begin{tabular}{l} 
An integer that specifies the size of pointer in bits on current machine. Equals to 32 for 32-bit version of Hex \\
Editor Neo and 64 for 64-bit version.
\end{tabular} \\
\hline SVC_POINTER_SIZE_REAL & \begin{tabular}{l} 
An integer that specifies the size of pointer in bits on current machine. Always returns the real value, even if 32-bit \\
version of Hex Editor Neo is running on 64-bit OS.
\end{tabular} \\
\hline
\end{tabular}

\section*{Built-in Types \\ Integer Types}

Hex Editor Neo natively supports the following integer types:
\begin{tabular}{|c|c|c|c|c|}
\hline Type Name & Other Valid Names & Typedefs in stddefs.h & Description & Range \\
\hline bool & N/A & N/A & One-byte boolean value. Displayed as "true" and "false". & N/A \\
\hline char & signed char & CHAR, int8, _int8 & 8-bit signed character & \[
\begin{aligned}
& -2^{7} \text { to } 2^{7}- \\
& 1
\end{aligned}
\] \\
\hline wchar_t & N/A & WCHAR & 16-bit UNICODE character & 0 to 216-1 \\
\hline unsigned char & N/A & UCHAR, BYTE, uint8 & 8-bit unsigned character & 0 to 28-1 \\
\hline short & signed short, short int, signed short int & SHORT, int16, __int16 & 16-bit signed integer & \[
\begin{aligned}
& -2^{15} \text { to } 2^{15} \\
& -1
\end{aligned}
\] \\
\hline unsigned short & unsigned short int & USHORT, WORD, uint16 & 16-bit unsigned integer & 0 to \(2^{16-1}\) \\
\hline int & signed int, signed & INT, int32, __int32 & 32-bit signed integer & \[
\begin{aligned}
& -2^{31} \text { to } 2^{31} \\
& -1
\end{aligned}
\] \\
\hline unsigned int & unsigned & UINT, uint32 & 32-bit unsigned integer & 0 to 232-1 \\
\hline long & signed long, long int, signed long int & LONG & 32-bit signed integer & \[
\begin{aligned}
& -2^{31} \text { to } 2^{31} \\
& -1
\end{aligned}
\] \\
\hline unsigned long & unsigned long int & ULONG, DWORD & 32-bit unsigned integer & 0 to \(2^{32-1}\) \\
\hline _int64 & long long, signed long long & LONGLONG, int64 & 64-bit signed integer & \[
\begin{aligned}
& -263 \text { to } 263 \\
& -1
\end{aligned}
\] \\
\hline unsigned __int64 & unsigned long long & ULONGLONG, FILETIME, uint64 & 64-bit unsigned integer & 0 to 264-1 \\
\hline
\end{tabular}

\section*{Type Modifiers}

Two modifiers, little_endian and big_endian may be used to change the default byte ordering of a type. They may be used directly in declaring fields:
```

C++
struct A
big_endian short value;
};
or in typedef declaration to create a new type:
C++ $\quad$ typedef big_endian short b_short;

```

Note that \#pragma byte_order directive may still be used to change default byte ordering for the rest of a scope. If little_endian or big_endian modifier is present, it always overwrites the current default.

\section*{Floating-Point Types}

Hex Editor Neo natively supports the following floating-point types:
\begin{tabular}{llll} 
Type Name & Description & Size, in Bytes & Range \\
\hline float & Single-precision floating-point type, according to IEEE 754-1985 & 4 & \(3.4 \cdot 10 \pm 38(7\) digits) \\
double & Double-precision floating-point type, according to IEEE 754-1985 & 8 & \(1.7 \cdot 10 \pm 308(15\) digits \()\)
\end{tabular}

\section*{String Types}

Hex Editor Neo natively supports the following string types:
\begin{tabular}{ll} 
Type Name & Description \\
\hline string & Null-terminated ANSI string (each character occupies single byte). \\
\hline wstring & Null-terminated UNICODE string (each character occupies two bytes).
\end{tabular}

\section*{Expressions}

Expressions are allowed in different places in the structure definition. For example, the size of the bit field, the number of items in array and pointer offset are all specified using expressions. Expressions are also used in calculating constant values and enumeration values.

Expression is a combination of immediate values, constants, enumeration values, function calls and field references. All these elements are connected with one or more operators.

An expression may be as simple as
```

C++
or as complex as:
C++
info.bmiHeader.sel.header.biSizeImage // take a value of a field several scopes deep
bfTypeAndSignature.bfType == 'BM' \&\& bfReserved1 == 0 \&\& bfReserved2 == 0 // verify a condition
RvaToVa(OptionalHeader.DataDirectory[i].VirtualAddress) // access a field in a nested structure and array and call an external function

```

\section*{Operators}

Below is a table of supported operators. Operators are sorted by their precedence, from highest to lowest. Operators in the same row have the same precedence value and are evaluated from left to right
\begin{tabular}{|c|c|}
\hline Operator(s) & Name or Meaning \\
\hline - [] () & Field access, array indexing, expression grouping \\
\hline () & Function call \\
\hline - ~ ! sizeof() \& & Unary minus, bitwise NOT, logical NOT, sizeof, address-of \\
\hline * / \% & Multiplication, division, modulo division \\
\hline + - & Addition, subtraction \\
\hline << >\gg>> & Left shift, right shift, right unsigned shift \\
\hline \ll= \gg \(=\) & Less than, less than or equal, greater than, greater than or equal \\
\hline == ! \(=\) & Equality, inequality \\
\hline \& & Bitwise AND \\
\hline \(\wedge\) & Bitwise XOR \\
\hline 1 & Bitwise OR \\
\hline \& \& & Logical AND \\
\hline 11 & Logical OR \\
\hline ?: & Conditional operator \\
\hline
\end{tabular}

\section*{ptimization}

All expressions are evaluated at the time a structure file is compiled. If expression is successfully evaluated to constant value (that is, it does not contain any field references), calculated value is used instead of the expression. It is used each time a type is bound to the data, thus greatly minimizing bind time. You can get advantage of this optimization taken by the Hex Editor Neo: instead of
```

const SecondsInHour = 3600; // 60 * 60
const SecondsInDay = 86400; // 60 * 60 * 24
use
C++
const SecondsInDay = SecondsInHour * 24;

```

As the result will be the same after the source file is compiled.
Hex Editor Neo is also capable of optimizing sub-expressions:
```

C++
{ int size
byte data[size * (sizeof(int) - 1)];
};

```
mportant note here is that constant sub-expression must be enclosed in parenthesis in order to be optimized. Otherwise, Hex Editor Neo will not optimize it because it considers a variable as having arbitrary type C++
// will be optimized to \(\mathrm{i}+\)
var \(j=i+5+3\); // will not be optimized (consider the case when \(i\) is a string, for example, which results in \(i+\) " 53 ").

\section*{mmediat}

An expression may contain an immediate value. There are several ways to specify an immediate:

\section*{nteger Number}

A signed or unsigned integer number may be specified in either base 10 or base 16 :

```

-7 // integer number -
0x5a // integer number 90

```

\section*{Floating-Point Number}

A floating-point number is specified in either standard or scientific notation:
\(\frac{2.3}{2.3} / /\) floating-point number 2.3
\(-4 e-9\) // floating-point number -0.000000004

Character Constants

A character constant is one, two or four symbols enclosed in single quotation marks:
```

C+++
// integer number 19778
'NTFS' // integer number 1397118030

```

A standard \(C\) escape character may be used inside the quotation marks:
```

// integer number 1
\t' // integer number 9

```
String Constants

String constants are only allowed in \$assert directive and in \$print directive. A sequence of characters (including escape characters) enclosed in double quotation marks. Several subsequent strings are automatically concatenated.
```

"Hello, World!" // "Hello, World!" string
"Hello,"" "World!" // the same "Hello, World!" string (automatically concatenated)
"Col1\tCol2\tCol3\n1\t2\t3" // simple table with two rows and three columns

```

\section*{References}

Hex Editor Neo allows getting a reference to a field by using the built-in function ref. The following operators are allowed for two references:
- == operator
- != operator
- < operator
- <= operator
- > operator
- >= operator

If one argument of a binary operator is a reference and another one is not, a reference is automatically dereferenced and then the normal operator rules apply.
In addition, a reference may be dereferenced using one of the built-in conversion functions.

\section*{Limitations}

The result of ref function application may not be used with [] operator and . operator. Assign the result of the function to the local variable and use the variable instead
```

C++
struct
{
struct { int a,b; } field;
var t1 = ref(field).a; // syntax erron
var r_field =ref(field); // ok
int array[100];
var t3 = ref(array)[50]; // syntax error
var r_array = ref(array); // ок
};

```

\section*{Byte Array}

Byte arrays are introduced in version 5.15. This is a new value type in addition to previously supported boolean, integer, floating-point, string and reference value types Byte arrays represent contiguous ranges in memory and usually contain unstructured data. The following operations are supported on byte arrays:
- References are implicitly converted to byte arrays if they are used in the expression with other byte arrays. See example below.
- Byte arrays may be specified as immediate values using the following syntax:
```

C++
struct Header
unsigned char Signature[3];
\$assert(Signature == { 0x20, 0xff, 0xfe }); // Immediate byte array
// Note the implicit conversion between the array field and byte array,
// Array field is first automatically converted to reference and then implicitly converted to
// byte array
};

```
- Other value types may be explicitly converted to byte array using the array() function.
- Arrays may be compared for equality and inequality. Arrays also support the following operators: \(\langle\rangle,,\langle=\) and \(\rangle=\) for which lexicographical comparison is used.
- An individual element of byte array may be taken using the element() function.
- A portion of an array may be extracted using the subarray() function.
- A length() function returns the number of bytes in an array. This function also works with strings, returning the length of the string.
- A find() function allows you to search for an occurrence of one byte array within another one. This function also works with strings

\section*{Field Access}

An expression may reference any field within visible scope. A field is referenced by its name, which is case sensitive. Name lookup continues from child scope to parent scope until the match is found. If the field is not found, constants and enumeration values are searched. If no match is found in constants and enumeration values, an error is

\section*{generated.}

The following example illustrates this:
```

C++
EnumerationValue =0\times20
};
const ConstantValue = 0x30;
struct A
{ int a
int Array1[a]; // references a in this structure
int Array2[ConstantValue]; // references ConstantValue in global scope
int Array3[EnumerationValue]; // references EnumerationValue in global scope
struct
int Array4[a]; // references a in parent scope
} contained;
};

```

Pointed to types are allowed to reference fields in pointer field's scope. It is their direct enclosing scope. For example:
```

C++
struct PointedType; // forward declaration
struct PointerType
int a;
int b as PointedType *; // b is converted to a pointer to PointedType structure
};
struct PointedType
int Array[a + c]; // references a and c in PointerType (note c is declared AFTER b, the reference is still valid)
};
this Pseudo-Field

```

The special field this is defined for each bound user-defined type and evaluates to the absolute offset of this bound structure.
```

C++
//.
};
struct A
int OffsetToB as B *(this); // B will automatically be bound to \&A + OffsetToB
};

```

When passed to the built-in ref function, the result is a reference to the current object.

This built-in variable evaluates to the current array element index, if the structure is being used inside an array, otherwise it is evaluated to -1 .
C++
struct B;
struct A
B b[10];
\};
struct B
int a;
if (array_index == 3) // array_index evaluates to a current array index
\};

This built-in variable evaluates to the address the next field will be bound to.

C++ \(\#\) pragma script("get_doc_size.js") // we use a GetDocumentSize() function
struct A
र int \(\mathrm{b}[10]\);
// pad this structure to the end of the file
char padding[GetDocumentSize() - current_offset]
\};

Field Access Operato

Field access operator is used to access fields in contained scopes
Syntax:

C++
C++.id[.id[.id ...] ]
The following code fragment illustrates the use of the field access operator:
```

C++
struct A
{ struct
{truct
int c;
} b;
int Array[b.c]; // access the c field inside the b field
};

```
struct, union, case union or pointer field may be used on the left of the field access operator.

\section*{[] Array Indexing Operator}

Array indexing operator allows you to reference individual array items.
Syntax:
\(\frac{\mathrm{C}++}{\mathrm{id}[\exp ]}\)

The following code fragment illustrates the use of the array indexing operator:
\(\frac{\text { C++ }}{\text { stru }}\)
struct A
int Array1[10]; // static array of 10 integers
int Array2[Array1[2]]; // dynamic array, which size is taken from the third value of Array1 array
\};
Array elements are numbered from 0 to Count -1 , where Count is (static or dynamic) array size
() Expression Grouping Operator

Expression grouping operator allows you to change the precedence rules of operators in an expression.
Syntax:
\(\frac{\mathrm{C}++}{(\exp )}\)

The following code fragment illustrates the use of the expression grouping operator:
```

C++

```

You can use parenthesis whenever you need to change the precedence of operators or increase the readability of the expression.

\section*{() Function Call Operator}

Function call operator allows you to call an internal or external function.
Syntax:
```

C++

```
param-list: expr [,expr , [...] ]

The following code fragment illustrates the use of the function call operator:
C++
\({ }_{\text {struct } A}^{\text {C }}\)
\[
\begin{aligned}
& \text { int Array1[int(10.2)]; } \quad / / \text { Call an internal cast function } \\
& \text { int Array2[ExternalFunction()]; // Call an external function with no parameters }
\end{aligned}
\]
\};
External function calls are never optimized at compile-time. All expressions containing external function calls will be computed at run-time.

\section*{Unary Minus Operator}

Unary minus operator change the sign of the expression.


The following code fragment illustrates the use of the bitwise NOT operator:
```

C++ // evaluates to -8

```

This operator is not applicable to floating-point values
\& Bitwise AND Operato

This operator performs a bit-wise AND operation on its operands.
```

Syntax:
C++
Usage:

```
```

C++

```
```

C++

```

This operator is not applicable to floating-point values.

\section*{^ Bitwise XOR Operator}

This operator performs a bit-wise XOR (eXclusive OR) operation on its operands
Syntax:
\(\frac{c_{++}}{\exp 1 ~ \wedge ~ e x p 2 ~}\)

Usage:
\(\begin{array}{ll}\frac{C++}{7} \text { ^ } 8 & \text { // evaluates to } 15 \\ 2 \wedge 2 & \text { // evaluates to } 0\end{array}\)

This operator is not applicable to floating-point values

\section*{Bitwise OR Operator}

This operator performs a bit-wise OR operation on its operands.
Syntax:
exp1 | exp2
Usage:
```

| C++ | // evaluates to 15 |
| :--- | :--- |
| 7 | 8 |
| 2 | 4 |

```
his operator is not applicable to floating-point values

\section*{Logical NOT Operato}

Logical NOT operator inverts (logically) an expression. Evaluates to 0 if the expression is non-zero or to non-zero if the expression is zero
Syntax:
\(\frac{\text { C++ }}{!} \exp\)

The following code fragment illustrates the use of the logical NOT operator:
```

C++

```

\section*{\&\& Logical AND Operator}

This operator evaluates to non-zero if and only if both operands are non-zero, otherwise it evaluates to zero.
Syntax: \(\stackrel{C++}{\exp 1 \& \& \exp 2}\)

Usage:
\begin{tabular}{ll} 
C++ & \\
\hline 7 \&\& 8 & |/ evaluates to 1 \\
\(0 \& \& 2\) & |/ evaluates to 0 \\
\(5==5 \& \& 2==2\) & |/ evaluates to 1
\end{tabular}
If \(\exp 1\) evaluates to zero, \(\exp 2\) is not evaluated

\section*{| Logical OR Operato}

This operator evaluates to non-zero if at least one of if its operands is non-zero, otherwise it evaluates to zero


If \(\exp 1\) evaluates to a non-zero value, \(\exp 2\) is not evaluated.

\section*{sizeof() Operator}

This operator returns the size, in bytes, of the enclosed identifier.
Syntax:
C++
sizeof(id)
id may be either a built-in type, typedef'ed type, user-defined type or field reference. Taking the size of user-defined dynamic type is incorrect and will result in compile error.
The following code fragment illustrates the use of the sizeof() operator:

\section*{\(\frac{\mathrm{C}++}{\text { struct }}\)}
\{ int a
int b;
int \(c[4]\)
\};
struct B
int \(a ;\)
int \(b[a]\)
\};
struct C
B b;
const SizeOfInt = sizeof(int);
const SizefA \(=\) sizeof(int
const SizeOfA \(=\operatorname{sizeof}(A)\);
const SizeOfB
var SizeOfb = sizeof(b);
\};
// correct, evaluates to 4
// correct, evaluates to 24 (size of static user-defined type A)
// incorrect, results in compile-time error, B is dynamic type
// correct, will be calculated at run-time
\& Address-Of Operator
This operator takes the address (offset) of the given field.
Syntax:
\(\frac{\mathrm{C}++}{\& f \text { ield-id }}\)
Returned address is absolute.
\(\frac{\mathrm{C}++}{\text { struc }}\)
\(\frac{c++}{\text { struct A }}\)
int a;
const OffsetTob \(=\& \mathrm{~b}\) - this; // offset to field b (in bytes) from the beginning of the
\};

\section*{* Multiplication Operator}

This operator computes the multiplication of two given expressions.
Syntax:
\(\xrightarrow[\text { exp1 }]{\text { C+ }} \exp 2\)
Usage:
\(\frac{\text { C++ }}{7 \times 8} \quad\) // evaluates to 56
\((2+2) *(7-1) \quad / /\) evaluates to 24

\section*{Division Operator}

This operator computes the result of division of one operand by another.
Syntax:
\(\frac{\text { C++ }}{\exp 1 / \exp 2}\)
If the divisor evaluates to zero, a run-time error occurs. If the result of the expression is used as an integer value, the result of the division is truncated to the nearest integer value.
```

C++

| $4 / 2$ | // evaluates to |
| :--- | :--- |
| $3 / 2$ |  |

5 / 0 // a run-time error occurs

```

\section*{\% Modulo Division Operator}

This operator computes the remainder of the division of the first operand by the second operand
Syntax:
\(\xrightarrow{\text { C++ }} \operatorname{exp1} \% \exp 2\)
If the divisor evaluates to zero, a run-time error occurs.
```

C++
5% 2 // evaluates to
5% 0 // a run-time error occurs

```

This operator is not applicable to floating-point values.

Addition Operato

This operator computes the sum of two given expressions.
Syntax:
\(\frac{\mathrm{C}++}{\exp 1+\exp 2}\)

Usage:
\begin{tabular}{ll} 
C++ & \\
\hline \(7+8\) & // evaluates to 16 \\
\(2+2\) & // evaluates to 4
\end{tabular}

Addition operator may also be used in string expressions to concatenate strings or to add immediate or field to a string.
Subtraction Operator

This operator computes the difference between two given expressions.
Syntax:
C++

Usage:
```

C++}7-8 // evaluates to -1

```
<< Left Shift Operator

This operator performs a left shift of the first operand. The second operand specifies how many bits to shift.
Syntax:
\(\frac{\operatorname{c}++}{\exp 1} \ll \exp 2\)

Usage:
\(7 \ll 1 \quad\) // evaluates to 14
\(-3 \ll 3\) // evaluates to -24
This operator is not applicable to floating-point values
>> Right Shift Operator
This operator performs an arithmetic right shift of the first operand. The second operand specifies how many bits to shift.
Syntax:
\(\frac{\text { exp1 }}{\text { c+ }}\) exp2

Usage:
```

C++
-100 >> 2 // evaluates to -25

```

This operator is not applicable to floating-point values.
>> Right Unsigned Shift Operator
This operator performs a logical (unsigned) right shift of the first operand. The second operand specifies how many bits to shift
\begin{tabular}{|c|c|}
\hline \multicolumn{2}{|l|}{Syntax:} \\
\hline \multicolumn{2}{|l|}{C++} \\
\hline \multicolumn{2}{|l|}{exp1 >>> exp2} \\
\hline \multicolumn{2}{|l|}{Usage:} \\
\hline \multicolumn{2}{|l|}{C++} \\
\hline 456 >>> 3 & // evaluates to 57 \\
\hline -100 >>> 2 & // evaluates to 1073741799 (0x3fffffe7) \\
\hline
\end{tabular}

This operator is not applicable to floating-point values.

\section*{< Less Than Operator}

This operator evaluates to non-zero value if first operand is less than the second operand or to zero value otherwise.
Syntax:
\(\underset{\exp 1}{\text { c+ }}<\exp ^{2}\)

Usage:
\begin{tabular}{ll}
\(\frac{C++}{7<8}\) & // evaluates to 1 \\
\(2<2\) & // evaluates to 0
\end{tabular}
<= Less Than or Equal Operator

This operator evaluates to non-zero value if first operand is less than or equal to the second operand or to zero value otherwise
Syntax:
\(\exp 1<=\exp 2\)
Usage:
\begin{tabular}{ll} 
C++ & \\
\(7<=8\) & // evaluates to 1 \\
\(2<=2\) & // evaluates to 1 \\
\(3<=2\) & // evaluates to 0
\end{tabular}

\section*{\(>\) Greater Than Operator}

This operator evaluates to non-zero value if first operand is greater than the second operand or to zero value otherwise
Syntax:
\(\xrightarrow{\text { C++ }} \operatorname{exp1~>exp2~}\)

Usage:
\begin{tabular}{ll} 
C++ & \\
\hline \(8>7\) & I/ evaluates to 1 \\
\(2>2\) & // evaluates to 0 \\
\(3>5\) & // evaluates to 0
\end{tabular}
\(>=\) Greater Than or Equal Operator
This operator evaluates to non-zero value if first operand is greater than or equal to the second operand or to zero value otherwise
\begin{tabular}{|c|c|}
\hline \multicolumn{2}{|l|}{Syntax:} \\
\hline \multicolumn{2}{|l|}{C++} \\
\hline \multicolumn{2}{|l|}{\(\exp 1>=\exp 2\)} \\
\hline \multicolumn{2}{|l|}{Usage:} \\
\hline \multicolumn{2}{|l|}{C++} \\
\hline \(8>=7\) & // evaluates to 1 \\
\hline \(2>=2\) & // evaluates to 1 \\
\hline \(3>=5\) & // evaluates to 0 \\
\hline
\end{tabular}
\(=\) Equality Operator

Evaluates to non-zero value if both operands are equal or to zero otherwise.
Syntax
\(\frac{\mathrm{C}++}{\exp 1}==\exp 2\)

Usage
\(\begin{array}{ll}\text { C }++ & \text { // evaluates to } 0 \\ 7==8 & \text { / } \begin{array}{l}\text { evaluates to }\end{array} \\ 2==2 & \end{array}\)

Evaluates to non-zero value if operands are different or to zero otherwise.
\begin{tabular}{|c|c|}
\hline \multicolumn{2}{|l|}{Syntax:} \\
\hline \multicolumn{2}{|l|}{C++} \\
\hline \multicolumn{2}{|l|}{exp1 ! \(=\exp 2\)} \\
\hline \multicolumn{2}{|l|}{Usage:} \\
\hline \multicolumn{2}{|l|}{C++} \\
\hline 7! \({ }^{\text {a }}\) & // evaluates to 1 \\
\hline \(2!=2\) & // evaluates to 0 \\
\hline
\end{tabular}

\section*{?: Conditional Operator}

This is the only supported ternary operator. It evaluates to the value of its second operand if the first operand evaluates to non-zero value, otherwise, it evaluates to the value of the third operand.
Syntax:
\begin{tabular}{l} 
C++ \\
exp1 ? exp2 : exp3 \\
Usage: \\
\begin{tabular}{ll} 
C++ & // evaluates to 7 \\
\(5=3\) ? \(8: 7\) & // evaluates to 8 \\
\(2==2 ? 8: 7\) &
\end{tabular}
\end{tabular}.

\section*{Functions}

Internal Functions

Built-In Functions

Hex Editor Neo provides a number of built-in functions:
\begin{tabular}{|c|c|}
\hline Function Name & Description \\
\hline \multicolumn{2}{|l|}{\multirow[t]{2}{*}{bool(exp)}} \\
\hline & \\
\hline & Value's type Behavior \\
\hline & bool exp returned \\
\hline & int false if exp equals to zero, true otherwise \\
\hline & double false if exp equals to zero, true otherwise \\
\hline & string false if exp is an empty string, true otherwise \\
\hline & reference false if reference is not initialized, true otherwise \\
\hline & Note that when a statement expects a boolean, it converts the passed value to boolean automatically. The same rules are used during this implicit conversion. \\
\hline int(exp) & Convert a given parameter exp to an integer value. Floating-point numbers are truncated to nearest integer, strings are parsed as containing an integer. If parsing error occurs, the result of the conversion is zero. If passed string starts with \(\theta\), it is considered as containing octal number. If passed string starts with \(0 x\) or \(0 x\), it is considered as containing hexadecimal number. References are first dereferenced and then converted to integer. \\
\hline double(exp) & Convert a given parameter exp to a floating-point value. If used on string, a string is parsed as containing a floating-point number. If parsing error occurs, the result of the conversion is zero. References are first dereferenced and then converted to floating-point. \\
\hline string(exp) & Convert a given parameter exp to a string value. References are dereferenced. \\
\hline decimal(exp) & Convert a given parameter exp to an integer value. Always treats a passed string as containing decimal number (regardless of any prefixes). References are dereferenced. \\
\hline hex(exp) & Convert a given parameter exp to an integer value. Always treats a passed string as containing hexadecimal number (regardless of any prefixes). References are dereferenced. \\
\hline octal(exp) & Convert a given parameter exp to an integer value. Always treats a passed string as containing octal number (regardless of any prefixes). References are dereferenced. \\
\hline binary (exp) & Convert a given parameter exp to an integer value. Always treats a passed string as containing binary number (regardless of any prefixes). References are dereferenced. \\
\hline integer_convert(exp, radix) & Convert a given parameter exp to an integer value. An integer's base is passed as second argument to the function. It is automatically converted to integer. References are dereferenced. \\
\hline format(fmt_string, ...) & This function takes a format string fmt_string and arbitrary number of arguments. It returns a string after placing each passed parameter to corresponding placeholder in a format string. \\
\hline substring(string, pos, [count]) & This function extracts a substring from a given string. pos is zero-based index of the first character of the substring and count is an optional number of characters in a substring. If omitted, substring continues until the end of the string. If pos is greater than the string length, an exception occurs. Count can be any positive integer or -1 , which is equivalent to omitting the parameter. \\
\hline subarray(array, start_offset[, length]) & This function extracts a sub-array of a byte array. First and second arguments are required. array is an array from which you are extracting a sub-array and start_offset is a zero-based offset of the beginning of sub-array. Optional length parameter specifies the length of the resulting sub-array in bytes. \\
\hline element(array, position) & Returns the byte at a given offset in an array byte array. \\
\hline length(exp) & Return a length of a string if exp is a string expression, or byte array (in bytes), if exp is a byte array. \\
\hline
\end{tabular}
\begin{tabular}{|c|c|}
\hline Function Name & Description \\
\hline find(find_where, find_what[, start_pos) & Find a substring in a string (if both find_where and find_what expressions are strings) or one array within another, if both expressions are byte arrays. Optional start_pos parameter specifies the location from which to start searching. It defaults to 0 . The function returns integer specifying the found location or -1 if no occurrence is found. \\
\hline \multirow[t]{7}{*}{type(exp)} & Determine the type of the expression exp. Returns one of the following values (symbolic names are predefined in stddefs.h): \\
\hline & Symbolic
Constant \\
\hline & BooleanType 0 bool \\
\hline & IntegerType 1 int \\
\hline & FloatingPointType 2 double \\
\hline & StringType 3 string \\
\hline & ReferenceType 4 reference \\
\hline \multirow[t]{2}{*}{ref(exp)} & Take the reference of expression. Expression must be a Field Access or Array Indexing Operator. \\
\hline & A special construct ref(this) is used to get a reference to the current object. \\
\hline array (exp) & Convert an expression to a byte array. Little-endian representation of boolean, integer and floating-point values are returned. If exp is a reference, a copy of referenced value is returned as byte array. All other conversions are prohibited. \\
\hline is_valid(exp) & This function evaluates the expression and returns false if any exception occurs during evaluation, otherwise, it returns true. Expression's result is never used and is silently discarded. \\
\hline \multirow[t]{4}{*}{evaluate_if(exp1, exp2)} & This function evaluates expression exp1 and returns its result. If any exception occurs during evaluation, it returns the value of expression exp2. Note that if another exception occurs during evaluation of exp2, it is handled as usual (propagated to user interface). This function is equivalent to \\
\hline & c++ \\
\hline & is_valid(exp1) ? exp1 : exp2 \\
\hline & but evaluates exp1 only once. \\
\hline
\end{tabular}


\section*{Native Functions}

Hex Editor Neo supports native functions. A native function is a function defined in the header file that uses the same language syntax.
The following syntax is used to define a function:
```

Op+
function f_name ( parameter-list
{
ion-body
}
optional-attributes := [ function-attribute [, function-attribute [, ...] ] ]
function-attribute := arguments_array | nooptimize
parameter-list := param-name1 [, param-name2 [, ... ] ]
function-body := *(variable-declaration | statement)

```

The function may be declared at any scope (either global or user-defined). Function's name must be unique within a scope. A function can take any number of arguments (or no arguments). It is allowed to pass different number of arguments during a function call. If fewer arguments passed, the rest are undefined, if more arguments passed, extra parameters are ignored.

Only the following constructs are allowed in a function body:
- A declaration of constants or constant arrays
- A declaration of variables or variable arrays.
- Any supported statement, including expression calculation statement

Ability to use any statement allows you to create branches and loops. A return statement sets function's return value and exits function. A function is allowed to have multiple return statements.

A following pseudo-variables are available for a function's body:
parameter_count
Holds the actual number of passed parameters

\section*{arguments_array}

Holds values of all passed parameters. The size of the array is parameter_count. If function has named arguments, both named parameters and values of this array may be used to reference passed parameters.
Note: for performance reasons this array is generated only when arguments_array attribute is specified before the function's definition.

All parameters are always passed by value. If arguments_array attribute is specified and a function has named arguments, corresponding array elements and named parameters have copies of values (that is, modifying one does not modify another)

\section*{Function Scope}

Each function defines its own scope. All variables declared in this scope are destroyed when function execution finishes and re-created next time it is run.

Function Optimization

Hex Editor Neo automatically optimizes constant functions. If it encounters a function that always returns a constant value and is being used with constant arguments, it replaces the function call with a calculated value. That means that all native functions must be immutable, that is, they must not affect any global state.

If you still need to use a function that modifies a global state, use the nooptimize attribute to suppress function optimization.
Note that function optimization always works at the call site. Consider the following example:
```

C++
function square(x)
return x* x;
}
struct A
char array[square(5)]; // will be optimized to 25
int size;
char array2[square(size)]; // will not be optimized

```
\};

\section*{External Functions}

\section*{Attaching Scripts}

External functions referenced in your structure definition files must be defined in separate files. Hex Editor Neo supports functions written in JavaScript.
Use the following syntax to attach a script file to the structure definition file:
\#pragma script(path-to-script-file)
Path-to-script-file is a string containing the absolute or relative (to the structure definition file) path to a script file. Only JavaScript external files (with any extension) are supported.

As with included files, Hex Editor Neo automatically rescans a file if it is modified outside the editor.

\section*{Examples}
functions.js
\(\frac{\text { C++ }}{\text { function } f()}\)
\{ return 10;
\}
structure.h:

C++ \#pragma script("functions.js")
public struct A
char array \([f()]\)
\};
javascript Keyword

In addition, JavaScript code may be specified in the structure definition file using the javascript keyword:
C++
javascript
function \(f()\)
\{ return 10;
\}; \({ }^{\}}\)
public struct A
\{ public struct \(A\)
char \(\operatorname{array}[f()] ;\)

The number of javascript blocks is not limited. All blocks are processed before any structure is bound, so all functions declared in these blocks are always visible to any structure, regardless of the place where you define them

\section*{External Functions}

Hex Editor Neo allows you to use external functions in expressions. External functions are defined in script files, attached to structure definition files using the \#pragma script preprocessor directive

An external function accepts zero or more parameters and must always return a value ("void" functions, or procedures are not supported). External function must be written in JavaScript.

Hex Editor Neo automatically performs parameter type conversion when the function call is made. Although, make sure the value returned by the function is of correct type, as

Hex Editor Neo expects a value of a given type in several places, such as in array declaration:
```

functions.js:
C++
function GetArraySize()
{ return 11/2
}
structure.h:
\#pragma script("functions.js")
struct A

```

```

};

```

\section*{Script Objects}

Hex Editor Neo provides scripts running as part of expression evaluation with two built-in objects:
\begin{tabular}{l}
\begin{tabular}{l} 
Object \\
Name
\end{tabular} \\
\hline description \\
parser \(\quad\)\begin{tabular}{l} 
Allows script to reference other bound structures, their fields, as well as bind other structures to the current document and \\
more information. \\
display debugging messages. This object is a default "global" object for a script, so its methods may be called directly, without \\
using parser.Method notation. Refer to the Working with Parser Object section for more information.
\end{tabular} \\
\hline
\end{tabular}

Example
```

C++

```
\{ return document. Filesize
\}

\section*{Statements}

Statements control the flow of execution. They are allowed on user-defined type scope and in the function body. This section describes all statements supported by the Hex Editor Neo.

The general statement syntax may be described by the following grammar
```

C++
statement-if | statement-switch | statement-break | statement-while | statement-for | statement-dowhile | statement-return
statement-block:

```
    statement | field-declaration

Where field-declaration is either standard or user-defined type, or structure definition.
See the following topics for more information:
- if statement
- switch statement
- break statement
- while statement
- for statement
- do...while statement
- return statement

\section*{f Statement}
if statement has the following syntax:
```

statement-if
if (expr)
statement-block | { statement-block * }
statement-block | { statement-block * }
]
statement-block
statement | field-declaration

```
expr is evaluated at run-time and if true, the first statement-block is evaluated, otherwise, the second statement-block is evaluated. If it is omitted and expr is false, nothing is evaluated and control flows to the next statement or declaration. If multiple statements or declarations need to be specified in the body of if or else, use the curly braces. For example:
```

C+
struct
int
if (a>0)
float b;
else
double b;
int c;
}
};

```

\section*{switch Statement}
switch statement has the following syntax.
```

statement-switch:
switch (sw-expr)
case const-case-expr1:
[case const-case expr2
statement-block *
`]
default
statement-block +
} ]

```

WARNING
switch statement's block must not be followed by a ';' character!
switch statement is evaluated according to the following procedure:
1. All const-case-exprn expressions are evaluated at compile time. If they fail to compute to a constant value, compilation error occurs. If you need to use non-const expressions, consider using the case union
2. sw-expr is evaluated at run time
3. The resulting value is compared with each const-case-exprN value one by one until a match is found. If the match is not found, statement-block after the "default:" label is evaluated, if present.
4. If the match is found, all statement blocks after the corresponding "case" label are evaluated, until the "default:" label, end of switch statement or a break statement are met.

C++
struct A
BYTE type;
switch (type)
case 0 :
int value;
break
case 1 :
double value
break;
case 2 :
string value;
break;
default:
\$assert("Invalid file")
\} // note: no ';' allowed here!

\section*{break Statemen}
break statement has the following syntax:
```

statement-break
break;

```

NOTE
The ' \(;\) ' character at the end of the statement is mandatory.

\section*{The break statement may be used}
- Inside the switch statement to stop statement evaluation
- Inside the for statement, while statement or do...while statement to cancel a loop.

In addition, break statement always ends the current scope, even if it is used outside of the switch or loops

\section*{wile Statement}
while statement has the following syntax:
```

statement-while
while (expr)
statement-block

```

Evaluates statements and declarations in statement-block until expr becomes false. If expression is false at the first iteration, no statements are evaluated.
\(\frac{\mathrm{C}++}{\mathrm{var}} \mathrm{i}=10 ;\)
while (i)
    int \(a ;\)
\(i=i\)
\(i=i-1 ;\)

\section*{or Statemen}
for statement has the following syntax:
```

statement-for:
for (init-expr; condition-expr; increment-expr)
statement-block

```
for statement is equivalent to the following construct:
\(\frac{\text { C++ }}{\text { init-expr; }}\)
while (condition-expr)
statement-block
increment-expr;
\}

It evaluates statements in statement-block while condition-expr evaluates to a non-zero value
Note that unlike C/C++, all for statement expressions must be present and cannot be omitted. Hex Editor Neo does not support unary operators like ++ , -- , \(+=\), \(-=\) and others, so increments must be specified in the form of \(i=i+1\) instead of \(++i\).
init-expr must have the following syntax
C++
var name \(=\) exp

For example
\(\frac{\text { C+ }}{\text { for }}(\) var \(i=0 ; i<10 ; i=i+1)\)
\{ int a
\}
do...while Statement
do...while statement has the following syntax:
statement-dowhile:
do statement-block
while (expr);
Evaluates statements and declarations in statement-block until expr becomes false. If expression is false at the first iteration, statements in statement block are evaluated exactly one time.
\(\xrightarrow{\text { C++ }}\) var \(\quad\) =10;
var
do
\{
int a;
\} while \(\underset{(i)}{i}=1\) i

\section*{return Statemen}
return statement has the following syntax:
return expr;

Evaluates the passed expression and sets the current native function's return value. It also exits the current function. If used outside the function body, an exception is generated.

C++
function square( x )
return \(\mathrm{x} * \mathrm{x}\);
\}

\section*{Scopes}

A scope is a namespace for user-defined types, typedef-ed types, constants and native functions.
There is always a global scope, a scope that represents the source file itself. All enumerations, typedefs and user-defined types declared in the file (and not enclosed by other user-defined types) are said to be defined at the global scope.
```

C++
typedef int MyIntType;
enum MyEnum
}. // .
};
const MyConstant = 5;
struct MyStruct
// .

```

In the example above, all identifiers, that is, MyIntType, MyEnum, MyConstant and MyStruct are declared in the global scope.
Each user-defined type creates its own scope:
C++
\(\frac{c++}{s t r u c t ~ A} / / A\) is declared in the global scope and creates its own scope
\}. const \(B=10 ; \quad / / B\) is declared in scope of structure \(A\)

Every enclosed scope "sees" all its parent scopes. That is, when name-lookup is performed (for the search of the identifier), first, the most enclosed scope is searched. If the identifier is not found, a parent scope is searched and so on

This allows an identifier to be overloaded in the enclosed scope:
```

C++
struct A
{ const MyConstant = 5
int array[MyConstant]; // will use MyConstant from the A scope
};
struct B
int array[MyConstant]; // will use MyConstant from the global scope
};

```

A compiler does not provide a way to reference MyConstant from the global scope from scope \(A\) in this example. But once the scope is closed, a global scope is active again (see structure \(B\) ).

Enumerations are slightly different in a way they use scopes: enumeration declaration does not create a scope but places all enumeration values into the parent scope. See the enumerations section for more information.

\section*{Constants and Constant Arrays}

Constants allow you to calculate the value of an expression and store it. Compare constants with preprocessor constants. Preprocessor is run before compilation of source file occurs and therefore has limited expression evaluation capabilities

\section*{Syntax}
```

const name-id = value-expr;

```
Constants are allowed in any scope. A value-expr must be constant:
\(\frac{\text { C++ }}{\text { const MyConstant }=5 \text {; // valid, constant expression }}\)
struct A
\{ int a;
    int a;
const double_a \(=a * 2 ; \quad / / ~ e r r o r, ~ v a l u e-e x p r ~ i s ~ n o t ~ a ~ c o n s t a n t ~\)
\};

\section*{Constant Arrays}

You can use the following syntax to declare an array
\(\frac{\text { c++ }}{\text { const }}\) name-id []\(=\{\) initializer-list \(\} ;\)
Where name-id is a name of array and initialize-list is a comma-separated list of expressions that initialize array elements.

\section*{Variables and Variable Arrays}

Variables work almost like constants with an exception that you can change their value at a later time.
Syntax:
```

c++

```

You do not specify the type of the variable. It is automatically determined from the type of the value-expr. You may change the variable type at a later time by assigning another value to a variable.

Variables are allowed at user-defined type scope. If value-expression is omitted, variable is initialized with zero. You can change the value of a variable using the following syntax:

C++
variable-name-id = expression;
Variables may be used in expressions just like constants. For example:
C++
\(\frac{\text { ct+ }}{\text { struct A }}\)
// var SomeVariable \(=10\); // declare variable and assign a value to it
SomeVariable \(=\) SomeVariable * 2-20; // change the value of the variable
// ... char Data[SomeVariable]; // use variable
\};

\section*{Optimizations}

Hex Editor Neo performs an optimization when it compiles variables. All constant expressions are evaluated to their numeric equivalents. Hex Editor Neo can also optimize constant sub-expressions if they do not have side effects.
```

C++
struct A
var MyConstant =60* 60; // will be optimized directly to 3600
var PI = 3.1415926; // will be optimized directly to 6.2831852
int à;
var test = a* (_2PI / 180); // sub-expression in parenthesis will be optimized
};

```

\section*{Using Variables}

Variables let you overcome the limitations of standard lookup procedure. A good example is a definition of a PNG file structure (installed with Hex Editor Neo). A PNG file
consists of several chunks of different type and size. Although these chunks are almost unrelated, sometimes the structure of a chunk greatly depends on some of the fields of one of the previous chunks. Using variables you may "capture" the value of such field and later use it in subsequent chunks to choose between one structure or another. See the provided sample file for more information on using variables.

Variable Arrays

You can use the following syntax to declare an array
\(\frac{\text { C++ }}{\text { var name-id [ [total-elements-expr] ] [ = \{ initializer-list \} ]; }}\)
Where name-id is a name of array, total-elements-expr is an optional number of elements in an array. If omitted, the number of expressions in the initializer-list sets the number of elements in an array.
initialize-list is a comma-separated list of expressions that initialize array elements.

\section*{Enumerations}

Hex Editor Neo supports the special form of integer constants, called enumerations.
Syntax:
```

C++
enum [name-id[<integer-type>]]
{ value-id [ = const-expression
[, value-id [ = const-expression].
};
// new syntax
enum [name-id[ : integer-type]]
{ value-id [ = const-expression]
[, value-id [ = const-expression] ...
};

```

Enumeration may optionally have a name and a type. If name is omitted, a nameless enumeration is created. If type is omitted, it is defaulted to int.
An expression, if specified, must be a constant expression, that is, its value must be calculable at the compile-time. You may use immediates, constants and other previously defined enumerations as well as sizeof() operator in its static form.

If expression is omitted, the value of the current element is computed as a value of the previous element plus one. If this is a first enumeration element, its value will be 0 :
enum MyEnum : unsigned
\(\{\)
```

{ FirstValue,
FirstValue,
ThirdValue = 5, // overridden, equals
FourthValue, // defaulted to 6
FifthValue = ThirdValue - SecondValue, // equals to 4
};

```

An enumeration does not create its own scope and places all values in the parent scope.
```

$\xrightarrow{\text { C++ }}$ enum MyEnum
{ enum MyEnum
=0x00000010
};

```
const test = FirstValue; // valid, as FirstValue (and SecondValue) are placed in the global scope, parent to MyEnum's scope

\section*{Using Enumerations}

You may use named enumerations as types for fields in user-defined types. If you do this, Hex Editor Neo will automatically recognize the enumeration value when the structure is bound to the data. For example:
```

C++
enum MyFlags : unsigned
FIRST BIT SET =0x00000001,
FIRST_BIT_SET =0x00000001,
};
struct A
MyFlags flags;
};

```

When you bind structure A to data, flags gets visualized as following:
\begin{tabular}{lll} 
Data Value & Display & \\
\hline 1 & FIRST_BIT_SET & \\
2 & SECOND_BIT_SET & \\
0 & 0 & SECOND_BIT_SET \\
3 & FIRST_BIT_SET & 4 \\
5 & FIRST_BIT_SET & \\
8 & 8 &
\end{tabular}

Hex Editor Neo automatically parses the enumeration value and displays its symbolic name (or names).

\section*{User-Defined Types}
```

[public | private] (struct | union | protocol) [name-id]
[ element-decl; [ element-decl; ... ] ]
};
[public | private] case_union [name-id
case expression1:
[ element_decl; .. ]
[
case expression2:
[ element_decl; ... ]
`]
default:
[ element_decl; ... ]
}; ]
element-decl:
[ (field-decl | typedef-decl | const_decl | user-defined-type-decl) ; ...]
field-decl:
type var-decl [, var-decl ...] ;

```
var-decl:
    (id | id[expression] | id:expression | id as type-id * [(expression)] )

The first syntax form allows you to define structure or union, while the second form allows you to define a case union
When public keyword is placed before the structure declaration, the structure becomes a public user-defined type and appears in Bind Structure Window. A private keyword may be used to prevent the structure from being listed in the list of user-defined types in the Structure Binding window. If omitted, a structure declaration is private by default. Applies only to user-defined types, declared on the global scope

A special directives may appear within a declaration of a structure, union or a case union:
\(\frac{\mathrm{C}++}{\text { hidden }}\)
visible

A hidden: directive hides all subsequent fields and visible: directive makes fields visible. By default, all fields are visible. Hiding a field only hides it from the screen, the field remain visible when referenced in expressions.

This and subsequent sections provide an in-depth description of user-defined types.
Each user-defined type creates a scope. All enclosed constants, enumerations, typedefs, native functions and user-defined types are then included into this newly created scope.

A structure, union or case union may be nameless. Nameless types are allowed anywhere besides the global scope. A nameless type may also be used in the typedef declaration:

C++
typedef struct
( \({ }^{1 /}\)
equivalent to:
C++
struct A
\{ // .
\};
while the following fragment creates a structure named \(A\) and its aliases \(B\) and \(C\) :
C++
typedef struct A
\{ // ..
\} \(\mathrm{B}, \mathrm{C}\);

Supported Types

Structures

A structure is a combination of data fields. A structure occupies space required to store all its fields, one by one, subject to structure packing or alignment.
A structure definition consists of zero or more of data fields:
type var-decl [, var-decl...];
where var-decl is:
(id | id[array-size-expression] | id:bit-field-size-expression | id as type-id *)

A structure may contain different fields with a same name. If such a field is referenced in expression, an [] operator may be used to address individual fields. If this operator is not used, the first field is referenced.

Typedefs, constants, enumerations, native functions and nested user-defined types are also allowed within a structure definition.
Plain field, array field, bit field and pointer field are described in more detail in their corresponding sections.
Empty structures are eliminated from the output. See also the noautohide attribute section.
Example:
```

C++
struct A
int a; // plain data field
int b:3; . // bit-field
int c[10]; *. // array
int d as B *; // pointer
short s,t[5],u:12; // multiple fields may be combined
};

```
```

Packing and Alignment

```

The important thing about user-defined types is the size and alignment of a type. The size of the structure is a sum of sizes of all its fields (subject to alignment). The alignment of the built-in type equals its size, and alignment of the structure is calculated by Hex Editor Neo, taking in account the alignment of all structure fields and current structure packing value. By default, structure packing value is 1 .

You may change the structure packing value using the following directive:

C++
\#pragma pack(N)
where \(N\) is one of the following values: \(1,2,4,8,16,32\).

\section*{NOTE}

The following rule is used when computing the alignment of each structure field:
Each data field starts at offset which is a multiple of its alignment. A number of unused padding bytes is inserted if required, but no more than the current structure packing value

The implementation of structure packing and field alignment in Hex Editor Neo complies with standard C implementation.

\section*{Byte Orde}

By default, Hex Editor Neo respects the current byte order specified for the editor window. You can change the byte order at any time using the following directive:
```

\#pragma byte_order(LittleEndian | BigEndian | Default)

```

This directive changes the current byte order until the end of the current scope, or until another byte_order directive.

Unions

A union is a combination of data fields. In contrast to a structure, all union data fields are located at the same address and, therefore, share the same storage space. The size of the union equals to the size of the largest field, and alignment of the union equals the largest field's alignment.

A union definition consists of zero or more of data fields:
```

type var-decl [, var-decl...];

```
where var-decl is:
(id | id[array-size-expression] | id:bit-field-size-expression | id as type-id*)
A union may contain different fields with a same name. If such a field is referenced in expression, an [] operator may be used to address individual fields. If this operator is not used, the first field is referenced

Typedefs, constants, enumerations, native functions and nested user-defined types are also allowed within a union definition.
Plain field, array field, bit field and pointer field are described in more detail in their corresponding sections.
Example:
\(\frac{\text { C++ }}{\text { union }}\)
\{ union \(A\)
int intVal;
short shortVal;
double dblVal;
char charArray[4]
struct
struc
int a;
int 64
\} nestedStruct;
\};

\section*{Case Unions}

A case union is a special construct offered by a Hex Editor Neo to dynamically select types at run time. It is virtually impossible to describe real-world data structures using only static types (types offered by C compiler, for example). And although dynamic types greatly increase the flexibility of the language to describe data structures, they still lack the ability to select one or another type based on run time conditions.

For example, imagine we have a byte in the data structure, followed by one of three different data structures, depending on this byte's value. Hex Editor Neo's case unions allow you to describe such data structure.

Case union syntax (copied from User-Defined Types section):
A union definition consists of zero or more of data fields:
```

C++
case_union [name-id]
case expression1:
[ element_decl; ... ]
case expression2:
[ element_decl; ... ]
\#
default:
[ element_decl; ... ]
};

```
element-decl:
    [ement-decl:
(field-decl | typedef-decl | const_decl | user-defined-type-decl) ; ...]
field-decl:
    type var-decl [, var-decl ...] ;
var-decl:
(id | id[expression] | id:expression | id as type-id *)

Case union consists of one or more case blocks and an optional default block. Hex Editor Neo evaluates each expressionN and if it is nonzero, elements immediately following a case block are used. All other case blocks and default block are ignored. If none of case expressions is evaluated to a non-zero value, the default block is used. If the default block is omitted, a case union becomes an empty structure and is removed from the output.
```

C++
struct B
//.
};
struct C
{ // ..
};
struct A
BYTE val;
case_union
{
case val == 0:
int b;
case val == 1
B b;
C c;
} s;
};

```

Case Union Optimization

All case expressions are evaluated at compile time. If an expression is a non-zero constant expression, a whole case union becomes equivalent to a corresponding structure. If all case expressions evaluate to constant zero, the whole case union becomes equivalent to a structure with fields from the default block. If there is no default block in a case union, it becomes an empty structure and is eliminated from the output.

\section*{Forward Declarations}

Hex Editor Neo Neo requires that each referenced type must be defined before it is used; otherwise, the compile-time error occurs. Sometimes it is impossible or inconvenient to follow this rule. Forward declarations allow you to declare the identifier as a user-defined type, without actually defining it:
```

struct B; // forward declaration of B
struct A
B b; // compiles OK, as B has been declared
};
struct B // actually define B
// ..
};

```
Syntax:
(struct | union | case_union) name-id ;
name-id is required in forward declarations.
All forward declarations must be resolved before the end of the source file, otherwise an error occurs. However, it is not an error not to resolve a forward declaration if it has not been referenced.

\section*{Data Fields}

\section*{Plain Field}

Plain field is an ordinary field of a given type.
Syntax:
type-id var-id;

Example of plain fields:
```

C++
{ // .
};
struct A
{
int a;
};

```
Both \(a\) and \(b\) fields of structure \(A\) are plain fields.
Array Field

Array field describes an array of some data type. Array is a sequence of several values of the same type.
Syntax:
[[noindex]] type-id var-id[expression]
Expression may be constant expression or dynamic expression. If it evaluates to 0 , the field is removed from the output.
Hex Editor Neo automatically distinguishes between a simple array and an ordinary array. An array of elements of integer and floating-pointbuilt-in types is considered a simple array. Simple arrays do not impose a limit on a number of items and are extremely cheap to bind and consume no memory at all. Ordinary arrays (that is, arrays of userdefined types, or string types) have a hard-coded limit on array item count (about 2 million items) and require a corresponding amount of free RAM. A noindex attribute may precede the array field declaration if you do not use an Array Indexing Operator in any of the expressions to refer to elements of this array.

This will save the memory and structure binding time.
Example of array fields:
```

C++
struct B
\/l
};
struct A
int a[5]; // static array, considered as simple array
B b[a[0]]; // dynamic array (a number of elements is taken from the first element of a). Not a simple array
};

```

\section*{Simple and Ordinary Arrays}

Any array of elements of integer or floating-point built-in types is called a simple array. Array of elements of other types is called an ordinary array. The following table briefly describes the difference between two array types:
\begin{tabular}{lll} 
Property & Simple Array & Ordinary Array \\
\hline User-defined element type supported & No & Yes \\
\hline Consumes more memory as array's size increases & No & Yes \\
\hline Consumes more time as array's size increases & No & Yes \\
\hline Has upper array size limit & No & Yes \\
\hline Supports [noindex] attribute & No, ignored if used & Yes, consumes less memory and time if used on an array
\end{tabular}

\section*{"Infinite" Arrays}

You are allowed to declare an array of "infinite" size when declaring an array. The following syntax is used:
type-id var-id[*];
type-id must be a user-defined type that must contain at least one \$break_array directive that will specify the last element of the "infinite" array.
This is very useful if array size is not known at the array declaration point. For example, the following code snippet is capable of parsing a C-style null-terminated string:
```

C++
struct StringCharacter
{
if (c;= 0)
\$break_array(true)
};
struct NullTerminatedString
StringCharacter chars[*];
};
// Display the class usage
struct FileStructure
{
NullTerminatedString FileName
NullTerminatedString Location
Nul
};

```

\section*{Visualization}

When Hex Editor Neo visualizes an array, it optionally visualizes its first several values in-line. Other values are displayed when you expand the array item. Character arrays (arrays of elements of char and wchar_t types) are visualized as ANSI or UNICODE strings respectively

Bit field is an integer field which occupies less space than the underlying integer type.
Syntax:
typeid var-id:expression;

The underlying type-id of a bit field must be an integer type. Expression may be constant expression or dynamic expression.
Example of bit fields:
```

C++
struct Date
unsigned short nWeekDay : 3; // 0..7 (3 bits)
lun
unsigned short nYear : 8; // 0..100 (8 bits)

```
\};

The conceptual memory layout of an object of type Date is shown in the following figure.


Note that nyear is 8 bits long and would overflow the word boundary of the declared type, unsigned short. Therefore, it is begun at the beginning of a new unsigned short. It is not necessary that all bit fields fit in one object of the underlying type; new units of storage are allocated, according to the number of bits requested in the declaration.

The ordering of data declared as bit fields is from low to high bit, as shown in the figure above.

\section*{Pointer Field}

Pointer field is functionally equivalent to a plain field but additionally describes the type this field points to. When Hex Editor Neo binds a pointer field, it automatically binds a pointed type at a calculated offset.

Syntax:
```

type-id var-id as pointed-type-id * [(expression)];

```

Expression, if present, is evaluated at run-time and the result is added to a field value. type-id must be an integer type. pointed-type-id must be a user-defined type.
Example of pointer field
```

C++
struct B
// .
};
struct A
{hort ptr1 as B *
unsigned int ptr2 as B *(10); // B will be bound at offset (ptr2 + 10)
};

```

The resulting offset must be an absolute offset in a file. In cases where fields contain only relative offsets, this keyword may be used in an expression to "convert" a relative offset to an absolute offset:
\(\frac{\text { C++ }}{\text { struct } B ;}\)
\({ }^{\text {struct } A}\)
short ptr1 as B *(this); // B will be bound at offset (this + ptr1)
\};
When pointers are processed and pointed structures are bound at resulting offsets, the current structure scope is used as a parent scope for a bound structure. This allows referencing its fields from the pointed structure:

C++
struct B;
\(\sum_{\{ }^{\text {struct } A}\)
int a;
int ptr as B *;
\};
struct B
int array[a]; // will reference a in A, if B is automatically bound via a pointer
\};
Late-evaluation is performed for pointers, which allows pointed types to reference fields from the parent scope even if they are defined below the pointer field.

\section*{Attributes}

Field Attributes

Hex Editor Neo supports a number of useful attributes that change the default behavior for individual bound fields. You should put attributes before a field you want them affect. The following syntax is supported:
```

attribute-list:
[ *attribute-decl]
attribute-decl:
noindex | noautohide | read(expr) |
format(expr) | description(expr) | color_scheme(expr)

```

See the following sections for attribute descriptions.

\section*{Type Attributes}

In addition to field attributes, a single display attribute is supported on types. It allows the user to change the default visualization for a type.

\section*{Field Attributes}
noindex Attribute

Syntax:
noindex

Specifying this attribute for an array field turns off automatic building of array index. Array index is required for Array Indexing Operator to work properly. This attribute is ignored if used on simple array or non-array field

\section*{noautohide Attribut}

Syntax:
noautohide
By default, Hex Editor Neo eliminates fields which have zero size during binding. Specifying this attribute allows you to turn this behavior off.
onread Attribute

Syntax:
onread(expr)
Allows you to change the way this field is read by the Hex Editor Neo. You may specify expression to be evaluated each time Structure Viewer accesses the field's value.
Take the following notes into consideration:
- Your expression will be evaluated each time the field's value is read from the document, even during binding.
- Hex Editor Neo caches field values. This means that the result of your expression must be persistent. That is, for any \(x\) and \(y\), if \(x==y, \operatorname{expr}(x)==\operatorname{expr}(y)\).

In expression you may refer to actual field's value using a special variable _1:
C++
struct A
```

    // The following field stores the size of the array minus 2
    [onread(_1 + 2)]
    int array siz; will be displayed as actual value + 
    char array[array size];
    ```
\};

\section*{format Attribute}

Syntax:
format(const-string-expr)

Use this attribute to set the format string used during visualizing of the field's value. Note that only part of the full format string must be specified: instead of "\{0b16\}" use just "b16". const-string-expr is evaluated at compile time.
```

description Attribute

```

\section*{Syntax}
description(const-string-expr)
Set the field description to be displayed in Hex Editor Neo user interface. const-string-expr is evaluated at compile time.
```

color_scheme Attribute

```

Syntax:
color_scheme(const-string-expr)

Set the color scheme to be used by Hex Editor Neo to visualize the field. const-string-expr is evaluated at compile time.
```

exact_only Attribut
Syntax:
C++
exact_only

```

Using this attribute for enum fields forces exact matching algorithm. Normally, when this attribute is not specified, the following visualization algorithm is used for enumerations:
1. First, enumeration type is searched for exact match. If exact match is found, enumeration identifier is displayed.
2. If exact match is not found, enumeration type is considered a bit-field and is decomposited into components. Any remaining value is then added at the end.

\section*{Type Attributes}

\section*{display Attribute}

Syntax:
display(expr)

This attribute may be used to override the default type's visualization algorithm.
When Hex Editor Neo generates a value for a collapsed type, it by default displays values of first 5 type's fields. This attribute allows you to change that.
Hex Editor Neo evaluates the expr expression in the context of the current type (that is, you may reference all type fields directly). The result is then converted to string and displayed in Structure Viewer

The following example renders the user-friendly MAC address:

C++
[display(format ("\{0b16Xw2arf0\}:\{1b16Xw2arf0\}:\{2b16Xw2arf0\}:\{3b16Xw2arf0\}:\{4b16Xw2arf0\}:\{5b16Xw2arf0\}", data[0], data[1], data[2], data[3], data[4], data[5]))]
struct MAC
unsigned char data[6];
\};

The following example displays the sum of array's values:
```

C++
function sum(r)
var sum = r[0];
for (var i = 1; i < 6; i= i + 1)
sum = sum + r[i];
return sum;
}
[display(sum(ref(data)))]
[display(s
unsigned char data[6];
};

```

\section*{Typedefs}

You can create an alias for any built-in or user-defined type.
Syntax:
```

C++
typedef existing-type new-type-id [,new-type-id ...];

```
\(\frac{\text { C++ }}{\text { using new-type-id }=\text { existing-type; }}\)

Type alias definitions are allowed at any scope. existing-type must be either built-in type, enumeration or user-defined type. Nameless types are allowed.
C++
typedef int INT;
typedef struct \{ int \(\mathrm{a}, \mathrm{b}\); \} MyStruct;
struct A
//
typedef A B,C,D;
using \(E=A\);
Typedef does not create a new type, it only creates an alias for an existing type. You may reference a type by its original name, or by one of its aliases

\section*{Directives}

Directives are special commands to the compiler which are only allowed at non-global scope.
View the subsequent topics to get detailed description of each supported directive.

\section*{\$assert Directive}

Syntax:
\$assert(condition-expression [, message [, fatal-expression]]);
Assert directive is evaluated at run-time. First, condition-expression is evaluated. If it evaluates to non-zero value, nothing happens. But if it evaluates to a zero value, message is displayed to the user and structure binding is terminated. If message is omitted, standard "Assertion Failed" message is displayed. message, if present, must be a string expression. fatal-expression, if present, must be a constant expression. If it evaluates to a non-zero value, the assertion is fatal (this is a default behavior), that is, fired assertion terminates structure binding. If it evaluates to zero, assertion is only informational

Assertions are good at verifying whether the data structure is being bound to correct data.
```

C++
struct A
{ int a;
\$assert(5 < a \&\& a < 10,"a must be between 5 and 10");
};

```

\section*{\$print Directive}
```

Syntax:

```
```

\$print(var-name-string-expression, var-value-expression);

```
```

\$print(var-name-string-expression, var-value-expression);

```

The first expression must be a constant string expression and serves as a pseudo-field name. This name is displayed in the Structure Viewer Tool Window. It is also added as a real field into the current scope and may be later referenced in expressions. The second expression is a field's value. It may be either a constant expression or a non-const expression, that will be evaluated at run time.
```

C++
int a;
int b;
\$print("double_a", a * 2);
\$print("a/b", double(a) / b);
};

```

Both \(\$\) print directives in the example above introduce new fields into the current scope (of struct A), but only the first one may be referenced in expressions, because the second one has incorrect name. So, it's OK to use any name for the first argument, but only syntactically correct ones may be referenced in expressions.
\$break_array Directive

Syntax:
\(\frac{\mathrm{C}++}{\text { +bre }}\)
\$++ \(\$\) break_array(const-conditional-expression)

When used in a user-defined type, unconditionally breaks an enclosed array.
The parameter, which must be a constant value, specifies whether the current element should be included into an array (const-conditional-expression \(=\) true), or not (const-conditional-expression \(=\) false)

May be used either in conjunction with infinite array or with an ordinary array. If used not inside an array, the directive is ignored.
```

C++
struct StringCharacter
{ char c
char c;
if (c == 0)
_array(true);
};
struct NullTerminatedString
StringCharacter chars[*];
};

```

\section*{bind Directive}

Syntax:
C++
C++ \(\$\) bind(type-string-expr, var-string-expr, addr-expr);

All expressions are evaluated at run time. First two are automatically converted to strings, while the third is expected to be of integer type.
This directive instructs parser to bind another structure to a given address. Binding is delayed until the current structure binding is successfully finished.

\section*{NOTE}

You must reference the full type name, for example struct MyStruct, not the MyStruct, and the referenced type must have been declared as public.
```

C++
public struct B
{ // ...
};
public struct A
int Offset;
if (Offset != 0)
\$bind("struct B","pB",Offset);
};

```

\section*{\$alert Directive}

Syntax:
C++
\$alert(expr);
Evaluates expr at bind time and displays the result in a message box.
```

Syntax:
C++
C++
struct A
{ int type
switch (type)
{witch (type)
case 0:
B b;
case 1:
C c;
fault:
default:
\$revert_to(ref(type));
}
};

```
This directive updates the current_offset. It may be used to have look ahead in a structure. reference-expression is evaluated at run-time and must be a reference to a field
in the current type. After this directive executes, current offset becomes the start of a field.
\$shift_by Directive
Syntax:
\$shift_by(integer-expression);

This directive updates the current_offset. It may be used to have look ahead in a structure. integer-expression is evaluated at run-time and must be an integer value, which is added to current_offset upon directive execution.
```

C++
struct A
int skip_bytes;
\$shift_by(skip_bytes);
int next field;

```
\};

\section*{\$remove_to Directive}

Syntax:
\$remove_to(reference-expression);
This directive removes one or more last bound fields until the referenced field. reference-expression is evaluated at run-time and must be a reference to a field in the current type.

\section*{WARNING}

All fields being removed must be visible and no hidden fields must be between them.
```

C++
struct A
int type;
int type;
{wi
case 0:
break;
case 1:
C c;
default:
\$remove_to(ref(type));
}

```

\section*{Format String Syntax}

This section describes the format string syntax. Format string is used in format() function and format attribute.
Library format string syntax is not compatible with the standard printf syntax. Instead, it has a different syntax.
The format string has blocks of plain text which are directly copied to the output and parameter placeholders. Each placeholder has the following syntax:
```

\{<param-index>[width-dec1][alignment-dec1][plus-dec1]
precision-decl][base-decl][padding-decl][ellipsis-decl]
locale-decl]\}

```

The placeholder must be enclosed in curly braces. If you need to use the opening curly brace in the text, you need to duplicate it to distinguish from the placeholder beginning. There is no need to escape the closing brace, it will always be parsed correctly.

Parameter declaration starts with a parameter's number. This is the only mandatory field. Parameters are ordered starting from zero. All subsequent declarations are optional. If several declarations are used, their order is not significant and there must be no space or any other separator between them.

\section*{width-decl}

Use this declaration to limit the minimum and/or maximum length of a rendered parameter, in characters. The syntax of a declaration is one of:
- w<min-width>,<max-width>
- w<min-width>
- w, <max-width>

Both min-width and max-width must be decimal integers and if specified, max-width must be larger than min-width.
alignment-decl
Use this declaration to set parameter alignment. It is ignored unless width-decl is also used. Use one of:
- al - align left (default)
- ar - align right
- ac - align center
plus-decl
Forces the plus sign to be rendered for positive numbers. Syntax
- +

\section*{precision-decl}

Use the declaration to specify the number of digits to be displayed after the comma. Used only for floating-point types. If not specified, the default one (6) is used
- p<number>
base-decl
Specify a base for an integer. If any base besides 10 is used with the floating-point type, only the integer part is rendered. Only bases of \(2,8,10\), and 16 are supported. Lowercase or uppercase hexadecimal may be specified:
- b2 - binary
- b8 - octal
- b10 - decimal (default)
- \(\mathrm{b}[0] 16[\mathrm{x}]\) - lowercase hexadecimal. If prefix " 0 " is used, library adds " 0 x " before the number
- \(\mathrm{b}[0] 16 \mathrm{X}\) - uppercase hexadecimal. If prefix " 0 " is used, library adds " \(0 X\) " before the number

\section*{padding-decl}

Set the character to fill the space when min-width is set (see the width-decl above). The default one is space
- f<character>
ellipsis-decl
Add the ellipsis sign when truncating output. It is not compatible with center alignment (will act as left alignment).
locale-decl

Separate thousands with the default user locale's thousand separator. Will work only for base 10.
- 1

\section*{Errors}

This topic describes all compilation and binding errors generated by Hex Editor Neo.
CE001: The requested operation is not allowed on the given data type or not expected here
Operation or operator you attempted to use is not supported. Example:


CE002: Divison by zero
Divison by zero has been encountered.
C++
int array[5/0];
CE003: The specified identifier was not found
You referenced the previously undeclared identifier.
\(\frac{\text { c++ }}{\text { public struct } A}\)
\(\left\{\begin{array}{l}\text { public stru } \\ \text {, }\end{array}\right.\)
int a;
int arr[c]; // generates CE003, \(c\) is undeclared
\};

\section*{CE004: Scalar is expected}

A requested operation is allowed only on scalar values.
\(\frac{\mathrm{C}++}{\text { public struct } A}\)
\(\{\) int data[10];
int array[data]; // generates CE004, data is not a scalan
\};

\section*{CE005: Vector is expected}

A requested operation is allowed only on vector values.
\(\stackrel{\text { c++ }}{\text { public struct } A}\)
\(\left\{\begin{array}{l}\text { public str } \\ \text { int a; }\end{array}\right.\)
int \(\mathrm{b}[\mathrm{a}[1]]\); // generates CE005, a is not a vector
\};

\section*{CE006: Array index is out of range}

An attempt to access array's element that is outside of the declared array size.
```

C++
public struct A
int a[10];
int b[a[12]]; // generates CE006, a only has 10 elements (0..9)
};

```

\section*{CE007: Not implemented yet}

This operation has not yet been implemented.

\section*{CE008: Invalid bit field size}

Unsupported bit field size is used.

\section*{CE009: Syntax error}

See error's additional message for detailed syntax error information

\section*{CE010: Type has only been forward-declared}

An attempt to materialize a type that has only been forward-declared is detected.
```

C++
struct B;
public struct A
B data; // generates CE010
};
// end of file

```

CE011: Operation not supported for dynamic type
sizeof operator is used with dynamic type.
\(\frac{\text { C++ }}{\text { struct }}\)
\{ int size;
char data[size];
\};
public struct A
char reserved[sizeof(B)]; // generates CE011, B is dynamic type
\};

\section*{CE012: Type is redefined}

An attempt to redefine already defined type is detected.
c++
struct B
int a;
\};
struct B // generates CE012
int c ;
\};

\section*{CE013: Assertion failed}

Assertion (generated by \$assert directive) has failed.

\section*{CE014: Constant expression is expected}

Compiler expects a constant expression here.
CE015: Constant string expression is expected
Compiler expects a constant string expression here.
CE016: Wrong number of arguments for a function call
An invalid number of arguments used in a call to built-in function.
CE017: Subscript operation for non-indexed array (hint: remove [noindex] attribute)
[] operator has been used for non-indexed array. Remove the noindex attribute

C++
struct B \{ ... \};
struct A
\({ }^{\text {str }}\)
```

[noindex] B data[1000];
char reserved[data[5].size]; // generates CE017, remove [noindex] from previous line

```

CE018: Error returned by JavaScript engine
JavaScript function returned an error.

\section*{CE019: Invalid argument}

Invalid argument has been passed to built-in function

\section*{CE020: Maximum allowed recursion depth is reached}

Check your source file for infinite recursion.
```

C++
struct A
B b;
};
struct B
A a;
};
// will generate CE020 after several iterations

```

\section*{CE021: Expression of another type is expected}

Compiler expects expression of another type here. Additional information specifies what type is expected.

\section*{Examples}

HHD Software maintains a public structure definition files repository on-line. Hex Editor Neo can be configured to automatically synchronize with this public repository using one of the two supported channels:

\section*{Public release channel}

Hex Editor Neo synchronizes with published releases only. This is the default channel.

\section*{Commit channe}

Hex Editor Neo synchronizes with all repository commits.
Repository synchronization may also be turned off.

\section*{Collaboration}

Users are welcome to submit Pull Requests with corrections to existing structure definition files and their metadata as well as submit new structure definition files for wellknown data formats.

\section*{Disassembler}

A disassembler is a computer program that translates machine language into assembly language - the inverse operation to that of an assembler. A disassembler differs from a decompiler, which targets a high-level language rather than an assembly language. Disassembly, the output of a disassembler, is often formatted for human-readability rather than suitability for input to an assembler, making it principally a reverse-engineering tool.

\section*{Example}

A processor consumes instructions encoded as raw bytes to change its internal state (to calculate something or to change the operation flow). For example, sequence of three bytes 83 C4 04 is interpreted by 886 processor as an instruction to add value 4 to register esp. However, it is difficult to operate with raw instruction bytes for humans. It is much easier to represent the same command as an assembler instruction:
\[
\text { add esp, } 4
\]

This instruction is directly translated (or assembled) to 83 C4 04 .
Disassembler is intended to do the opposite. It will parse 83 C4 04 and produce a add esp, 4. So, a disassembler is needed to convert raw binary file into something readable by a human eye.

\section*{Supported Instruction Sets}

A current version of the Hex Editor Neo supports disassembling:
- x86
- x86/x64
- MSIL
- ARM (including Thumb, Thumb2)
- ARM64 (AArch64)
- MIPS
- PowerPC
- Sparc
- SystemZ
- XCore
- 68K
- TMS320C64x
- 680X
- MOS65XX (including MOS6502)
- RISC-V

\section*{Definitions}

This section provides a set of term definitions that will help you to get accustomed to the Hex Editor Neo's disassembler module.

\section*{PE File Format}

The Portable Executable (PE) format is a file format for executables, object code, and dynamic-link libraries, used in all versions of Microsoft Windows operating systems. The term "portable" refers to the format's versatility in numerous environments of operating system software architecture. The PE format is basically a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code. A PE file consists of a number of headers and sections that tell the loader how to map the file into memory.
Hex Editor Neo's disassembler gathers information needed to parse the PE (Portable Executable) file from headers that are located at the beginning of a file.

\section*{PE Sections}

An executable file (usually the one with .exe or .dll extension) is divided into sections (or segments) that are mapped to memory during load. Different sections can be mapped to different virtual addresses (raw section's size and size of section in memory may differ).
Usually different sections are introduced to split the file into several code/data blocks with different memory protection modes and to provide paging (swapping) mechanism.
Disassembler window allows you to select the section you want to disassemble. A section that contains executable code is usually named ".text" or "CODE".

\section*{ELF File Format}

ELF is a common standard file format for executable files, object code, shared libraries, and core dumps. It is the standard binary file format for Unix and Unix-like operating systems.

\section*{Virtual Address}

Virtual address is an address identifying a virtual (non-physical) entity. Virtual Address is used to describe location of data mapped into memory. Physical Address \(0 \times 1000\) (in file) can be mapped to virtual address \(0 \times 401000\). Disassembler displays both addresses - physical (column "Raw Address") and virtual (column "Virtual Address"). In case no section information is available for an image, raw offsets cannot be converted to virtual addresses and the column is empty.

\section*{Symbols}

Debug symbols is the information about what high-level programming language constructions generated specific piece of machine code in the given executable module. Sometimes it's embedded into the module's binary, or distributed as a separate file, or just discarded during the compilation and/or linking. Symbols enable a person using a debugger to gain additional information about the binary, such as the names of variables and routines from the original source code. This information is sometimes extremely helpful while trying to investigate and fix a crashing application.
Hex Editor Neo uses only part of debug symbols, namely, it is capable of displaying function and variable names where appropriate. It is also capable of automatically locating symbol files for used modules and showing exported function names.

\section*{Starting Disassembler}

To start disassembler, execute the Tools » Run Disassembler... command.
If you have a document opened in the editor, the disassembler window automatically selects this document for disassembling. Both files and processes may be disassembled this way. Hex Editor Neo is capable of locating a proper image and loading its PE header.

If the current document has a selection, its start and size are used to fill the corresponding fields in the Disassembler window. Otherwise, the cursor's position is used as a starting offset.

Disassembler Window


File
Enter the full path to an executable file or press the browse ("..." button)

\section*{Instruction Set}

Select a required instruction set. Disassembler automatically selects the valid instruction set, according to the fields in the image's PE header.

\section*{Section}

If the file is in the PE or ELF format, a list of file sections is displayed in this field. A first code section (usually named ".text" or "CODE") is automatically selected. You can select any other section, effectively changing the value of the Start and Size fields. As described above, if the current document is used for a disassembler, Start and Size fields are automatically filled using the cursor's position and/or current selection.

This field is disabled if MSIL instruction set is selected.

\section*{Symbols...}

Click to display the Symbols Window for a selected file. Using this window you can select a specific symbol and instruct the disassembler to disassemble only this symbol (that is, to update the Start and Size fields appropriately).

\section*{Start}

The starting offset for disassembling. This field is disabled if MSIL instruction set is selected.
Size
The size of the area for disassembling. This field is disabled if MSIL instruction set is selected

\section*{Hexadecimal/Decimal switch}

Set to specify whether offset and size are entered as hexadecimal or decimal numbers.

\section*{Little/Big Endian}

For formats that support different endianness (like ELF), allows you to choose the correct data encoding in an executable file

\section*{Dual view}

If checked, both Disassembler View and usual editor window will be opened for a document. The views will be connected to each other, that is, highlighting an instruction in Disassembler View will select its bytes in a editor window provided "Synchronize cursor with Hex View" and "Synchronize selection with Hex View" options are enabled.

\section*{Synchronize cursor with Hex View}

Paired editor window's cursor movement results in a disassembler's active row being changed.

\section*{Synchronize selection with Hex View}

Disassembler automatically selects all bytes of the currently selected instruction in disassembler. This selection replaces the current selection of the paired editor window.
Reset
Click to reset the state of all fields to their default values.

\section*{Headers}

This section displays the contents of the file's PE Header.
Symbols Window

Symbols window displays available symbolic information for a selected document.


The window is structurally divided into three parts. There is a horizontal splitter that can be used to change the size of these parts and the whole Symbols Window is resizable as well. There are Source files list at the top left, Module details field at the top right and Symbols list at the bottom of the window.

\section*{Source Files List}

This list contains a full list of source files used to compile the given executable. This list may be empty if corresponding information is missing in the symbol file.
Each source file is displayed with full path name. Please note that path names are only valid on the same computer where the image was compiled.
You can copy the whole list or only a part of it to the Clipboard using the shortcut menu.

\section*{Module Details}

Module details field provides information about the symbol file, its format, location and several other properties. It displays whether line numbers, global symbols, type information and public symbols are present in the symbol file or not. Hex Editor Neo does not use all information from the symbol file.
You can copy the contents of the window to the Clipboard using the shortcut menu.

\section*{Symbols List}

This list at the bottom of the window lists all symbols located for a given document. A symbol's raw address, symbol's undecorated name, size, type and flags are displayed.
You can use the Address and Name fields to search for a specific symbol.
Select a symbol and press the Select button (if available) to select it in the Disassembler Window, or press the Jump button to jump to it in the Disassembler View.

\section*{Disassembler View}

Disassembler view displays all decoded instructions in an instruction list. For each instruction, the following information is provided:
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{6}{|l|}{41 (2) FileDocument.dil \(\times\)} \\
\hline - Go To: Ox0040882 & - Raw Address & Go x64 & & & \\
\hline Raw Address & Virual Address & Bytecode & Mnemonic & Operands & n \\
\hline 0x00408838 & 0x0000000180409438 & 0x48, \(0 \times 8 \mathrm{~b}, 0 \times 44,0 \times 2\) & mov & rax,qword ptr [rsp+0x68] & \\
\hline 0x0040883d & 0x000000018040943d & 0x48, \(0 \times 8 \mathrm{~b}, 0 \times 40,0 \times 38\) & mov & rax, qword ptr [rax+0x38] & \\
\hline 0x00408841 & 0x0000000180409441 & 0x48,0x83, \(0 \times 0 \times 0,0 \times 04\) & add & rax, 0x4 & \\
\hline x00408845 & 000000180409445 & \(0 \times 48,0 \times 89,0 \times 44,0 \times 2\) & mov & qword ptr [rsp+0x20], rax & \\
\hline 0x0040884a & 0x000000018040944a & 0x4c, \(0 \times 8 \mathrm{bb}, 0 \times 44,0 \times 2\) & mov & r8,qword ptr [rsp+0x20] & \\
\hline 0x0040884f & 0x000000018040944f & 0x48, \(0 \times 8 \mathrm{~b}, 0 \times 54,0 \times 2\) & mov & rdx, qword ptr [rsp+0x68] & \\
\hline 0x00408854 & 0x0000000180409454 & 0x48, \(0 \times 8 \mathrm{bb}, 0 \times 4 \mathrm{c}, 0 \times 2\). & mov & rcx, qword ptr [rsp+0x58] & \\
\hline 0x00408859 & 0x0000000180409459 & 0xe8, 0xe2, \(0 \times 00,0 \times 0\) & call & _ GSHandlerCheckCormon (0.. & \\
\hline 0x0040885e & 0x000000018040945e & 0x48,0x8b, 0x44,0x2. & mov & rax, qword ptr [rsp+0x20] & \\
\hline 0x00408863 & 0x0000000180409463 & 0x8b, \(0 \times 00\) & mov & eax, dword ptr [rax] & \\
\hline 0x00408865 & 0x0000000180409465 & 0x89, 0x44,0x24,0x2c & mov & dword ptr [rsp+0x2c], eax & \\
\hline 0x00408869 & 0x0000000180409469 & 0x48, 0x8b, \(0 \times 44,0 \times 2\) & mov & rax, qword ptr [rsp+0x50] & \\
\hline 0x0040886e & 0x000000018040946e & 0x8b, 0x40, \(0 \times 04\) & mov & eax, dword ptr [rax+0x4] & \\
\hline 0x00408871 & 0x0000000180409471 & 0x83, 0xe0,0x66 & and & eax, 0x66 & \\
\hline 0x00408874 & 0x0000000180409474 & 0x85,0xc0 & test & eax, eax & \\
\hline 0x00408876 & 0x0000000180409476 & 0x75,0x0d & jnz & 0x180409485 & \\
\hline 0x00408878 & 0x0000000180409478 & 0x8b, 0x44,0x24,0x2c & mov & eax,dword ptr [rsp+0x2c] & E \\
\hline 0x0040887c & 0x000000018040947c & 0x83, 0xe0,0x01 & and & eax, 0x1 & \\
\hline 0x0040887f & 0x000000018040947f & 0x89, 0x44, \(0 \times 24,0 \times 30\) & ov & dword ptr [rsp+0x30], ea & \\
\hline 0x00408883 & 0x0000000180409483 & 0xeb, 0x0b & jmp & 0x180409490 & - \\
\hline & & & & & \\
\hline
\end{tabular}

\section*{Raw Address}

Instruction's raw address in a document.

\section*{Virtual Address}

Instruction's virtual address, if available.

\section*{Relative Offset}

A relative offset inside a method (applicable only in MSIL).

\section*{Bytecode}

Bytes that comprise an instruction.
Mnemonic
Instruction's name.

\section*{Operands}

All explicit instruction's operands, with appropriate modifiers. All referenced symbols and metadata tokens are decoded, if symbol information available.

\section*{Implicit Operands}

Implicit operands, that is operands, which are not explicitly specified, but are still referenced or modified by an instruction. An example of instruction that implicitly modifies registers (esp and eip) is call.

CPU
CPU family in which this instruction appeared.

\section*{Flags Checked}

All flags that this instruction reads.

\section*{Flags Modified}

All flags this instruction modifies
Right-click on the column bar to open the Select Columns Window where you can select columns for the list.
When symbols are available, all jumps and calls for which the Hex Editor Neo is able to locate a symbolic name show this name before the destination address. The destination address in this case follows the name in parenthesis.

Vertical lines in a view may be switched off, using the "Show grid lines" option in General Settings page.

\section*{View Refresh}

When disassembled document is changed through one of the editor windows, disassembly may need to be refreshed. In some cases, where the size of the instruction remains the same, Disassembler View refreshes itself automatically and no additional refresh is required.

In other cases, manual refresh is required. To refresh, use the Refresh command from either the shortcut menu, or from the window's toolbar.

\section*{Positioning within a Document}

Use the Edit » Go to Offset... command to go to a specific offset in the Disassembler View. In addition, you can use the window's toolbar to quickly jump to a given raw or virtual address.

\section*{Searching for a Text}

You can search a Disassembler View for a given text. Use the Edit» Find... command to enter the search sequence and then press the Find button to find the next occurrence. Searching is performed only in selected columns (see above).

\section*{Exporting and Copying to Clipboard}

You can export the whole Disassembler View or only selected part of it into a text file (.txt or .csv) or copy it into the Clipboard. Both commands are available in the shortcut menu. For the export command, you are provided with a choice of columns to export, for the Copy to Clipboard command, the currently selected data is exported.

\section*{Data Annotations}

Data Annotations module allows users to associate text comments with specific document locations. Hex Editor Neo saves those comments with a document and the next time user opens a document, comments are also loaded and are available for browsing and review.

It is possible to specify the coloring for commented cells. The Data Annotations Tool Window displays annotations for the current document.
When the user hovers a mouse cursor over the commented cells, the tooltip with a comment is displayed.
Hex Editor Neo supports several types of storage for a document. The Data Annotations Settings Page may be used to configure the storage location
The following commands are available for the user:

\section*{Annotations » Add Annotation..}

Add new annotation. The user specifies the beginning and end of the range to annotate.

\section*{Annotations » Remove Annotation}

Remove selected annotation.
Annotations » Annotate Selection...
Annotate current selection.
Annotations » Annotate Cursor...
Add annotation for the current cursor location.
Annotations » Clear Annotations
Remove all document annotations

\section*{Data Annotations Tool Window}

Data Annotations tool window displays all data annotations for the current document.


For each annotation, its address, size and comments are displayed. Text comments are colored according to configured coloring scheme.
You can select the annotation and remove it using the Annotations » Remove Annotation command or use the Annotations » Clear Annotations command to remove all annotations.

\section*{Create/Modify Annotation}

This window allows you to provide text comments for a new or existing data annotation and specify a color scheme for it.


Make sure you set non-opaque alpha value for a background color to achieve best color mixing in the editor

\section*{Bookmarks}

A bookmark is a stored location within a document. Bookmarks may be used to quickly navigate through the document.
Hex Editor Neo supports creation of several bookmark groups. Each group holds an unlimited number of bookmarks. You can specify different coloring schemes for different bookmarks. The Bookmarks Tool Window is used to display and manage all bookmark groups and their properties.

Each bookmarks group may be visible or hidden. All bookmarks of the visible bookmarks group are displayed in all editor windows belonging to a given document. All bookmarks groups are properties of a document, so each opened document may have its own bookmarks.

\section*{Bookmarks Tool Window}

Bookmarks Tool Window displays all defined bookmarks groups for a current document. Each bookmarks group may be visible or hidden.


The check box next to each bookmark group signals if the group is visible.
On the right, the Bookmark Details window is displayed. It displays all the bookmarks of the currently selected bookmark groups. You can navigate the list with mouse or keyboard. When you select a given bookmark, the current editor window scrolls to the position of this bookmark.

\section*{Active vs. Selected Bookmark Group}

There is one active and one selected bookmark group in a list. Selected bookmark group is signaled by a selection box while active bookmark group is signaled with an active radio button, which is next to group's name.

Several bookmarks commands always operate on the active group, while other bookmarks command operate on the selected group.
In addition, the Bookmark Details window always displays the details of the selected bookmarks group.

\section*{Creating and Deleting Bookmarks Groups}

To create a new bookmarks group, select the Bookmarks " New Bookmarks Group command. In addition, new bookmarks group is automatically created in some cases, as described in Working with Bookmarks section.

To delete a bookmarks group, select it in a list and press the Del key. You can also delete a group from a shortcut menu.
Shortcut menu's Remove All command may be used to delete all defined bookmarks groups.

\section*{Working with Bookmarks}

There are several commands that work with bookmarks.

\section*{Working with Active Bookmarks Group}

\section*{Toggle Bookmark}

Puts a bookmark on the currently active cell (a cell with a cursor). If there is already bookmark (in an active bookmarks group) on the current cell, it is removed. If there are no bookmarks groups defined for the active document, the new one is created and becomes active.
Complexity: constant-time.

\section*{Next Bookmark}

The cursor is moved to the beginning of the next bookmark within the active bookmarks group. If there are no more bookmarks till the end of the document, the cursor is moved to the beginning of the very first bookmark in a group.

Complexity: constant-time.

\section*{Previous Bookmark}

The cursor is moved to the beginning of the previous bookmark within the active bookmarks group. If there are no more bookmarks till the beginning of the document, the cursor is moved to the beginning of the very last bookmark in a group.
Complexity: constant-time.

\section*{Selection to Bookmark}

Current editor window's multiple selection is converted to bookmarks and added to the selected bookmarks group. If there are no bookmarks groups defined for the active document, the new one is created
Complexity: from constant-time to linear-time.

\section*{Bookmarks to Selection}

The selected bookmarks group is converted to the current editor window's selection. If the window currently does not have a selection, the bookmarks group is silently converted into selection, otherwise, you are presented with a following choice:
- New selection - the current selection is dropped and bookmarks group is converted into the selection.
- Add to selection - the current selection and bookmarks group are merged together.
- Subtract from selection - the current bookmarks group is subtracted from the selection

Complexity: linear-time (depends on the number of blocks in the selection and/or bookmarks group)

\section*{Other Commands}

Save Bookmarks...
All defined bookmarks groups are saved to the file
Complexity: linear-time.

\section*{Load Bookmarks..}

Load one or several bookmarks groups from a file. In addition, loading data from selection files is also supported.
Complexity: linear-time.

\section*{Statistics}

Hex Editor Neo provides unique capability of calculating several file statistics. You may calculate General Statistics and Pattern Statistics for any opened document. In addition, Descriptive Statistics is calculated for both modes, and Entropy Analysis is performed for General Statistics mode.

Statistics Tool Window is used to display the results of statistics calculations. To calculate statistics, open (or activate) the document for which you wish to calculate it, and execute the Tools » Statistics » Refresh command

Statistics is always calculated for the whole file, unless the selection is present in the current editor window. If selection is present, statistics calculation is limited to this selection. Multiple selections are fully supported.

\section*{Statistics Tool Window}

Statistics Tool Window is used to display the results of statistics calculation.
The window provides two views: histogram

and table
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline \multicolumn{9}{|l|}{Statistics} \\
\hline \multicolumn{2}{|l|}{Whole Document} & \multicolumn{2}{|l|}{[88) on dil} & General Statistics & \(\checkmark\) & & & \\
\hline Value (Dec) & Value (Hex) & Value (Char) & Quantity & Percent & Descrin & criptive Satistics & & \\
\hline 0 & 0 & N/A & 388 & 2.9113 & Mean & & 52.0585 & \\
\hline 1 & 1 & N/A & 25 & 0.1875 & Sampl & de Variance & \({ }_{24.6457}\) & \\
\hline 2 & 2 & N/A & 33 & 0.2476 & Stand & dard Erior & 1.5403 & \\
\hline 3 & 3 & N/A & 28 & 0.2101 & Kurtos & & 135.8050 & \\
\hline 4 & 4 & N/A & 77 & 0.5777 & Skewn & ness & 10.0124 & \\
\hline 5 & 5 & N/A & 31 & 0.2326 & Media & & 50 at 51 & \\
\hline 6 & 6 & N/A & 25 & 0.1875 & Range & & 368 & \\
\hline 7 & 7 & N/A & 36 & 0.2701 & Minim & & 20 & \\
\hline 8 & 8 & N/A & 81 & 0.6077 & Maxim & mum & & \\
\hline 9 & & N/A & 28 & 0.2101 & Sum & & 13,327 & \\
\hline
\end{tabular}

\section*{Use the Tools » Statistics » Table View to switch between views.}

There is a statistics calculation type switch at the top of the window. Choose from General Statistics and Pattern Statistics modes. For "Pattern Statistics" mode, a pattern needs to be defined. Press the Define... button to define a new pattern. After it is defined, it is displayed next to the type switch.

\section*{Histogram View}

In histogram view, the graphical representation of the recently calculated statistics is displayed. You may define colors using the Tools »Statistics » Modify Colors... command. You may also copy the histogram image to the clipboard using the shortcut menu.

The Statistics » Hide Maximum Value switch may be used to temporary remove a maximum value from the histogram.
As you move a mouse over the histogram view, a tooltip with more information about the point under mouse cursor is displayed.

\section*{Table View}

In table view, the tabular representation of the recently calculated statistics is displayed. You may sort the table by any column. In addition, the descriptive statistics window is displayed to the table's right. You may change their sizes using the splitter bar between them.

\section*{General Statistics}

General Statistics is a distribution of individual bytes within the document (or selection). A document or selection is scanned and total number of each byte's occurrences is
calculated. The results are then displayed in a table and on a histogram. Descriptive Statistics and Entropy Analysis are then calculated for the resulting data.
General Statistics may be used to estimate the amount of redundancy in a file - if you observe a number of large peaks on a histogram, it means several bytes are more often found in a document than others. This is very typical for executable and most data files. Compressed files, on the other hand, tend to be more "smooth" on their histograms: the number of occurrences of each byte in a file is nearly the same.

\section*{Pattern Statistics}

Pattern Statistics is another statistics calculation mode, provided by Hex Editor Neo, in addition to the General Statistics. In this mode, you define a pattern which is then searched in a document or current selection, if it is present. The whole target range is divided into a number of blocks (you define the number of blocks). The number of occurrences of a pattern in each block is then calculated. The results are displayed in a table and on a histogram. Descriptive Statistics is also calculated for the results.

\section*{Edit Pattern Window}

When you switch to the "Pattern Statistics" mode, or press the Define... button in the Statistics Tool Window, the Edit Pattern Window is displayed.


A pattern is defined using the Pattern Window, which is encapsulated in this window. In addition, the Ignore Case option may be enabled to ignore case while matching the pattern. You also specify the number of blocks into which the file is divided.

\section*{Regular Expressions}

The Pattern Statistics command fully supports regular expressions. To use regular expressions, select either "ASCII string (char[])" or "UNICODE string (wchar_t[])" pattern type, enter the regular expression, make sure the Regular expression checkbox is checked and enter the sub-expression number you want to search for. Sub-expression 0 represents the expression itself.

NOTE
Using Pattern Statistics in regular expressions mode within a selection (either single-range, or multiple) is not supported.

\section*{Descriptive Statistics and Entropy Analysis}

In addition to calculating two types of statistics, Hex Editor Neo also calculates some cumulative values for statistics results.

\section*{Descriptive Statistics}

Descriptive Statistics is calculated for both General Statistics and Pattern Statistics results and is a standard statistical calculation for a random value distribution. The following values are calculated:
- Mean
- Sample Variance
- Standard Deviation
- Standard Error
- Kurtosis
- Skewness
- Median
- Mode (one or several)
- Range
- Minimum value
- Maximum value
- Sum

\section*{Entropy Analysis}

Entropy analysis is performed for General Statistics mode only. The single-byte ( 256 different symbols) alphabet is used in entropy calculations. The following values are computed:

\section*{Entropy}

Entropy value in bits and as percentage from maximum (which is 8 for selected alphabet). The higher entropy, the less redundancy in the file

\section*{Redundancy}

Redundancy equals to \(100 \%\) - Entropy as percentage. Indicates the data redundancy in a file (or selection).

\section*{Compression size limit}

Theoretical compressed size limit. Again, as a simple single-byte alphabet is used in computing entropy, this "theoretical" limit may be much larger than the real compressed file size, produced by any modern compressor, as they use advanced alphabet selection strategies.

\section*{Attributes}

\section*{Attributes Tool Window}

The Attributes Tool Window displays the current file's attributes. It also allows you to modify the state of some attributes.


The following attributes are displayed for files and file streams:
\begin{tabular}{lll}
\hline Attribute & Description & Changeable? \\
\hline File Name & Full file's path & NO \\
\hline Archive & File is ready to be archived (file has been modified since last archival) & YES \\
\hline Hidden & File is hidden & YES \\
\hline System & File is system & YES \\
\hline Read-Only & File is read only & YES \\
\hline Sparse & File is a sparse file & NO \\
\hline Temporary & File is a temporary & NO \\
\hline Offline & File is offline & NO \\
\hline Encrypted & File is encrypted & NO \\
\hline Compressed & File is compressed & NO \\
\hline Has Reparse Points & File has reparse points & NO \\
\hline Created Date/Time & File's creation date and time & YES \\
\hline Last Access Date/Time & File's last access date and time & YES \\
\hline Last Write Date/Time & File's last write date and time & YES \\
\hline Hard Links & A number of file's hard links & NO \\
\hline Streams & A number of alternate data streams in a file & NO \\
\hline Total Size (with streams) & A total size a file occupies on the disk & NO \\
\hline Total Size (without streams) & A size of the file's unnamed (default) stream & NO \\
\hline Total Streams Size & A total size of all file's named streams & NO \\
\hline
\end{tabular}

The following attributes are displayed for opened processes:
\begin{tabular}{lll} 
Attribute & Description & Changeable? \\
\hline Process Name (ID) & The process name and process ID & NO \\
\hline File Path & Full path to process' executable & NO \\
\hline Number of Blocks & Total number of opened memory blocks & NO \\
\hline Number of Modules & Total number of opened modules & NO \\
\hline Start & Starting memory address & NO \\
Size & Size of opened range & NO \\
End & Ending memory address & NO \\
\hline
\end{tabular}

To a change a flag value, click on the checkbox. To change a date/time value, double-click on it. The following window appears:
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline \multicolumn{6}{|l|}{Change Time} & \(x\) \\
\hline & 8/1947 & 4: & 58:02 P & & & - \\
\hline \multicolumn{7}{|c|}{May, 1947} \\
\hline \multicolumn{7}{|l|}{Sun Mon Tue Wed Thu Fri Sat} \\
\hline \multicolumn{7}{|l|}{\(\begin{array}{rrrrrrrr}27 & 28 & 29 & 30 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 & 8 & 9 & 10\end{array}\)} \\
\hline \multicolumn{7}{|l|}{\(\begin{array}{lllllll}11 & 12 & 13 & 14 & 15 & 16 & 17\end{array}\)} \\
\hline \multicolumn{7}{|l|}{\(\begin{array}{lllllll}18 & 19 & 20 & 21 & 22 & 23 & 24\end{array}\)} \\
\hline \multirow[t]{3}{*}{} & 26 & & 28 & & 30 & 31 \\
\hline & 2 & 3 & 4 & & 6 & 7 \\
\hline & & & oday: & 8/31/ & & \\
\hline
\end{tabular}

To apply changes you made to a file's attributes, execute the Tools » Attributes » Apply Changes command. The Tools » Attributes » Reset Attributes command may be used to undo all attribute changes.

\section*{Base Converter}

\section*{Base Converter Tool Window}

The Base Converter Tool Window provides you with a mechanism to convert numbers from one format into another.


The following formats and encodings are supported by the Base Converter:
- Decimal
- Hexadecima
- Octal
- Binary
- Float (single-precision floating-point number)
- Double (double-precision floating-point number)
- ASCII character
- UNICODE character
- EBCDIC character

Once you enter the value in one of the supported formats, it gets immediately converted into other compatible formats. Use the Reset button to clear all fields. Hexadecimal values may be entered with or without the ex prefix.

\section*{Data Inspector}

The Data Inspector Tool Window allows you to interpret data under the cursor in several formats. For each format, decoded data is displayed in corresponding field in a tool window. As the cursor is moved, the window updates all fields. In addition, it can be instructed to highlight the currently selected field in an editor window, thus showing you all the cells the highlighted value occupies in the document.
\begin{tabular}{|c|c|c|}
\hline Data Inspector & & \(\times\) \\
\hline \(\Leftrightarrow\) E 國 & ఏके बहे बी & \\
\hline Type & Value & \\
\hline Standard types & & \(\wedge\) \\
\hline BYTE & 115 & \\
\hline USHORT & 52339 & \\
\hline UINT & 573230195 & \\
\hline ULONGLONG & 7521971604120325235 & \\
\hline char & \(s\) & \\
\hline wchar_t & 韭 & \\
\hline SHORT & -13197 & \\
\hline INT & 573230195 & \\
\hline Longlong & 7521971604120325235 & \\
\hline float & \(2.31475305 \mathrm{e}-18\) & \\
\hline double & 7.0850722544852002e+194 & \\
\hline Filetime & Tue, 07 Mar 25437 15:40:12 GMT & \\
\hline time_t (64-bit) & Invalid Date & \\
\hline time_t (32-bit) & Tue, 01 Mar 1988 14:36:35 GMT & \\
\hline signed time_t (64-bit) & Invalid Date & \\
\hline signed time_t (32-bit) & Tue, 01 Mar 1988 14:36:35 GMT & \\
\hline dosdate & N/A & \\
\hline dostime & N/A & \\
\hline RGB & 115, 204, 42 & \\
\hline
\end{tabular}

For example, in Hex Bytes view type, selecting the ULONGLONG fields in the Data Inspector window highlights the cursor cell and next 7 cells, as 64 -bit ULONGLONG value occupies 8 bytes.

You can configure the highlight colors using the Tools » Data Inspector » Modify Colors... command. Highlighting is toggled on or off with a Tools » Data Inspector » Enable Highlighting command.

Data Inspector allows you to define your own formats. See the Creating New Format section for more information.
Fields

The following formats are automatically installed with Hex Editor Neo:
\begin{tabular}{|c|c|}
\hline Format & Description \\
\hline BYTE & 8 -bit unsigned integer value ranged from 0 to 255 (0 to 0xFF) \\
\hline USHORT & 16-bit unsigned integer value ranged from 0 to 65,535 (0 to 0xFFFF) \\
\hline UINT & 32 -bit unsigned integer value ranged from 0 to 4,294,967,295 ( 0 to 0xFFFF FFFF) \\
\hline ULONGLONG & 64-bit unsigned integer value ranged from 0 to \(18,446,744,073,709,551,615\) ( 0 to 0xFFFF FFFF FFFF FFFF) \\
\hline char & 8-bit ASCII character \\
\hline wchar_t & 16-bit UNICODE character \\
\hline SHORT & 16-bit signed integer value ranged from -32768 to 32767 (0x8000 to 0x7FFF) \\
\hline INT & 32 -bit signed integer value ranged from \(-2,147,483,648\) to \(2,147,483,647\) ( \(0 \times 80000000\) to \(0 \times 7\) FFF FFFF) \\
\hline LONGLONG & 64-bit signed integer value ranged from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (0x8000 000000000000 to 0x7FFF FFFF FFFF FFFF) \\
\hline float & 32-bit single-precision floating-point value \\
\hline double & 64-bit double-precision floating-point value \\
\hline
\end{tabular}

\section*{Data Modification}

In addition to displaying data in different formats, the Data Inspector allows you to modify data in different formats. In order to do it, activate any field and double click it to start editing. The Edit» Edit Cell command may also be used to start editing currently active field. After you finish editing a field, press the Enter key to commit changes. Entered data will be propagated to the active document at the cursor's position. The field type will be used to determine the number of cells affected by the change. This action will result in a Write command being added to the document's operation history

Hexadecimal values may be prefixed by the ex prefix. Character values may be entered either inside quotation marks, or without them. You may also use the Structure Viewer expression when entering a field value.

User-defined data format must include the special "member function" assign to successfully support data modification. See the Creating New Format section for more information.

\section*{Byte Order}

Data Inspector respects the current editor window's byte order to display and process data in the list.

\section*{Creating New Format}

Data Inspector allows you to define your own formats in addition to standard ones. Structure Viewer language definition format is used to define the format


\section*{Use the following steps to define a new format}
1. Use Tools » Data Inspector » Add New Type... command to open the Type Definition window.
2. Enter the new type's name. 3, Enter the new type's definition. Definition must be a Structure Viewer's public user-defined type with optional display attribute.

If data modification is required, the defined public type must also have a special "member" function called assign. This function takes a single parameter and must convert it and assign to defined type's field(s).

The following example defines new Data Inspector type which displays an IP address
```

C++}\mathrm{ // display attribute allows IP address formatting
[display(format("{0}.{1}.{2}.{3}", a, b, c, d))]
public struct IP
{
// Little-endian IP address encoding
unsigned char d, c, b, a;
// assign function allows data modification through the user interface
function assign(value)
var na_end = int(find(value, "."));
var nb_end = int(find(value, ..., na_end + 1));
var nc_end = int(find(value, ".", nb_end + 1));
a = int(substring(value, 0, na_end))
b = int(substring(value, na_end + 1, nb_end - na_end - 1));
c= int(substring(value, nb_end + 1, nc_end - nb_end - 1));
int(substring(value, nc_end + 1));
};

```

\section*{Checksum Calculation}

Hex Editor Neo provides a number of checksum calculation algorithms. All algorithms are capable of quickly calculating the result for the wholedocument, or only part of it. All checksum calculation algorithms fully support multiple selection.

The following section, Checksum Tool Window, describes the user interface window used to configure and start checksum calculation. Below is a complete list of supported algorithms.

\section*{Simple summator}

This section contains the following algorithms: 8 -bit sum, 16 -bit sum and 32 -bit sum. It treats the document (or selection) as a stream of unsigned bytes and computes the sum of them. A summator is sized according to the selected algorithm and overflow may occur during calculation.

\section*{More summators}

This section contains the algorithms that treat the document as a stream of 8,16,32 or 64-bit unsigned little-endian or big-endian numbers. Each algorithm calculates the simple 64-bit sum of all values.

\section*{Signed sums}

This section has the same algorithms as in previous section, but each algorithm now operates on signed numbers.

\section*{Checksum}

The following algorithms are supported:
\begin{tabular}{|c|c|}
\hline Algorithm & Notes \\
\hline CRC-16 & Cyclic redundancy check. CRC polynomial: \(x^{16}+x^{15}+x^{2}+1\). Used in XMODEM, USB, many others. \\
\hline CRC-16 (CRCCCITT) & Cyclic redundancy check. CRC polynomial: \(x^{16}+x^{12}+x^{5}+1\). Used in X. \(25, \mathrm{~V} .41\), Bluetooth, PPP, IrDA, BACnet \\
\hline CRC-32 & Cyclic redundancy check. CRC polynomial: \(x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+\) 1. Used in V.42, MPEG-2. \\
\hline CRC XMODEM & Cyclic redundancy check. CRC polynomial: \(x^{16}+x^{12}+x^{5}+1\). Used in XMODEM. \\
\hline Custom CRC & Allows you to specify the polynomial values used in checksum calculation. See the Configuring Custom CRC section for more information. \\
\hline Adler-32 & Adler-32 checksum. Used in zlib and others. \\
\hline Fletcher & RFC 1146. Fletcher checksum. \\
\hline
\end{tabular}

\section*{Fowler-Noll-Vo (FNV) Hashes}

The following algorithms are supported:
\begin{tabular}{ll} 
Algorithm & Notes \\
\hline FNV-1 32-bit & 32-bit FNV-1 hash \\
\hline FNV-1a 32-bit & 32-bit FNV-1a hash \\
\hline FNV-1 64-bit & 64-bit FNV-1 hash \\
\hline FNV-1a 64-bit & 64-bit FNV-1a hash \\
\hline Custom FNV-1 & FNV-1 (a) hash with custom parameters \\
\hline
\end{tabular}

\section*{Hashes}

Note that actual list of algorithms in this section depends on version and localization of operating system. Some of the following algorithms may not be available in old or localized versions of operating system.
\begin{tabular}{ll} 
Algorithm & Notes \\
\hline MD2 & RFC 1319. Digest size is 128 bits. \\
MD4 & Digest size is 128 bits. \\
\hline MD5 & Digest size is 128 bits. \\
\hline SHA-1 & RFC 3174. Digest size is 128 bits. \\
\hline SHA-256 & Not supported on Windows XP \\
\hline SHA-384 & Not supported on Windows XP \\
\hline SHA-512 & Not supported on Windows XP \\
\hline RMD-160 & RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) -160 -bit message digest algorithm. \\
\hline RMD-128 & RIPEMD-128 (RACE Integrity Primitives Evaluation Message Digest) -128 -bit message digest algorithm.
\end{tabular}

\section*{Checksum Tool Window}

Checksum Tool Window is used to display checksum calculation results. It lists all algorithms supported on the current platform.
\begin{tabular}{|c|c|c|c|}
\hline Checksum & & & \\
\hline \multicolumn{4}{|l|}{} \\
\hline \(\square\) Algorithm & Value (Hex) & Value (Dec) & Parameters \\
\hline Fowler-Noll-Vo (FNV) Hashes - & & & \(\wedge\) \\
\hline - FNV-1 32-bit & 0x60039f1 & 100,678,129 & \\
\hline - fNV-1a 32-bit & 0x29a9b685 & 698,988,165 & \\
\hline - FNV-1 64-bit & 0xf450e8ebeeffc8f1 & 17,604,827,043,41... & \\
\hline - FNV-1a 64-bit & 0x6184fd868ca259c5 & 7,027,020,072,931,... & \\
\hline Custom FNV-1 & N/A & N/A & FNV-1 32-bit: Pr... \\
\hline Hashes & & & - \\
\hline - Secure Hash Algorithm (SHA-1) & \(77 \mathrm{a0}\) a5 Oc c1 ba 43 4e 9c 2893 6f 20718 Ba 91 3... & 11916016512 19... & \\
\hline - Message Digest 2 (MD2) & d3 29 da 04 d 7284414 ef 772 f 2 f 158480 d 8 & 211412184215 4... & \\
\hline Message Digest 4 (MD4) & cf d7 02 1e 3156 ea \(5 \mathrm{c} 634 \mathrm{fc3} 35624 \mathrm{~b} 90 \mathrm{f4}\) & \(2072152304986 . .\). & \\
\hline \(\square\) Message Digest 5 (MD5) & c8 7327049 gfa 74 e5 9b fd 647 fb 5 cfc 9 Ca 86 & 200115394159 2... & \\
\hline
\end{tabular}

Use the checkbox near the algorithm name to specify which checksums you want to compute. Choose whether you want selected checksums to be calculated for the whole document ("Whole Document" type) or for a current selection ("Selection Only" type). Click the Refresh toolbar button to start checksum calculation.

\section*{Displaying Results}

As soon as checksum calculation completes, results for all computed checksums are displayed in the list. Checksum's value is displayed both as hexadecimal and decimal number. The shortcut menu may be used to copy the results to the Clipboard or export them to a comma-separated file. You can also double-click the individual result to bring up the Result window where you can easily view and copy the checksum value.

\section*{Adding, Removing and Restoring Algorithms}

Hex Editor Neo supports algorithms which provide one or more customization parameters. Currently Custom CRC and Custom FNV-1 algorithms support this. You may have multiple entries of such algorithms in a list, each with its own set of parameters.

To add new algorithm, use the Tools » Checksum » Add Algorithm... command. To configure parameters for selected algorithm, use the Tools » Checksum » Parameters... command. To remove algorithm, use the Tools "Checksum » Remove Algorithm command. To reset the list of algorithms (including the default checkboxes), use the Tools » Checksum » Reset Algorithms command.

All commands are available through the main menu, Checksum Tool Window toolbar or context menu.

\section*{Performance Considerations}

Hex Editor Neo is capable to utilize several processors or processor cores when multiple algorithms are selected. Most algorithms (especially summators) are extremely fast. The slowest algorithms are Custom CRC and MD2. When several algorithms are started on multi-core computer, the total execution time will be limited by the speed of the slowest algorithm.

Fast algorithms, executed on file opened from fast HDD or SSD are parallelized almost linearly.

\section*{Configuring Custom CRC}

Custom CRC Parameters window is used to configure the Custom CRC checksum algorithm.
\begin{tabular}{|c|c|c|c|}
\hline \multicolumn{3}{|l|}{Custom CRC Parameters} & \(\times\) \\
\hline \multirow[b]{3}{*}{Initial Value:} & CRC-16 & O CRC-32 & ок \\
\hline & Hex: & Decimal: & \multirow[b]{2}{*}{Cancel} \\
\hline & fffffff & 4294967295 & \\
\hline Polynomial: & 4 c 11 db 7 & 79764919 & Res \\
\hline \multirow[t]{2}{*}{XOR Out:} & fffffff & 4294967295 & Help \\
\hline & Reflection In & - Reflection Out & \\
\hline Name: & Custom CRC & & \\
\hline
\end{tabular}

\footnotetext{
To open a Custom CRC Parameters window, use the Tools » Checksum » Parameters... command
}

Select the type of CRC to compute: either 16-bit or 32-bit. Enter the algorithm initial value, polynomial and XOR out constants. Reflection In and Reflection Out switches specify whether the algorithm should reflect bits on input and/or on output.

\section*{Configuring Custom FNV-1}

Custom FNV-1 Parameters window is used to configure the Custom FNV-1 checksum algorithm.
\begin{tabular}{|c|c|c|c|}
\hline \multicolumn{3}{|l|}{Custom FNV-1 Parameters} & \(\times\) \\
\hline & O 32-bit & 64-bit & ок \\
\hline \multirow[b]{2}{*}{Prime:} & Hex: & Decimal: & \multirow[b]{2}{*}{Cancel} \\
\hline & 1000193 & 16777619 & \\
\hline Offset basis: & 811c9dc5 & 2166136261 & Reset \\
\hline Version: & O FNV-1 & F FNV-1a & Help \\
\hline Name: & Custom FN & & \\
\hline
\end{tabular}

To open a Custom FNV-1 configuration dialog, use the Tools » Checksum » Parameters... command
Select the type of hash to compute: either 32-bit or 64-bit. Enter the algorithm prime, offset basis and select if you want to compute FNV-1 or FNV-1a version of the hash.
Specify the custom algorithm's name and press the OK button to save the changes.

\section*{Volume Navigator}

Starting from version 6.01 Hex Editor Neo includes a built-in Volume Navigator which is capable of parsing volume structure.
As of version 7.01, the following file systems are supported by Volume Navigator:
- NTFS (including support for some damages to file system)
- FAT
- FAT32
- ExFAT
- CDFS

\section*{Features}

Volume Navigator implements the following functionality:

\section*{Parsing volume structure}

An entire volume is parsed and represented in Volume Navigator Tool Window. This includes all directories, files, file streams, file attributes and so on.

\section*{Showing file record structure}

Whenever file record is selected in navigator's window, it is automatically parsed in Structure Viewer Tool Window.

\section*{Locating volume items}

A location (all clusters) an item occupies on a volume may be highlighted in opened editor window.

\section*{Copying volume items}

Any selected item (stream, file, directory or entire volume) may be copied.

\section*{Opening Volume}

Every time an supported volume is opened with File» Open» Open Volume... command, Volume Navigator automatically parses volume structure and displays it in its Volume Navigator Tool Window.

\section*{Volume Navigator Tool Window}
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{5}{|l|}{Volume Navigator} & \\
\hline \multicolumn{5}{|l|}{\multirow[t]{2}{*}{}} & \\
\hline & & & & & \\
\hline Name Size & Attribu... & Created & Modified & Accessed & \\
\hline - SymCache & & 4/5/2019 11:46:39 PM & 4/5/2019 11:46:39 PM & 4/5/2019 11:46:39 PM & \\
\hline - System Volume Informat... & HS & 3/29/2019 7:49:08 PM & 3/29/2019 7:49:08 PM & 3/29/2019 7:49:08 PM & \\
\hline SharedWithBuh & & 4/4/2019 11:27:54 AM & 4/4/2019 11:27:54 AM & 4/4/2019 11:27:54 AM & \\
\hline - Signing & & 4/4/2019 11:27:55 AM & 4/4/2019 11:27:55 AM & 4/4/2019 11:27:55 AM & \\
\hline - SVNRep & & 4/4/2019 11:27:55 AM & 4/4/2019 11:27:55 AM & 4/4/2019 11:27:55 AM & \\
\hline - TODO! VIDEO TO EDIT & & 4/4/2019 11:37:48 AM & 4/4/2019 11:37:48 AM & 4/4/2019 11:37:48 AM & \\
\hline - Users & & 3/31/2019 10:43:36 AM & 3/31/2019 10:43:36 AM & 3/31/2019 10:43:36 AM & \\
\hline - backups & & 4/4/2019 10:37:48 AM & 4/4/2019 10:37:48 AM & 4/4/2019 10:37:48 AM & \\
\hline - crash & & 4/4/2019 10:40:11 AM & 4/4/2019 10:40:11 AM & 4/4/2019 10:40:11 AM & \\
\hline - centrino2 & & 4/4/2019 10:40:10 AM & 4/4/2019 10:40:10 AM & 4/4/2019 10:40:10 AM & \\
\hline - dos & & 4/4/2019 10:43:12 AM & 4/4/2019 10:43:12 AM & 4/4/2019 10:43:12 AM & \\
\hline - exchange & & 3/30/2022 1:18:30 PM & 3/30/2022 1:18:30 PM & 3/30/2022 1:18:30 PM & \\
\hline - temp & & 4/4/2019 11:30:20 AM & 4/4/2019 11:30:20 AM & 4/4/2019 11:30:20 AM & \\
\hline - v & & 4/5/2019 9:17:23 AM & 4/5/2019 9:17:23 AM & 4/5/2019 9:17:23 AM & \\
\hline - wer & & A/5/2nio 0.17.25 AM & A/L/2010 0.17 .05 Am & \(415 / 7 \mathrm{O} 1000.17 .25 \mathrm{AM}\) & \\
\hline
\end{tabular}

Volume Navigator Tool Window contains a parsed file system structure. Each row represents a parsed file record. File records can be one of two types: directory record and file
record.
When you change selection, cursor location in current editor window is moved to the beginning of the selected file's record. In addition, Structure Viewer shows parsed file record.

The following information is displayed for each file record:
- Name
- Size (not available for directories)
- Attributes
- Creation Date
- Last Modification Date
- Last Access Date

Double-click a directory item to open it. Click the plus sign next to file entry to see all file's streams. Clicking on a stream navigates the current editor window to the beginning of stream's data.

\section*{Volume Navigator Commands}

The following commands are available on Volume Navigator toolbar or in item's context menu:

\section*{Up to Parent}

Navigate to parent directory.

\section*{Update Current View}

Rescan current directory.

\section*{Select All Stream Data}

Select all clusters that selected stream occupies. May produce multiple selection object if stream is fragmented.

\section*{Open File in Editor}

Open selected file in Hex Editor Neo.

\section*{Open Containing Folder}

Open Windows Shell and locate it to selected file.

\section*{Copy to..}

Copy selected item to a given location.

\section*{Copy Volume...}

Copy the entire volume based on parsed volume structure.

\section*{Built-In Explorer}

\section*{Explorer Tool Window}

This tool window mimics the Windows Explorer. It consists of two parts -Folder List and File List. You may hide a Folder List using the Folders button on the toolbar. Use the splitter bar between parts to change their relative sizes


Hex Editor Neo's Explorer behaves much like standard Windows Explorer. You may browse your file system, network neighborhood, open files in the editor by double-clicking on them, or dragging to the editor's window. You may also use drag\&drop to copy or move files and folders between different folders, as you do it in Windows Explorer. Right-clicking the file or folder brings up the standard file or folder shortcut menu.

Previous and Next commands may be used to navigate backward and forward (within a list of visited folders). Up command navigates to a parent folder. Locate Current command navigates to the location where the current document is situated. Find in Files command launches the Find in Files function in the current folder.

Two Explorer tool windows are present in the Hex Editor Neo. Each window has its own position, current folder and other settings.

\section*{Data Operations}

Hex Editor Neo supports a number of data operations that can be applied either to the whole document, or a current multiple selection. All operations are found under the Operations main menu item and are structurally divided into five categories: Bitwise Operations, Arithmetic Operations, Shift Operations, Case Change Operations and Reverse Operations.

All data operations execute in linear time and depend on the selection's complexity. That is, for single-range selection operation execution time does not depend on data size.

\section*{Bitwise Operations}

Bitwise operations include NOT, OR, AND and XOR. The first operation does not require an operand, while all others require the operand of a given size. More information is provided in the table below (in the table below x is a variable holding the value being processed):
\begin{tabular}{lll} 
Name & Operand & Operation Description \\
\hline NOT & N/A & \(x=\sim \mathrm{x}\) \\
OR & \(a\) & \(x=x \mid a\) \\
AND & \(a\) & \(x=x \& a\) \\
XOR & \(a\) & \(x=x \wedge a\)
\end{tabular}

The size of both operands depends on the view type of the current editor window.

When you execute a bitwise operation, the following window appears:


The operation you chose is automatically selected, but you can change it if you like.
If selected operation requires an operand, select its type and enter its value.

\section*{Using Patterns as Operands}

You can use a pattern for a bitwise operation (except for NOT). In this case, the operation iterates through each byte (or word etc., depending on view type) as the operation is applied to the selection.

\section*{Alignment}

Non-byte size operands are always aligned.

\section*{Scripting}

Scripts may execute bitwise operations using the IDocumentView.bitwiseOpAsync method.

\section*{Arithmetic Operations}

Arithmetic operations include Negation, Addition, Subtraction, Multiplication, Division and Remainder. The first operation does not require an operand, while all others require the operand of a given size. More information is provided in the table below (in the table below x is a variable that represents a current value):
\begin{tabular}{lll} 
Name & Operand & Operation Description \\
\hline Negation & N/A & \(x=-x\) \\
Addition & \(a\) & \(x=x+a\) \\
Subtraction & \(a\) & \(x=x-a\) \\
Multiplication & \(a\) & \(x=x * a\) \\
\hline Division & \(a\) & \(x=[x / a]\) \\
Remainder & \(a\) & \(x=x \% a\) \\
Set Minimum & \(a\) & \(x=x<a ? a: x\) \\
Set Maximum & \(a\) & \(x=x>a ? a: x\)
\end{tabular}

\section*{Arithmetic Operation Window}

When you execute an arithmetic operation, the following window appears:


The operation you chose is automatically selected, but you can change it if you like.
If selected operation requires an operand, select its type and enter its value.

\section*{Alignment}

Non-byte size operands are always aligned.

\section*{Scripting}

Scripts may execute bitwise operations using the IDocumentView.arithmeticOpAsync method.

\section*{Shift Operations}

Shift operations include Logical Left, Logical Right, Arithmetic Left, Arithmetic Right, Rotate Left and Rotate Right. All operations require an integer operand that specify the number of bits to shift or rotate. Arithmetic shift differs from logical shift in that it saves the sign of the operand. Only right logical and arithmetic shift perform differently and only if the value is negative

\section*{Shift Operation Window}

When you execute a shift operation, the following dialog appears:


The operation you executed is automatically selected, but you can change it if you like.
Enter the number of bits to shift or rotate.

\section*{Alignment}

Non-byte size operands are always aligned.
Scripting
Scripts may execute bitwise operations using the IDocumentView.shiftOpAsync method.

\section*{Case Change Operations}

Case change operations include Make Lowercase, Make Uppercase and Invert case. The result of an operation depends on the current window's view type. If view type is set to Hex Words or Decimal Words, selection is treated as Unicode string, otherwise it is treated as single-byte string. In the latter case, current window'sencoding is used to perform case change.

\section*{Scripting}

Scripts may execute bitwise operations using the IDocumentView.caseOpAsync method

\section*{Reverse Operations}

Reverse operations include Reverse Bits and Byte Swap operations. The actual behavior of both commands depends on the current editor window's group mode. For BYTE group mode, Byte Swap operation is not available.

When you execute a reverse operation, the following window appears
\begin{tabular}{|l|c|c|}
\hline 2 Reverse Operation & & \\
\hline Operation: & Reverse Bits & \\
\hline Operand type: & Words & OK \\
\hline O Current cell & Cancel \\
\hline Selection & & Help \\
\hline Whole file & & \\
\hline
\end{tabular}

The complexity of both commands is linear, depending on the number of ranges in a selection. For a single range selection, the complexity is always constant, not depending on selection size.

\section*{Reverse Bits}

Reverse Bits operation reverses the order of bits in each byte, word, double word or quad word in a current multiple selection.
- \(01011101_{2}=>10111010_{2}\)
- \(01011101^{\prime} 11100110_{2}=>01100111 ' 10111010_{2}\)

\section*{Byte Swap}

Byte Swap command reverses the order of bytes for each word, double word or quad word in a current multiple selection. This function is not available for a BYTE group mode
- \(1234_{16}\) => \(3412_{16}\)
- 1234 abcd \(_{16}=>\) cdab3412 \({ }_{16}\)

\section*{Scripting}

Scripts may execute bitwise operations using the IDocumentView.reverseOpAsync method.

\section*{Output Window}

Output Tool Window is a central window that displays some important diagnostic information, produced by various application components.
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{6}{|l|}{Output} \\
\hline Structure Errors & \(\checkmark\) & 8 & & & \\
\hline File & Type & Line & Document Name & Description & \\
\hline Compile Errors (1) & & & & & ^ \\
\hline C:|MyProjects|HexFrame|Sa.. & & 9 & & CE009: Syntax error: eoi expected & \\
\hline
\end{tabular}

At the top of the window is a switch which holds all installed components and a single Clear All button to clear the contents of the current window page.
Currently, it hosts the following pages.

\section*{Structure Errors}

Displays errors produced by Structure Viewer compilation of structure definition files.

\section*{Volume Errors \\ Displays volume parsing errors, produced by Volume Navigator}

\section*{Console}

Displays output of a currently running script or macro

\section*{Open Source Components}

Hex Editor Neo uses a number of open source components. This page lists all projects used and provides their licenses. HHD Software expresses its enormous gratitude to the authors and contributors of the following projects:

\section*{Scintilla}

A free source code editing component for Win32, GTK, and macOS. License is available at https://www.scintilla.org/License.txt.

License for Lexilla, Scintilla, and SciTE
Copyright 1998-2021 by Neil Hodgson <neilh@scintilla.org>
All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

\section*{TypeScript}

TypeScript is a language for application-scale JavaScript. TypeScript adds optional types to JavaScript that support tools for large-scale JavaScript applications for any browser, for any host, on any OS. TypeScript compiles to readable, standards-based JavaScript. https://github.com/microsoft/TypeScript/blob/main/LICENSE.txt

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.
"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.
"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License
"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this \(c\)
"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.
"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.
"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated docume "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaboratic "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is in "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Wo 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-fre 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, pr You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excludit If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained 1 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the te

\begin{abstract}
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BA 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate a 9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warr END OF TERMS AND CONDITIONS
\end{abstract}

\section*{ChakraCore}

ChakraCore is a JavaScript engine with a C API you can use to add support for JavaScript to any C or C compatible project. It can be compiled for x64 processors on Linux macOS and Windows. And x86 and ARM for Windows only. It is a future goal to support \(x 86\) and ARM processors on Linux and ARM on macOS.

ChakraCore is available under MIT License (https://github.com/chakra-core/ChakraCore/blob/master/LICENSE.txt).

\section*{The MIT License (MIT)}

Copyright (c) Microsoft Corporation
All rights reserved.
Copyright (c) 2021 ChakraCore Project Contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in al copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

\section*{Monaco Edito}

The Monaco Editor is the code editor that powers VS Code. It is available under MIT License (https://github.com/microsoft/monaco-editor/blob/main/LICENSE.txt).

The MIT License (MIT)

Copyright (c) 2016 - present Microsoft Corporation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

\section*{Boost}

Boost provides free peer-reviewed portable C++ source libraries. https://www.boost.org/users/license.html

\footnotetext{
Boost Software License - Version 1.0 - August 17th, 2003
Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.
}

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

\section*{yaml-cpp}
yaml-cpp is a YAML parser and emitter in C++ matching the YAML 1.2 spec. https://github.com/jbeder/yaml-cpp/blob/master/LICENSE.

Copyright (c) 2008-2015 Jesse Beder.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

\section*{Eigen3}

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. https://www.mozilla.org/en-US/MPL/2.0/.

\section*{Capstone}

Capstone is a disassembly framework with the target of becoming the ultimate disasm engine for binary analysis and reversing in the security community. https://github.com/capstone-engine/capstone/tree/next.

This is the software license for Capstone disassembly framework
Capstone has been designed \(\&\) implemented by Nguyen Anh Quynh
See http://www.capstone-engine.org for further information.
Copyright (c) 2013, COSEINC
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
* Neither the name of the developer(s) nor the names of its
contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\section*{Macros}

Several Hex Editor Neo components can be controlled automatically. Application programming interfaces they expose can be utilized by scripts running inside Hex Editor Neo.
This section describes macro support in the editor, also known as "internal scripting". Hex Editor Neo also supports so-calledexternal scripting, which allows external code to execute basic file editing algorithms without launching Hex Editor Neo.

Application Programming Interfaces (APIs) exposed for internal scripting and external scripting are not compatible between each other. External APIs must be compatible with OLE Automation and must support old version of JavaScript (pre-ES5) and therefore are quite limited. Internal APIs, on the other hand, are used by controlled environment inside the Hex Editor Neo, support latest versions of TypeScript and asynchronous execution.

The terms "scripts" and "macros" will be used interchangeably in this documentation section and will refer to scripts running inside the Hex Editor Neo.

\section*{Macro Support}

Currently scripts running inside Hex Editor Neo may do the following
- Control the general application tasks through the methods and properties of the Global Object. For example, the user may create new documents, open documents, activate and close windows, open settings pages and import or export application configuration settings.
- Control the opened documents through the methods and properties of the Document Object and Document View Object. Almost all document and editing actions may be performed using scripting.
- Control the Find in Files feature

The unique Macros» Start Recording command may be used to start automatic recording of user actions. When Macros » Stop Recording/Playback command is later executed, a new macro is created. The user may give a name to macro, set its hotkey and optionally open the macro' script for editing.

\section*{Editor Support}

The built-in script file editor allows the user to open script files and edit them. The editor provides full syntax coloring. It also supports automatic method completion, displays brief method and parameter documentation and lists all possible overloads, if any. The editor also supports placing and controlling break points.

\section*{Debugging}

Hex Editor Neo contains a built-in Script Debugger which will greatly simplify macro development. It supports asynchronous script breaking and breakpoints. When in break state, Debug Watch Tool Window and Debug Call Stack Tool Window may be used to study the current execution state.

\section*{Asynchronous Execution}

Many methods may potentially execute for a very long time, if they are instructed to operate on very large data block or complex multiple selection object.
By convention, names of all such methods end with Async and return a TypeScript Promise object. You generally has to await a result of an asynchronous operation before continuing.

Promise completion can either be successful (and optionally provide a result, unless it is Promise<void>) or erroneous, in which case an exception is thrown.
User script may use the await keyword to wait for completion and get the result of an asynchronous operation, or use the Promise.then or Promise.catch methods to schedule a continuation or error handler.

\section*{Awaiting Promises}
```

TypeScript
async function foo()
// await allows the control flow to be nice and "synchronous"
try
var occurrences = await activeView.findAllAsync(...);
alert(occurrences + " occurrences found")
} catch(e)
alert("Error occurred: " + e);
} }
function bar()
{ // traditional, "callback" style
indAllAsync(...);
operation.then(occurrences => {
alert(occurrences + " occurrences found");
}).catch(e => {
});

```

\section*{WARNING}

It is forbidden to start a new asynchronous operation for a document during the execution of a previous operation. Doing so will lead to an undefined behavior. However, you are allowed to start several asynchronous operations for different documents simultaneously without waiting for their completions.

\begin{abstract}
Scripts
Hex Editor Neo has the full-featured built-in script editor. You can open script files (TypeScript *.ts or JavaScript .js) in the editor using the File» Open » Open File... or File » Open » Open Script... commands, using Drag\&Drop from other applications, or create new empty script file using the File » New » New Script File command.

Macros »Start Recording command, followed by a Macros » Stop Recording/Playback command automatically generates a script file, which is then associated with a named macro. This script may also be opened in the editor for review and modification.

The script editor has the full knowledge of all application objects, their interfaces, their methods and parameters, provides syntax coloring, method and parameter completion and built-in brief documentation.

Syntax errors are highlighted directly in the editor, simplifying creation of a syntactically valid code.
Modified script files are marked with an asterisk (*).
\end{abstract}

\section*{Running Scripts}

You may start script execution using the Debug » Run Script command. Script debugging is started using the Debug » Debug Script command.
The results of the script execution is provided in the Output Window.

\section*{Working with Scripts}

The built-in Script Editor is a full-featured modern programming language editor, based on open-source Monaco Editor by Microsoft. It supports syntax coloring, syntax checking, auto-indentation, method and parameter completion and built-in documentation.

The editor supports a wide range of standard and extended text editing commands, most of which may operate on several document locations thanks to the support of multiple cursors.

In addition to standard text editing command, like Edit » Find, Edit» Replace, Edit» Go to and Clipboard commands like Edit » Cut Edit » Copy and Edit » Paste, editor provides the following special commands (available in the editor's shortcut menu):

\section*{Go to Definition}

Jump to the definition of the symbol under the cursor.

\section*{Peek Definition}

Temporary "peek" on the symbol definition.

\section*{Find All References}

Locate all references of a symbol under the cursor

\section*{Go to Symbol...}

Navigate to specified symbol.

\section*{Change All Occurrences}

Smart rename of a symbol.

\section*{Format Document}

Format the document or the current selection
A special Command Palette with additional extended editor commands may be opened using the Command Palette shortcut menu (Alt+F1).

\section*{Script Execution}

Script execution starts when the Debug » Run Script command is invoked. Hex Editor Neo performs the following actions when script starts executing:
1. TypeScript compiler is used to compile the TypeScript source into JavaScript. Any compilation errors at this stage are immediately displayed in the Output Window and highlighted in the editor. Moreover, most syntax errors are highlighted in the editor even before the script is started. If script compilation fails, script is not running.
2. Main script body is executed. If main code launches any non-asynchronous functions, they are completely executed. Any non-caught exceptions are displayed in the Output Window. All asynchronous functions as well as other delayed code may not complete execution at the time the main script body execution is finished. Uncaught exceptions terminate the script execution but do not put it into the stopped state. Delayed functions or asynchronous continuations may still continue to run.
3. Any delayed execution code as well as asynchronous continuations may execute at any time until the script is stopped.
4. Debug » Stop Script command must be used to stop script execution. This command completely stops script code execution and cancels any outstanding asynchronous continuations or delayed code. The script may also stop itself using the stopScript function.

\section*{Debugging}

Hex Editor Neo has a built-in script debugger. To start script debugging, execute the Debug » Debug Script command.

\section*{Break State}

To debug a script, it must be put into a break state. The user may put a breakpoint at any script line. When execution reaches a breakpoint, a script execution is paused and put into the break state

Alternatively, you can use the Debug »Break Execution command to immediately put the script into the break state.
When in break state, script is not executing. You can examine the current state using the Debug Watch Tool Window and Debug Call Stack Tool Window.
Use one of the following commands to continue execution:

\section*{Debug » Continue}

Continue script execution. Script execution continues either until the end of the main body, or until another breakpoint is reached.

\section*{Debug » Step In}

Execute a single script statement and stop. If the next statement is a function call, the script "enters" the function and stops.

\section*{Debug » Step Over}

Execute a single script statement and stop. If the next statement is a function call, the entire function is executed as a single statement.

\section*{Debug » Step Out}

Continues script execution until the control flows out of the current function and stop.

\section*{Debug Watch Tool Window}

Debug Watch tool window allows you to see values of the variables:


By default, Watch window displays this and local scope variables. In addition, the user may add additional variables to the list using the Add Watch command. Remove Watch command can be used to remove the variable and Edit Watch to edit it.

The script must be in break state, otherwise, the Watch window is blank.

\section*{Debug Call Stack Tool Window}

When script enters the break state, Call Stack window displays the current execution scope, as well as all parent scoped. You can see the name of the function as well as all function parameters.
\begin{tabular}{|ll|}
\hline Debug Call Stack & Line \\
\hline Name & 4 \\
macroBody & 11 \\
process \\
Global code & 18 \\
& \\
& \\
& \\
\hline
\end{tabular}

Double-clicking on the scope switches the context to this scope. Script window is updated to show you the call site and Debug Watch Tool Window is updated with new scope's locals.

Changing the context does not affect script execution.

\section*{Recording Macros}

Hex Editor Neo can "record" all applicable user actions into a script. To start recording, execute the Macros » Start Recording command.
When you finish, execute the Macros »Stop Recording/Playback command. If the recorded macro is not empty, a New Macro window appears.

\section*{Edit Macro Window}
\begin{tabular}{|l|c|}
\hline S. New Macro & \\
\hline Macro name: & \\
\hline \begin{tabular}{l|c|} 
& OK \\
\hline Macro hot key: & Edit Macro Script \\
\hline None & Cancel \\
\hline & Help \\
\hline
\end{tabular} \\
\hline
\end{tabular}

You need to enter the unique macro name and optionally specify the unique macro hot key. Press the OK button to finish or the Edit Macro Script button to finish and open the macro script for editing

\section*{WARNING}

While Hex Editor Neo tries to record as many actions as possible and as accurately as possible, some user interactions are impossible to record. It is therefore recommended to review and possibly fix script upon recording completion.

\section*{Macros Tool Window}

Macros Tool Window lists all macros currently present in the editor.
\begin{tabular}{|c|c|c|}
\hline Macros & & \(\times\) \\
\hline  & & \\
\hline Macro & Hot Key & \\
\hline Macro 1 & & \\
\hline Macro with a hot key & Ctrl+Shift+ & \\
\hline
\end{tabular}

Double-click the macro or use the Macros» Edit Macro... command to edit its name, hot key or script. Press the Del key or use the Macros » Delete Macro command to delete a macro or Macros » Delete All Macros command to remove all commands.

The full list of macros may be exported into an external file or imported from it using Macros » Export Macros... and Macros » Import Macros... commands correspondingly.

\section*{Macro Playback}

If the macro is assigned a hot key, you can press this hot key in order to execute the macro. Otherwise, use the Macros » Start Playback command.
See also Script Execution section for more information.

\section*{API Reference}

This section describes all internal objects, interfaces and enumerations, exposed by Hex Editor Neo components.
- Global Object
- Document Object
- Document View Object
- Process Object
- Process Window Object
- Process Snapshot Object

In addition, a lot of supplement interfaces and enumerations are use by mentioned main scripting objects. They are referenced in corresponding sections or provided later in this documentation

\section*{Global Object}

Global object has properties and methods that provide core functionality of Hex Editor Neo to running scripts. In addition, it provides a set of convenient script execution infrastructure methods.

The following methods are provided to help script authors to achieve their tasks in a more convenient and fast manner:
\begin{tabular}{ll} 
Method & Description \\
\hline log & Prints the message to the Console. \\
alert & Displays the message box and optionally asks for user choice. \\
input & Requests the user to enter text string. \\
async & Schedules a given callable for delayed execution. \\
cancelAsync & Cancels scheduled delayed execution. \\
delay & Returns a promise which is completed in a given time. \\
\hline
\end{tabular}

\section*{Creating New Documents}
newDocument method and its overloads are used to create new documents.

\section*{Opening Documents}

Shell object provides the following methods for opening various kinds of documents:
\begin{tabular}{ll} 
Method & Opens \\
\hline openFile & Binary files, Intel Hex and Motorola S-Records \\
openVolume & Volumes (logical disks) \\
openDisk & Disks (physical disks) \\
openHexAsync & Intel Hex and Motorola S-Records \\
\hline
\end{tabular}

\section*{Managing Opened Documents}
activate method activates a given document view, documents property returns a list of all opened documents, while closeAll method closes all opened documents.

\section*{Working With Settings}
exportSettings and importSettings exports and imports all application settings to an external file. exportConfiguration and importConfiguration does the same with UI settings like toolbars and tool windows layout. resetConfiguration resets the default UI configuration.
displaySettings method opens the Settings Window.
restart allows the script to restart Hex Editor Neo, optionally displaying a message to the user.

\section*{Find in Files, Replace in Files}
findInFilesAsync method starts the Find in Files operation and replacelnFilesAsync method starts the Replace in Files operation. cancelFindlnFilesOperation method may be used to cancel an ongoing Find in Files operation

\section*{Working with Processes}

Use the findProcess and findProcesses methods to locate the running process objects, getting the IProcess interface. Use the openProcess method to open the process window for a given process and get its IProcessWindow interface back. activeProcessWindow property holds the reference to an active process window or null, if there is no such window.

\section*{Events}
activeDocumentChanged event is fired when the active document is changed in the editor. activeViewChanged event is fired when the active editor window is changed.

\section*{Global Object Reference}

\section*{Description}

The global object defines several functions that are directly accessible to the script.

\section*{Declaration}
```

TypeScript
interface IGlobal
// Properties
readonly apiVersion: number;
readonly activeDocument: IDocument;
readonly activeView: IDocumentView
readonly wholefile: ISelection;
readonly documents: IDocument[];
// Methods
// Control Flow and Tracing Methods
log(message: any): void;
alert(message: any, params?: { title?: string; buttons?: AlertButtons; icon?: AlertIcon }): AlertResult;
input(message: string): string;
async(handler: () => void, ms: number, repetitive = false): number
cancelAsync(handlerId: number): void;
delay(ms: number): Promise<void>;
loadTextFile(path: string): string
// Pattern Object Creation Methods
createPattern(): IPattern;
createPattern(text: string, encoding?: Encodings): void
createPattern(data: number[], dataType = ValueType.Byte, bigEndian = false): IPattern
createPattern(data: Uint8Array | Uint16Array | Uint32Array): IPattern;
// New Document Creation Methods
newDocument(format?: string): IDocument;
// Document Opening Methods
openFile(path: string, mode = OpeningMode.Normal, readOnly = false, format?: string): IDocument
openVolume(path: string, friendlyName?: string, mode = OpenVolumeMode.ReadOnly): IDocument;
openDisk(path: string, friendlyName?: string, readOnly = true): IDocument
openHexAsync(path: string, format: HexFormat, flags = InsertHexFlags.None): IDocument
// Window Management Methods
activate(value: string | IWindow): void;
closeAll(closeMode = CloseMode.Prompt): boolean;
// General Application Control Methods
displaySettings(page?: string, hideOtherPages = false): void;
exportSettings(path?: string): boolean;
importSettings(path?: string): boolean
restart(message?: string): void;
exportConfiguration(path?: string): boolean
importConfiguration(path?: string): boolean
resetConfiguration(): void;
// Find in Files Methods
findInFilesAsync(pattern: IPattern | string
folders: null | string | string[],
params?: { fileMask?: string; flags?: FindInFilesFlags; subExpression?: number }): Promise<FindInFilesResult>
replaceInFilesAsync(search: IPattern | string
eplace: IPattern,
folders: null | string | string[],
params?: { fileMask?: string; flags?: FindInFilesFlags; subExpression?: number }): Promise<FindInFilesResult>
cancelFindInFilesOperation(): void;
// Process Methods
findProcess(processId: number): IProcess,
openProcess(process: number | IProcess, withSnapshot = false): IProcessWindow;
// Events
activeDocumentChanged(handler: (doc: IDocument) => void): number;
activeDocumentChanged(eventId: number): void;
activeViewChanged(handler: (view: IDocumentview) => void): number;
activeViewChanged(eventId: number): void;
}
// This interface is not available in managed environment
C++
// This interface is not available in native environment

```

\section*{Global Properties}

TypeScript
readonly apiVersion: number;

C\# // This property is not available in managed environment

C+
// This property is not available in native environment
Description
Returns the current API version. Equals \(0 \times 100\) in Hex Editor Neo 7.01

\section*{TypeScrip}
readonly activeDocument: IDocument;

C\# // This property is not available in managed environment
\(\frac{\text { c++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

The reference to the active document's document object. If there is no active document, this property holds null

\section*{activeView}

\section*{TypeScript \\ readonly activeView: IDocumentView;}

C\# This property is not available in managed environment
// This property is not available in native environment

\section*{Description}

The reference to the active editor window's document view object. If there is no active window, this property holds null.
wholefil

\section*{TypeScript \\ readonly wholeFile: ISelection}

C\# // This property is not available in managed environment
\(\frac{\text { c++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

Returns a special selection object that means "whole file".

\section*{documents}

\section*{TypeScript \\ readonly documents: IDocument[];}

C\# // This property is not available in managed environment
\(\frac{\text { C++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

Returns an array of all opened documents in the editor.

\section*{activeProcessWindow}

\section*{TypeScript \\ readonly activeProcessWindow: IProcessWindow; \\ C\# // This property is not available in managed environment \\ C++ This property is not available in native environment}

\section*{Description}

Returns a reference to an active process window or null if no such window exists.

\section*{Global Methods}

\section*{\(\log\)}

\section*{\(\frac{\text { TypeScript }}{\text { log(message }}\) \\ \(\log\) (message: any): void;}

C\# This method is not available in managed environment

\section*{C++}
\(\frac{\mathrm{C}++}{/ /}\) This method is not available in native environment

\section*{Parameters}

\section*{message}

A message to display. Accepts value of any type and automatically converts it to string.

\section*{Description}

Prints the message to the Console. Does not halt script execution.

\section*{Example}

TypeScript
\(\log (" T h e ~ v a l u e ~ o f ~ t h e ~ p a r a m e t e r ~ v ~=~ " ~+~ v) ; ~ ; ~\)
alert

TypeScript
Typescript (message: any, params?: \{ title?: string; buttons?: AlertButtons; icon?: AlertIcon \}): AlertResult;

C\# // This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}

\section*{message}

A message to display. Accepts value of any type and automatically converts it to string
params
Optional parameters include window title and combinations of buttons and icon to use. If title is omitted, default title is used. If buttons is omitted, a single OK button is displayed and if icon is omitted, no icon is displayed ina message box.

\section*{Return Value}

A button pressed by the user. Should be one of the values of AlertResult enumeration.

\section*{Description}

Display a message in a message box and return user's choice. Allows the script to customize the window title and buttons and icons used.

\section*{Example}
```

TypeScript
if (AlertResult.yes === alert("Do you want to execute the sequence of operations", { title: "Overwrite Prompt", buttons: AlertButtons.YesNo, icon: AlertIcon.Ques
{
// ..
}

```
input
TypeScript
input(message: string): string;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
message
The message to be displayed to the user.

\section*{Return Value}

The string entered by the user.

\section*{Description}

Displays a message to the user and asks him to enter the line of text. The method then returns the text returned by the user. The call to this method results in a message box to be displayed.

\section*{Example \\ ```
TypeScript \\ var name = input("Enter your name:");
``` \\ \(\log (\) "User name is " + name);}
```

TypeScript
async(handler: () => void, ms: number, repetitive = false): number;
// This method is not available in managed environment
C+
// This method is not available in native environment

```

\section*{Parameters}
handler
JavaScript function that takes no parameters and returns nothing. This function is invoked after ms milliseconds once or until cancelled, depending on the repetitive parameter.
ms
A number of milliseconds to wait until calling the passed function.
repetitive
An optional boolean that tells if async handler should be called once (repetitive is omitted or equals to false) or until cancelled (repetitive equals to true)

\section*{Return Value}

Returns an asynchronous function identifier. You may pass this identifier to cancelAsync method to cancel delayed function

\section*{Description}

Schedules a passed Javascript function for delayed execution. A caller may optionally specify if the async function should be repetitive

\section*{Example}

TypeScript lambda that is executed once after 1 second:

TypeScript
async(() => alert("Async handler executed"), 1000);
JavaScript function that is invoked every 2 seconds until cancelled after 20 seconds:
```

JavaScript
var h1 = async(function() { alert("Async handler executed"); }, 2000, true);
async(function() { cancelAsync(h1); }, 20000);

```

TypeScript
cancelAsync(handlerId: number): void;

C
/ This method is not available in managed environment

C+
// This method is not available in native environment

\section*{Parameters}

\section*{handlerId}

Async handler identifier to cancel.

\section*{Description}

Cancels pending async handler

\section*{Example}

JavaScript function that is invoked every 2 seconds until cancelled after 20 seconds
var h1 = async(function() \{ alert("Async handler executed"); \}, 2000, true);
async(function() \{ cancelAsync(h1); \}, 20000);
delay

TypeScript
delay(ms: number): Promise<void>

C\# This method is not available in managed environment
l/ This method is not available in native environment

\section*{Parameters}
ms
A number of milliseconds to wait until completing the returned promise object

\section*{Description}

Returns a Promise object that gets completed in a given number of milliseconds.

\section*{Example}

Using await:
```

TypeScript
async function test() {
await delay(500);

```
\}
Using continuations:
\(\frac{\text { TypeScript }}{\text { delay }(500) \cdot \text { then }(()=>\{\ldots\}) ;}\)
loadTextFile
    TypeScript
    loadTextFile(path: string): string;
// This method is not available in managed environment
C++

\section*{Parameters}
path
Full path to a text file.

\section*{Description}

Loads contents of a text file into a string
Example
Load text file into string and print it
```

JavaScript
log(loadTextFile("c:<br>temp<br>test.js"));

```
stopScript
\(\frac{\text { TypeScript }}{\text { stopScript(): void; }}\)

C\#
// This method is not available in managed environment

C++

\section*{Description}

Stops execution of a current script.

\section*{createPattern}

TypeScript
Typescript createPattern(): IPattern;

C\#
// This method is not available in managed environment

C++ This method is not available in native environment

\section*{Return Value}

An empty pattern object.

\section*{Description}

Creates an empty pattern object.

\section*{Example}

TypeScript
createPattern(text: string, encoding?: Encodings): void;
// This method is not available in managed environment

C++ This method is not available in native environment

\section*{Parameters}
text
Text string to be used as a pattern. encoding parameter specifies how Hex Editor Neo interprets this string.
encoding
Optional encoding to interpret the text string. Use Encodings.UTF16 to specify UTF-16 unicode string

\section*{Description}

Create a pattern from a given text string. Optional encoding parameter specifies how the editor interprets the given string.
createPattern

TypeScript
TypeScript
createPattern(data: number[], dataType = ValueType.Byte, bigEndian = false): IPattern;

C\# This method is not available in managed environment

C++
// This method is not available in native environment
Parameters
data
Pattern data values in a number array.
dataType
Optional value type. Tells the editor how to interpret values in a data array. Defaults to ValueType.Byte.
bigEndian
true to encode values from the data array as big-endian values, false otherwise. Defaults to false. Does not change data encoding if dataType equals ValueType.Byte.

\section*{Return Value}

A pattern object.

\section*{Description}

Create a pattern from a given number array. Optional dataType and bigEndian parameters change how the values in a data array are interpreted.

\section*{Example}

TypeScript
var pattern = createPattern([0x1234, 0x9876], ValueType.Word, true);

\section*{createPattern}

TypeScript
reatePattern(data: Uint8Array | Uint16Array | Uint32Array): IPattern;
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
data
Pattern data in a given format.

\section*{Return Value}

A pattern object
Description
Create a pattern from a given data. Data is used as-is, without any conversion.

\footnotetext{
TypeScript
ewDocument(format?: string): IDocument
}

C\# // This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method }}\) is not available in native environment

\section*{Parameters}

\section*{format}

Optional format for the new document
\begin{tabular}{ll} 
Format & Description \\
\hline "binary" & Create standard empty document. This is the default format. \\
\hline "intel-hex" & Create an encoded Intel Hex document. \\
\hline "motorola-hex" & Create an encoded Motorola S-Records document. \\
\hline
\end{tabular}

\section*{Return Value}

A document object for the created document

\section*{Description}

Create an empty document. Created document becomes an active document. A single view is also automatically created (it becomes an active view) and is opened in the editor.

\section*{Example}
```

TypeScript
var document = newDocument()

```
\(\log (\) "New document created.");
openfile

TypeScript
TypeScript
openFile(path: string, mode \(=\) OpeningMode.Normal, readOnly \(=\) false, format?: string): IDocument;

C\# This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
path
A full path to the file. File's extension is used to determine the type of the document to open.
mode
An optional opening mode
readOnly
true to open a file in read-only mode, false otherwise. Defaults to false
format
Optional format. If specified, forces the editor to try to open a file in a given format, overriding default behavior which analyzes the file's extension to determine format.
\begin{tabular}{ll} 
Format & Description \\
\hline "binary" & Open as standard binary document. \\
\hline "intel-hex" & Open as Intel Hex document. \\
"motorola-hex" & Open as Motorola S-Records document. \\
\hline
\end{tabular}

\section*{Return Value}

A document object.

\section*{Description}

Open a given file. If format is omitted, Hex Editor Neo analyzes file extension to determine the format of the file. .hex extension switches to the Intel Hex format and .s19 .s28 and .s37 extensions switch to the Motorola S-Records format.

Created document becomes an active document. A single view is also automatically created (it becomes an active view) and is opened in the editor.

\section*{Example}
ypeScript
var document = openFile("c:\\temp\\test.bin");
openVolum
```

TypeScript
openVolume(path: string, friendlyName?: string, mode = OpenVolumeMode.ReadOnly): IDocument
C\#}\mathrm{ // This method is not available in managed environment
// This method is not available in native environment

```

\section*{Parameters}
path
Full path to the volume.

\section*{friendlyName}

Optional volume "friendly name" to use in the editor.
mode
Volume opening mode. If omitted, defaults to OpenVolumeMode.ReadOnly.

\section*{Return Value}

A document object.

\section*{Description}

Open a given volume (logical disk) in the editor.
Created document becomes an active document. A single view is also automatically created (it becomes an active view) and is opened in the editor.

\section*{Example}

TypeScript
Typescript
var document \(=\) openVolume("\\\\? ?\VVolume\{7c1102f4-68e7-22e5-a511-80aa6f644963\}
(\", "Volume: \(\mathrm{C}: \backslash \backslash "\), OpenVolumeMode.ReadOnly);
openDisk(path: string, friendlyName?: string, readOnly = true): IDocument;

C\#
// This method is not available in managed environment

C++
C++ This method is not available in native environment

\section*{Parameters}

\section*{path}

Full path to the disk.
friendlyName
Optional disk "friendly name" to use in the editor
readonly
true to open the disk in read-only mode, false otherwise. If omitted, defaults to true.

\section*{Return Value}

\section*{A document object.}

\section*{Description}

Open a given disk (physical disk) in the editor.
Created document becomes an active document. A single view is also automatically created (it becomes an active view) and is opened in the editor.
Example
TypeScript
var document = openDisk("\\\\.\\PHYSICALDRIVE日", "First Physical Disk", true);

\section*{openHexAsync}

TypeScript
openHexAsync(path: string, format: HexFormat, flags = InsertHexFlags.None): IDocument;

C\#
// This method is not available in managed environment

C++ // This method is not available in native environment

\section*{Parameters}
path
Full path to the file. Specify null to take file contents from the Clipboard.
format
A format of the file you are opening. flags

One or more of the flags that modify the behavior of this method.

\section*{Return Value}

A document object.

\section*{Description}

Open a given encoded hex file in the editor. format is used to specify the format of the hex file and flags modify the file decoding behavior
Created document becomes an active document. A single view is also automatically created (it becomes an active view) and is opened in the editor.

\section*{Example}

TypeScript
var document = await openHexAsync(null, HexFormat.Intel, InsertHexFlags.IgnoreDataOffset);
activate

TypeScrip
activate(value: string | IWindow): void;

C\#
// This method is not available in managed environment
\(+\)
// This method is not available in native environment

\section*{Parameters}
value
A name of the window or a reference to a window object.

\section*{Description}

Activates a given window. This method takes a reference to the window to activate or a window name.
Example
TypeScript
activate("New Document 1");
/ activate the first view of the first document
activate(documents[0].views[0]);
closeAll

TypeScript
closeAll(closeMode = CloseMode. Prompt): boolean;

C\# // This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method is not available in native environment }}\)
Parameters

\section*{closemode}

An optional closing mode

\section*{Return Value}
true if operation was successful, false otherwise.

\section*{Description}

Close all opened windows. The optional closemode parameter is interpreted as follows:
\begin{tabular}{ll} 
Value & Description \\
\hline Prompt & Ask the user what to do with unsaved changes to documents. \\
Save & Force save any unsaved changes. \\
Discard & Discard all unsaved changes.
\end{tabular}

\section*{displaySettings}

TypeScrip
displaySettings(page?: string, hideOtherPages = false): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
page
Optional (non-localized) name of the page to display.

\section*{hideOtherPages}
true to hide all other pages besides page, false otherwise

\section*{Description}

Display the application Settings Window, optionally highlighting a given settings page

\section*{exportSettings}

TypeScript
exportSettings(path?: string): boolean;

C\#
/ This method is not available in managed environment
\(\frac{\mathrm{C}++}{\text { // This method is not available in native environment }}\)

\section*{Parameters}
path
Optional path of an external file where to write current application settings. If omitted, Hex Editor Neo asks the user to specify it.

\section*{Return Value}
true if the export was successful and false otherwise

\section*{Description}

Exports all application settings to a given file. If a full path is not given, an Save File window opens so the user may select the location of the export file

\section*{TypeScript}
importSettings(path?: string): boolean;

C\#

C+
/ This method is not available in native environment

\section*{Parameters}
path
Optional path of an external file where to read application settings. If omitted, Hex Editor Neo asks the user to specify it.

\section*{Return Value}
true if the import was successful and false otherwise.

\section*{Description}

Imports all application settings from a given file. If a full path is not given, an Open File window opens so the user may select the location of the file.
restart

TypeScript
restart(message?: string): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
message
Optional message to display to the user before restarting application

\section*{Description}

Restart Hex Editor Neo. Optional message, if specified, is displayed to the user before proceeding with a restart.
exportConfiguration

\section*{ypeScrip}
exportConfiguration(path?: string): boolean;

C\#
// This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method is not available in native environment }}\)

Parameters
path
Optional path of an external file where to write current UI configuration. If omitted, Hex Editor Neo asks the user to specify it.

\section*{Return Value}
true if the export was successful and false otherwise.

\section*{Description}

Exports the current UI configuration (toolbars and tool window layout) to a given file. If a full path is not given, an Save File window opens so the user may select the location of the export file.

\section*{importConfiguration}

\section*{TypeScript}
importConfiguration(path?: string): boolean;

C\#
// This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}
path
Optional path of an external file where to read application UI configuration. If omitted, Hex Editor Neo asks the user to specify it.

\section*{Return Value}
true if the import was successful and false otherwise.

\section*{Description}

Imports UI configuration (toolbars and tool window layout) from a given file. If a full path is not given, an Open File window opens so the user may select the location of the file.
resetConfiguration

\section*{TypeScript}
resetConfiguration(): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Description}

Resets the UI configuration to default state

\section*{findınfilesAsync}

TypeScript
findInFilesAsync(pattern: IPattern | string,
folders: null | string | string[],
operation: FindInFilesOperation
params?: \{ fileMask?: string; flags?: FindInFilesFlags; subExpression?: number \}): Promise<findInFilesResult>;

C\#
C\# This method is not available in managed environment

C++ \(/\) This method is not available in native environment

\section*{Parameters}

\section*{pattern}

A pattern or regular expression string to search. Use the createPattern method to create a pattern object.
folders
One or more folder paths that serve as the starting point for a search operation. If params.flags include the FindlnFilesFlags.IncludeSubFolders flag, sub-folders of a given folder(s) are also searched. If folders parameter is null, the operation is performed for opened documents only.

\section*{operation}

A type of Find in Files operation to perform
params
Optional operation parameters. fileMask, if omitted, defaults to "*". flags, if omitted, equals to FindlnFilesFlags.None.

\section*{Return Value}

A Promise object that produces a result of operation (as FindlnFilesResult object) when completed

\section*{Description}

Start the Find in Files operation. folders parameter specifies the starting search location. pattern specifies the pattern to search. The behavior of this method changes depending on the type of this parameter:

\section*{Type Description}

IPattern Standard matching mode is used. params.flags parameter may include the FindInFilesFlags.IgnoreCase flag to ignore patter case when matching.
string Regular expression mode is used. params.flags parameter may include the FindInFilesFlags.IgnoreCase and FindInFilesFlags.RegularExpressionUnicode flags. Optional params. subExpression, if set, specifies the regular expression's subexpression to use.

\section*{replacelnFilesAsync}

TypeScript
replaceInFilesAsync(search: IPattern | string,
replace: IPattern,
olders: null | string | string[],
params?: \{ fileMask?: string; flags?: FindInFilesFlags; subExpression?: number \}): Promise<FindInFilesResult>;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{search}

A pattern or regular expression string to search. Use the createPattern method to create a pattern object.

\section*{replace}

A pattern for use for replacing. Use the createPattern method to create a pattern object.
folders
One or more folder paths that serve as the starting point for a search operation. If params.flags include the FindlnFilesFlags.IncludeSubFolders flag, sub-folders of a given folder(s) are also searched. If folders parameter is null, the operation is performed for opened documents only.

\section*{operation}

A type of Replace in Files operation to perform.
params
Optional operation parameters. fileMask, if omitted, defaults to "*". flags, if omitted, equals to FindlnFilesFlags.None.

\section*{Return Value}

A Promise object that produces a result of operation (as FindInFilesResult object) when completed

\section*{Description}

Start the Replace in Files operation. folders parameter specifies the starting search location. pattern specifies the pattern to search. The behavior of this method changes depending on the type of this parameter:

Type Description
IPattern Standard matching mode is used. params.flags parameter may include the FindlnFilesFlags.IgnoreCase flag to ignore patter case when matching.
string Regular expression mode is used. params.flags parameter may include the FindlnFilesFlags.IgnoreCase and FindInFilesFlags.RegularExpressionUnicode flags. Optional params.subExpression, if set, specifies the regular expression's subexpression to use.
TypeScript
cancelFindInFilesOperation(): void;
C\# This method is not available in managed environment
C++
// This method is not available in native environment

\section*{Description}

Cancel a currently running Find in Files operation.
findProcess

TypeScript
TypeScript
findProcess(processId: number): IProcess;

C\# // This method is not available in managed environment

C++

\section*{Parameters}

\section*{processid}

Process ID of a running process.

\section*{Return Value}

A reference to a running process or null if there is no such process

\section*{Description}

Locate the running process object using the provided process ID.
findProcesses

\section*{TypeScript}
findProcesses(name: string): IProcess[];

C\# // This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}
name
A process name to match

\section*{Return Value}

An array of running process objects. If no match is found, the returned array is empty

\section*{Description}

Find all instances of a running processes that match the provided name. Name matching is case insensitive
openProces

TypeScript
openProcess(process: number | IProcess, withSnapshot = false): IProcessWindow;
C\#
This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}

\section*{process}

Process ID or a reference to a process object to open.

\section*{withSnapshot}

Set to true to automatically create a process snapshot

\section*{Return Value}

A reference to a created process window object.

\section*{Description}

Opens the given process and returns its process window object.

IGlobal Events

\section*{activeDocumentChanged}
ypeScrip
(handler: (doc: IDocument) \(\Rightarrow\) ) void): number;
activeDocumentChanged(eventId: number): void;

\section*{Parameters}

\section*{doc}

A reference to a document object that becomes active. May be null

\section*{Description}

This event is fired when the active document changes

\section*{activeViewChange}

TypeScript
activeViewChanged(handler: (view: IDocumentView) => void): number
activeViewChanged (eventId: number): void;

\section*{Parameters}
view
A reference to a document view object that becomes active. May be null

\section*{Description}

This event is fired when the active view changes

\section*{Document Objec}

Document object represents a document for a running script. Each opened document is represented as a single object instance. Document object implements the IDocument interface.

You can query and set object's properties and call methods to control the document instance.
Document.name property returns the current document's name (usually a file or device path). IDocument.size property returns the current size of a document. You can also change the size by simply assigning to this property

All document views can be accessed using the IDocument.views property.
Changes made to the document may be saved using IDocument.saveAsync and IDocument.saveAsAsync methods. Document history can be accessed using the Document.undo and IDocument.redo methods.

An ongoing asynchronous operation may be cancelled at any time by calling the IDocument.cancel method.
A document may be closed with a help of IDocument.close method.

\section*{Document Interface}
```

TypeScript
interface IDocument
// Properties
readonly name: str
readonly name: string;
readonly views: IDocumentView[];
// Methods
saveAsync(forceBackup = false): Promise<void>;
saveAsAsync(path: string, format?: string): Promise<void>;
undo(steps = 1): void;
redo(steps = 1): v
close(closeMode = CloseMode.Prompt): boolean;
}
C\#
// This interface is not available in managed environment
C++

```

\section*{Document Properties}

\section*{name}

\section*{TypeScript \\ readonly name: string}

C\# // This property is not available in managed environment

C++ // This property is not available in native environment

\section*{Description}

Returns current document's name. Maybe empty if the document was created using newDocument method. The following information is returned, depending on document's type:
\begin{tabular}{ll} 
Type & Name \\
\hline File, encoded file & Full path to the file. \\
Volume, physical disk & Either the full path to device, or device's friendly name.
\end{tabular}
size

\section*{TypeScrip}
size: number;

C\# // This property is not available in managed environment

C++ This property is not available in native environment

\section*{Description}

This property returns the current document size. You can assign a new value to this property to change the size of the document. This operation creates a record in document operation history.

\section*{Example}
```

ypeScript
var document = newDocument()
document.size $=100$;

```
view
\(\frac{\text { TypeScript }}{\text { readonly views: IDocumentView[]; }}\)

C\# // This property is not available in managed environment
// This property is not available in native environment

\section*{Description}

This property holds an array of all document views.

\section*{Example}
```

TypeScript
var document = newDocument()
activate(document.views[0]);

```

\section*{IDocument Methods}
saveAsync

TypeScript
saveAsync(forceBackup = false): Promise<void>;
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{forceBackup}

Set to true to force creation of a backup

\section*{Return Value}

A promise object that gets completed when save operation finishes. If save operation fails, promise is completed with an exception

\section*{Description}

Saves the changes made to the document. If the document does not have a name yet, a user is prompted to select the target.
saveAsAsyn

TypeScript
saveAsAsync(path: string, format?: string): Promise<void>;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{path}

A full path to the file where to save a document.
format
An optional format in which to save a file. If omitted, uses the original document format.

\section*{Return Value}

A promise object that gets completed when save operation finishes. If save operation fails, promise is completed with an exception

\section*{Description}

Saves a copy of the document to the specified file. Optional format parameter may override the original document format:
\begin{tabular}{ll} 
Format & Description \\
\hline "binary" & Save as a normal file. \\
\hline "intel-hex" & Save as Intel Hex encoded file. \\
\hline "motorola-hex" & Save as Motorola S-Records encoded file. \\
\hline
\end{tabular}

\section*{TypeScript \\ undo(steps = 1): void;}

C\#
// This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}
steps
The number of steps to undo

\section*{Description}

Undoes the given number of steps in the document's operation history
redo

TypeScrip
redo(steps = 1): void;

C\# This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}

\section*{steps}

The number of steps to redo.

\section*{Description}

Redoes (repeats) the given number of steps in the document's operation history.
cancel

TypeScript
cancel(): void

C\# This method is not available in managed environment

C+
// This method is not available in native environment

\section*{Description}

Cancel an ongoing asynchronous operation. If an asynchronous operation is actually running, it is completed with an exception.
close

TypeScript
close(closeMode \(=\) CloseMode.Prompt): boolean

C\# // This method is not available in managed environment

C++

\section*{Parameters}

\section*{closemode}

An optional closing mode.

\section*{Return Value}
true if the document has been closed successfully, false otherwise

\section*{Description}

Closes the document and all its document views. Optional closeMode parameter is used if the document has unsaved changes:
\begin{tabular}{ll} 
Value & Description \\
\hline Prompt & Ask the user to save changes. The user can choose "yes", "no" or "cancel". In the latter case, close returns false. \\
Save & Save unsaved changes. \\
Discard & Discard unsaved changes.
\end{tabular}

\section*{View Object}

Document View object is implemented by an editor window. If the document has several windows opened, each of them creates a separate document view object
Document View object implements the IDocumentView interface, which inherits from IWindow interface. You may get a reference to a document object this view is attached to using the IDocumentView.document property and you can get all views of a document using the IDocument.views property.

Other read-write view properties include:
\begin{tabular}{ll}
\hline Property & Description \\
\hline insertMode & Current Insert Mode. \\
\hline displayAs & Current View Type. \\
\hline addressDisplayAs & Current address display type. \\
\hline groupBy & Current Data Grouping mode. \\
\hline columns & Current number of columns \\
\hline bigEndian & Current Byte Order. \\
\hline encoding & Current encoding. \\
\hline codePane & Code pane visibility. \\
\hline textPane & Text pane visibility. \\
\hline
\end{tabular}

\section*{Cursor}

The view's cursor can be accessed using the read-only IDocumentView.cursor property. In addition, IDocumentView.goto can be used to execute the Edit » Go to Offset... command.

\section*{Selection}

Current view's selection can be accessed using the read-only IDocumentView.selection property. The following methods may be used to control the current selection:
\begin{tabular}{ll} 
Method & Description \\
\hline selectAll & Select all data. \\
\hline selectModifiedAsync & Select all modified data. \\
\hline copySelectionRangesAsync & Copy a list of selected ranges into the Clipboard. \\
\hline exportSelectionRangesAsync & Export a list of selected ranges to the external file. \\
\hline saveSelectionAsync & Save current selection to a file. \\
loadSelectionAsync & Load current selection from a file.
\end{tabular}

\section*{Clipboard}

Clipboard methods allow the view to communicate with the Clipboard:
\begin{tabular}{ll} 
Method & Description \\
\hline cutAsync & Cut selected data. \\
\hline copyAsync & Copy selected data. \\
\hline pasteAsync & Paste data from the Clipboard. \\
\hline clearAsync & Remove selected data. \\
\hline
\end{tabular}

\section*{Data Modification}

The following data modification methods are provided
\begin{tabular}{ll} 
Method & Description \\
\hline fillAsync & Fill selection with a pattern. \\
insert & Insert pattern. \\
\hline insertFile & Insert file. \\
insertHexAsync & Insert encoded hex file. \\
\hline encryptAsync & Encrypt selection. \\
\hline decryptAsync & Decrypt selection. \\
\hline modifyBits & Modify bits. \\
\hline
\end{tabular}

\section*{Find}

The following methods may be used to search for a pattern in a document
\begin{tabular}{ll} 
Method & Description \\
\hline findAsync & Find a pattern. \\
findRegExpAsync & Find a regular expression. \\
\hline findNextAsync & Find next occurrence. \\
\hline findPrevAsync & Find previous occurrence. \\
\hline findAllAsync & Find all occurrences of a pattern. \\
\hline findAllRegExpAsync & Find all occurrences of a regular expression. \\
\hline
\end{tabular}

\section*{Replace}

The following methods may be used to search and replace patterns:
\begin{tabular}{ll} 
Method & Description \\
\hline replaceAsync & Replace a pattern. \\
replaceRegExpAsync & Replace a regular expression. \\
replaceAllAsync & Replace all occurrences of a pattern. \\
replaceAllRegExpAsync & Replace all occurrences of a regular expression. \\
\hline
\end{tabular}

\section*{Data Operations}

The following data operations are provided:
\begin{tabular}{ll} 
Method & Description \\
\hline bitwiseOpAsync & Bitwise data operation. \\
\hline arithmeticOpAsync & Arithmetic data operation. \\
\hline shiftOpAsync & Shift data operation. \\
\hline caseOpAsync & Case change data operation. \\
\hline reverseOpAsync & Reverse data operation. \\
\hline
\end{tabular}

\section*{IDocumentView Interface}

\section*{Description}

This interface is implemented by a document view object. You can get a reference to an active document view using the activeView property. All views of a given document can be accessed using the IDocument.views property.

Declaration
```

TypeScript
interface IDocumentView extends IWindow {
// Properties
readonly document: IDocument;
readonly cursor: ICursor;
readonly selection: ISelection;
insertMode: boolean;
displayAs: DisplayAs
addressDisplayAs: AddressDisplayAs
groupBy: GroupBy;
columns: number;
bigEndian: boolean;
encoding: Encodings;
codePane: boolean;
textPane: boolean;
// Methods
activateCodePane(): void
activateTextPane(): void
editCell(value: string, shiftCursor = false): void
refresh(): void
connectWith(anotherView: IDocumentView): void
disconnect(): void;
lockUpdate(lock: boolean): void;
// Clipboard Methods
cutAsync(merge = false): Promise<void>;
copyAsync(merge = false): Promise<void>;
pasteAsync(textPane?: boolean): Promise<void>
learAsync(): Promise<void>;
// Selection Method
): void
selectModifiedAsync(). Promise<void>;
exportSelectionRangesAsync(fileName: string): Promise<void>;
saveSelectionAsync(fileName: string): Promise<void>;
loadSelectionAsync(fileName: string, loadType = SelectionLoadType.New): Promise<void>
// Navigation Methods
goto(offset: number, gotoMode: GotoMode, selectUpdateFlags = SelectionUpdateMode.NoChange): void;
fillAsync(pocument Modification Methods

```

```

    insert(pattern: IPattern, size?: number): void;
    insertHexAsync(path: string, format: HexFormat, flags = InsertHexFlags.None): Promise<void>;
    encryptAsync(providerName: string,
        providerType: number,
        algorithmp: number,
        password: string,
        range: ISelection)
        range: ISelection): Promise<void>;
    decryptAsync(providerName: string,
        providerType: number
        eyLength: number,
        password: string,
        range: ISelection): Promise<void>;
    \mathrm{ range: Iselection): Promise<}
    // Find Methods : IPattern, range: ISelection, flags = FindFlags.Forward): Promise<FindResult>;
    findRegExpAsync(pattern: string
        unicode: boolean,
        ubExpression = 0): Promise<FindResult>
    findNextAsync(): Promise<FindResult>;
    findPrevAsync(): Promise<FindResult>;
    findAllAsync(pattern: IPattern, range: ISelection, flags = FindFlags.Forward): Promise<number>
    findAllRegExpAsync(pattern: string
        nicode: boolean
        subExpression = 0): Promise<number>;
    // Replace Methods
    replaceAsync(command: ReplaceCommand,
        findPattern: IPattern,
        ange: ISelection,
        eplacePattern: IPattern,
        flags = FindFlags.None): Promise<boolean>
    replaceRegExpAsync(findPattern: string
        findUnicode: boolean,
        replacePattern: string | IPattern,
        flags = FindFlags.None,
        subExpression = 0,
        replaceRegExpUnicode = false): Promise<boolean>;
    replaceAllAsync(findPattern: IPattern,
        ange: ISelection,
        eplacePattern: IPattern,
    flags = FindFlags.None): Promise<number>;
    replaceAllRegExpAsync(findPattern: string,
        findUnicode: boolean,
            eplacePattern: string | IPattern,
            flags = FindFlags.None,
            subExpression = 0,
            replaceRegExpUnicode = false): Promise<number>;
    // Data Operations
    bitwiseOpAsync(operation: BitwiseOperation, pattern: IPattern, range: OperationRange): Promise<void>;
    arithmeticOpAsync(operation: ArithmeticOperation
        ange: OperationRange,
        value?: number,
    OperandSize): Promise<void>
    shiftOpAsync(operation: ShiftOperation
        range: OperationRange,
        value: number,
    caseOpAsync(operation: CaseOperation,
        ange: OperationRange,
        encoding: Encodings): Promise<void>
    reverseOpAsync(operation: ReverseOperation,
        range: OperationRange,
        operandSize?: OperandSize): Promise<void>
    ```
\}

C\# // This interface is not available in managed environment
\(\frac{\text { C++ }}{\text { // This interface }}\) is not available in native environment

\section*{DocumentView Properties}

\section*{document}

TypeScrip
readonly document: IDocument

C\#
// This property is not available in managed environment

C++

\section*{Description}

Returns a reference to a view's document object.
cursor
\(\frac{\text { TypeScript }}{\text { readonly cursor: ICursor; }}\)

C\# This property is not available in managed environment
\(\frac{\mathrm{C}++}{/ / \text { Th }}\)
// This property is not available in native environment

\section*{Description}

Returns a reference to a view's cursor object.

\section*{selection}

TypeScript
readonly selection: ISelection;

C\# This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Returns a reference to a view's selection object.

\section*{insertMode}

TypeScript
TypeScript
insertMode: boolean;

C\#
// This property is not available in managed environment

C++ This property is not available in native environment

\section*{Description}

Query or change the current view's insert mode.
displayAs

TypeScript
displayAs: DisplayAs;

C\#
// This property is not available in managed environment

C++ This property is not available in native environment

\section*{Description}

Query or change the current view's view type.
addressDisplayAs

TypeScript
TypeScrspDisplayAs: AddressDisplayAs;

C\#
// This property is not available in managed environment

C++
// This property is not available in native environment
Description
Query or change the current view's address display type
groupBy

TypeScrip
groupBy: GroupBy

C\# This property is not available in managed environment

C++

\section*{Description}

Query or change the grouping mode.
colums

TypeScript
TypeScript

C\# This property is not available in managed environment

C++ \(/\) This property is not available in native environment

\section*{Description}

Query or change the number of columns in an editor window. Set to 0 to switch to "auto columns" mode
currentColumns

TypeScript
\(\frac{\text { TypeScript }}{\text { readonly currentColumns: number; }}\)

C\#
// This property is not available in managed environment

C++ \(/\) This property is not available in native environment

\section*{Description}

Returns the actual number of columns. Equals columns unless it is 0 . In the latter case, returns the actual number of columns.
bigEndian

TypeScript
TypeScript bigEndian: boolean;

C\#
// This property is not available in managed environment

C++ This property is not available in native environment

\section*{Description}

Query or change the editor window's byte order.

TypeScript
encoding: Encodings;

C\#
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Query or change the current view's encoding. Set to Encodings.UTF16 to switch to the UNICODE mode
codePane
\(\frac{\text { TypeScript }}{\text { codePane: boolean; }}\)

C\#
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Switch the code pane on or off
textPane

TypeScript
textPane: boolean;

C\#
// This property is not available in managed environment

C++
C++ // This property is not available in native environment

\section*{Description}

Switch the text pane on or off.

\section*{IDocumentView Methods}
activateCodePane

TypeScript
activateCodePane(): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Description}

Activate the code pane (move cursor to the code pane).
activateTextPane

TypeScript
TypeScript
activateTextPane(): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Description}

Activate the text pane (move cursor to the text pane).
editCell

TypeScript
editCell(value: string, shiftCursor = false): void;

C\# \(/\) This method is not available in managed environment
\(\frac{\text { C++ }}{/ /}\) This method is not available in native environment

\section*{Parameters}

\section*{value}

New cell's value

\section*{shiftCursor}

Set to true to automatically shift cursor to the next cell.

\section*{Description}

Changes the value in the current cell. Used for decimal and floating-point value types.
typeChar

TypeScript
typeChar(text: string): void;

C\#
/ This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method }}\) is not available in native environment

\section*{Parameters}
text
The character to "type" into the view

\section*{Description}

Changes the value in the current cell. Used for hexadecimal, octal and binary value types.
refresh
\(\frac{\text { TypeScript }}{\text { refresh( })}\)
refresh(): void;

C\#
// This method is not available in managed environment
// This method is not available in native environment

\section*{Description}

Refresh the current view. Supported for volume, disk and process document types.
connectWith

\section*{TypeScript}
connectWith(anotherView: IDocumentView) : void;

C\#
This method is not available in managed environment

C++

\section*{Parameters}

\section*{anotherview}

A reference to a view you want to connect to. Cannot be equal to this

\section*{Description}

Connects two views. After connection, cursor movement and position are synchronized between the views. Use the disconnect method to break the connection. If one of the connected views is closed, the connection is also broken.

If the view has been previously connected, it is disconnected first.
Example
TypeScript
var doc1 = openFile("c: \\temp\\document1.bin");
doc1.views[0].connectWith(doc2.views[0]);

TypeScript
\(\frac{\text { TypeScript }}{\text { disconnect(): void }}\)

C\#
// This method is not available in managed environment

C++ \(/\) This method is not available in native environment

\section*{Description}

Disconnect the current view from any connected view. You don't have to disconnect the other "end" of a connection, it gets disconnected automatically.
lockUpdate

TypeScript
Typescript lockUpdate(lock: boolean): void;
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
lock
true to lock window updates and false to unlock.

\section*{Description}

Lock all updates to the document view's window (if lock is set to true) or unlock and refresh the window (if lock is set to false).

\section*{WARNING}

You must unlock the window before the script stops execution.

\section*{cutAsync}

\section*{TypeScript}
cutAsync(merge \(=\) false): Promise<void>;

C\# This method is not available in managed environment

C++

\section*{Parameters}

\section*{merge}
true to merge selection ranges together, false to leave them as is.

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Delete the selected ranges from the document and put them into the Clipboard. Optional merge parameter specifies whether multiple selection ranges should be joined into a single range.

\section*{copyAsync}

TypeScript
copyAsync(merge = false): Promise<void>;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
merge
true to merge selection ranges together, false to leave them as is.

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Copy selected document data into the Clipboard. Optional merge parameter specifies whether multiple selection ranges should be joined into a single range.

\section*{pasteAsync}

TypeScript
pasteAsync(textPane?: boolean): Promise<void>;

C\# This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}

\section*{textPane}
true to paste into the text pane, false to paste into the code pane. If omitted, the current pane is used
Return Value
A promise object that gets completed as soon as operation finishes.

\section*{Description}

Paste data from the Clipboard into the document.
clearAsync

\section*{TypeScript \\ clearAsync(): Promise<void>}

C\#
// This method is not available in managed environment
// This method is not available in native environment

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Deletes selected data from the document.
selectAll
\(\frac{\text { TypeScript }}{\text { selectAll(): void; }}\)

C\#
// This method is not available in managed environment

C++
/ This method is not available in native environment

\section*{Description}

Select the entire document.

\section*{TypeScript}
selectModifiedAsync(): Promise<void>;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

Return Value
A promise object that gets completed as soon as operation finishes.

\section*{Description}

Add all modified data ranges to a selection

\section*{copySelectionRangesAsyn}

TypeScript
copySelectionRangesAsync(): Promise<void>

C\# // This method is not available in managed environment
\(\frac{\text { C++ }}{/ /}\) This method is not available in native environment
Return Value
A promise object that gets completed as soon as operation finishes.

\section*{Description}

Copy the selection ranges information in text format into the Clipboard

\section*{exportSelectionRangesAsync}

\section*{TypeScript}
exportSelectionRangesAsync(fileName: string): Promise<void>;

C\#
// This method is not available in managed environment

C++ This method is not available in native environment

\section*{Parameters}

\section*{fileName}

The full path of the export file

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Export the selection ranges information in text format into the given file.
saveSelectionAsync(fileName: string): Promise<void>;

C\# This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
fileName
Full path to a file

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Save the current selection into the file.
loadSelectionAsync

TypeScript
\(\frac{\text { TypeScript }}{\text { loadSelectionAsync(fileName: string, loadType }=\text { SelectionLoadType.New): Promise<void>; }}\)

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{fileName}

Full path to a file
loadType
Selection loading mode. If omitted, defaults to SelectionLoadType.New

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Load a selection from a file and optionally combine it with a current selection, depending on the value of loadType parameter.
goto

TypeScript
goto(offset: number, gotoMode: GotoMode, selectUpdateFlags = SelectionUpdateMode.NoChange): void;

C\#
// This method is not available in managed environment

C++
\(\frac{\text { // This method is not available in native environment }}{\text { / }}\)

\section*{Parameters}

\section*{offset}

Target offset. The gotoMode parameter tells how to interpret this parameter.
gotoMode
Specifies how to interpret the value of a offset parameter.

\section*{selectUpdateFlags}

Specifies how the goto operation affects the current selection.

\section*{Description}

Move the cursor (and collapse all alternate cursors) to a given offset. The value of the offset parameter is interpreted according to the gotomode parameter. You can also specify how the operation affects the current selection with an optional selectUpdateFlags parameter. See the SelectionUpdateMode section for more information.

\section*{fillasync}

\section*{TypeScript}
fillAsync(pattern: IPattern, range?: ISelection, flags = FillFlags.None): Promise<void>;

C\#
C\# // This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
pattern
The pattern to fill. Use the createPattern function to create a pattern object.
range
Optional selection to fill. If omitted, the current selection is used. You can use the wholeFile global variable to specify the "whole file" fill mode.
flags
Fill operation flags.

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Fill the given selection (or a current selection, if range is null or omitted) with a given pattern. Optional flags parameter specify the operation mode.

\section*{insert}

\section*{TypeScript}
insert(pattern: IPattern, size?: number): void;

C\#
C\# // This method is not available in managed environment
\(\frac{\mathrm{C}++}{\text { // This method is not available in native environment }}\)

\section*{Parameters}

\section*{pattern}

The pattern to insert. Use the createPattern function to create a pattern object.
size
Optional pattern size. If the size is less than the size of a pattern, only part of a pattern is used, it the size is greater than the size of a pattern, a pattern is repeated. If omitted or equals to zero, the actual pattern's size is used.

\section*{Description}

Insert (or overwrite) a given pattern into the document. Insertion takes place at each multiple cursor location. The value of the IDocumentView.insertMode property is taken into account when performing this operation.

\section*{TypeScript}
insertFile(path: string, insertMode?: boolean): void;
// This method is not available in managed environment

C++
\(\frac{\text { C++ }}{\text { // This method }}\) is not available in native environment

\section*{Parameters}
path
Full path to the file to insert.

\section*{ner}
true to insert the file into the document, false to overwrite existing data. If omitted, the value of the IDocumentView.insertMode property is used.

\section*{Description}

Insert the contents of a given file at each multiple cursor location. The insertMode parameter tells if the existing data should be overwritten or not
\begin{tabular}{l} 
TypeScript \\
\begin{tabular}{l} 
insertHexAsync(path: string, format: HexFormat, flags = InsertHexFlags.None): Promise<void>; \\
\hline C\# \\
\hline This method is not available in managed environment \\
\hline C++ \\
\hline\(/\) This method is not available in native environment \\
\hline
\end{tabular} \\
\hline
\end{tabular}

\section*{Parameters}

\section*{path}

Full path to the file. Specify null to take file contents from the Clipboard.
format
A format of the file you are opening
flags
One or more of the flags that modify the behavior of this method.

\section*{Return Value}

A promise object that gets completed as soon as operation finishes.

\section*{Description}

Insert the (decoded) contents of a given encoded hex file in the editor. format is used to specify the format of the hex file and flags modify the file decoding behavior.

\section*{encryptAsync}

Typescrip
encryptAsync(providerName: string,
providerType: number,
algorithmId: number,
keyLength: number,
password: string,
range: ISelection): Promise<void>

C\#
// This method is not available in managed environment
+ This method is not available in native environment

\section*{Parameters}

\section*{providerName}

Cryptographic provider name

\section*{providerType}

Cryptographic provider type.

\section*{algorithmId}

Cryptographic algorithm to use

\section*{keyLength}

Key length. If algorithm supports single key length only, pass 0

\section*{password}

Encryption password

\section*{range}

Encryption range. Note that most algorithms require the total length to be a multiply of a given number

\section*{Return Value}

Promise object that get completed as soon as operation finishes.

\section*{Description}

Encrypt the given selection using specified cryptographic algorithm and using a passed password to generate encryption key.

\section*{decryptAsync}

Typescrip
decryptAsync(providerName: string,
providerType: number,
keyLength: number
password: string,
range: ISelection): Promise<void>

C\#
C\# This method is not available in managed environment

C++ This method is not available in native environment

\section*{Parameters}

\section*{providerName}

Cryptographic provider name

\section*{providerType}

Cryptographic provider type.
algorithmId
Cryptographic algorithm to use

\section*{keyLength}

Key length. If algorithm supports single key length only, pass \(\theta\).
password
Encryption password.
range
Decryption range. Note that most algorithms require the total length to be a multiply of a given number

\section*{Return Value}

Promise object that get completed as soon as operation finishes.

\section*{Description}

Decrypt the given selection using specified cryptographic algorithm and using a passed password to generate encryption key.

\section*{modifyBits}

\section*{TypeScript \\ modifyBits(mask: number): void}

C\#
/ This method is not available in managed environment

C++ // This method is not available in native environment

\section*{Parameters}

\section*{mas}

A mask to use for the operation

\section*{Description}

Modify the current cell with a given mask
findAsyr

\section*{TypeScript}
findAsync(pattern: IPattern, range: ISelection, flags = FindFlags.Forward): Promise<FindResult>

C\# // This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{pattern}

A pattern to find. Use the createPattern function to create a pattern object
range
Selection object to use for searching
flags

An optional flags that modify the behavior of the find operation

\section*{Return Value}

A promise object that gets completed when a next occurrence is found or the end of the file is reached.

\section*{Description}

Find the first occurrence of a given pattern in a given range. Optional flags parameter modifies the search. The method returns a promise that produces an instance of a FindResult object when completes. If no more occurrences have been found, the result is null.

\section*{findRegExpAsync}
```

TypeScript
indRegExpAsync(pattern: string,
unicode: boolean,
flags = FindFlags.Forward,
lol
C\#
// This method is not available in managed environment
// This method is not available in native environment

```

\section*{Parameters}

\section*{pattern}

A regular expression to search.
unicode
true to search for UTF-16 encoded text, false to use the current encoding.
flags
Optional flags that modify the search operation.
subExpression
An optional sub-expression of a regular expression to match. \(\theta\) means the full expression.
Return Value
A promise object that gets completed when a next occurrence is found or the end of the file is reached.

\section*{Description}

Find the first occurrence of a given regular expression pattern. Optional flags parameter modifies the search. The method returns a promise that produces an instance of a FindResult object when completes. If no more occurrences have been found, the result is null.
findNextAsync
```

ypeScript
findNextAsync(): Promise<FindResult>;

```

C\#
C\# This method is not available in managed environment

C++
C++ This method is not available in native environment

\section*{Return Value}

A promise object that gets completed when a next occurrence is found or the end of the file is reached.

\section*{Description}

Find the next occurrence of a pattern used in the previous search.
findPrevAsync

TypeScript
findPrevAsync(): Promise<FindResult>;
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Return Value}

A promise object that gets completed when a next occurrence is found or the beginning of the file is reached.

\section*{Description}

Find the previous occurrence of a pattern used in the previous search.
```

TypeScript
C\#
// This method is not available in managed environment
C++}\mathrm{ // This method is not available in native environment

```

\section*{Parameters}
```

pattern
A pattern to search for. Use the createPattern function to create a pattern object.
range
Selection object to use for searching.
flags
Optional flags that modify the search operation.

```

\section*{Return Value}

A promise object that gets completed when a all occurrences have been found. The promise result is the total number of occurrences

\section*{Description}

Locate all instances of a given pattern and select them in a document view. This methods returns a promise that produces a total number of occurrences when completed.

\section*{findAllRegExpAsync}

TypeScript
findAllRegExpAsync(pattern: string,
unicode: boolean,
flags = FindFlags. Forward,
subExpression \(=0\) ): Promise<number>;

C\#
This method is not available in managed environment

C++ This method is not available in native environment

\section*{Parameters}

\section*{pattern}

A regular expression to search for.
unicode
true to search for UTF-16 encoded text, false to use the current encoding.
flags
Optional flags that modify the search operation.

\section*{subExpression}

An optional sub-expression of a regular expression to match. 0 means the full expression.

\section*{Return Value}

A promise object that gets completed when a all occurrences have been found. The promise result is the total number of occurrences.

\section*{Description}

Locate all instances of a given regular expression pattern and select them in a document view. This methods returns a promise that produces a total number of occurrences when completed.

\section*{replaceAsync}

TypeScript
findPattern: IPattern,
range: ISelection,
replacePattern: IPattern,
flags = FindFlags.None): Promise<boolean>;

C\#

C++ This method is not available in native environment

\section*{Parameters}

\section*{command}

Replace command to perform.

\section*{findPattern}

A pattern to search. Use the createPattern method to create a pattern object.
range
Selection object to use for searching.
```

replacePattern
A pattern to replace. Use the createPattern method to create a pattern object.
flags
Optional flags to modify the command behavior.

```

\section*{Return Value}

A promise object that gets completed the when operation finishes. The promise yields a boolean that tells if the match was found.

\section*{Description}

Find the next occurrence of a findPattern if command equals ReplaceCommand.Find or replace the last found occurrence with a replacePattern if command equals ReplaceCommand.Replace.

\section*{TypeScript}
replaceRegExpAsync(findPattern: string,
findUnicode: boolean,
replacePattern: string | IPattern
flags = FindFlags.None,
subExpression \(=0\)
replaceRegExpUnicode = false): Promise<boolean>;
// This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method }}\) is not available in native environment

\section*{Parameters}

\section*{findPattern}

A regular expression string to search.

\section*{findUnicode}
true to treat search string as UTF-16 encoded and false to use the current view's encoding.

\section*{replacePattern}

A replace pattern to use. Can be a string or a pattern object.
flags
Optional flags that modify the command behavior.
subExpression
Optional sub-expression to match. 0 matches the whole expression.

\section*{replaceRegExpUnicode}

If replacePattern is a regular expression replace pattern, specifies if it has UTF-16 encoding

\section*{Return Value}

A promise object that gets completed when a match is found. The promise result is true if match is found and false otherwise.

\section*{Description}
command: ReplaceCommand
Find and replace a next match of a specified regular expression. A replace pattern may either be a pattern object or a special form of regular expression, called replace regular expression

\section*{replaceAllAsync}

\section*{TypeScript}
replaceAllAsync(findPattern: IPattern range: ISelection,
replacePattern: IPattern,
flags = FindFlags.None): Promise<number>
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{findPattern}

A pattern to search for. Use the createPattern method to create a pattern object.
range
Selection object to use for searching

\section*{replacePattern}

A pattern to replace. Use the createPattern method to create a pattern object.
flags
Optional flags that modify the command behavior.

\section*{Return Value}

A promise object that gets completed when a all occurrences have been found. The promise result is the total number of occurrences.

\section*{Description}

Find and replace all occurrences of a findPattern with a replacePattern. the next occurrence of a given regular expression. If match is found, it is replaced with a replacePattern, which can be either a pattern object or a replace regular expression string
replaceAllRegExpAsync
```

TypeScript
eplaceAllRegExpAsync(findPattern: string,
findUnicode: boolean,
eplacePattern: string | IPattern
flags = FindFlags.None
bExpression = 0
replaceRegExpUnicode = false): Promise<number>;
C\#
// This method is not available in managed environment
C++
// This method is not available in native environment

```

\section*{Parameters}

\section*{findPattern}

A pattern to replace. Use the createPattern method to create a pattern object.

\section*{findUnicode}
true to treat search string as UTF-16 encoded and false to use the current view's encoding.
replacePattern
A replace pattern to use. Can be a string or a pattern object.
flags
Optional flags that modify the command behavior.
subExpression
Optional sub-expression to match. \(\theta\) matches the whole expression.

\section*{replaceRegExpunicode}

If replacePattern is a regular expression replace pattern, specifies if it has UTF-16 encoding

\section*{Return Value}

A promise object that gets completed when a all occurrences have been found. The promise result is the total number of occurrences

\section*{Description}

Replace all occurrences of a given regular expression with either pattern object or a replace regular expression.

\section*{bitwiseOpAsync}
bitwiseOpAsync(operation: BitwiseOperation, pattern: IPattern, range: OperationRange): Promise<void>;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{operation}

Bitwise operation to perform
pattern
Parameter for some of the bitwise operations. Pass empty pattern for a bitwise NOT operation
range
Operation range.

\section*{Return Value}

A promise object that gets completed when the operation finishes

\section*{Description}

Perform a given bitwise operation. For all values of the operation parameter besides BitwiseOperation.Not, a non-empty pattern is required

\section*{Example}

TypeScript
TypeScript await activeView.bitwiseOpAsync(BitwiseOperation.Xor, createPattern(new Uint8Array([0xaa])), OperationRange.WholeFile);
```

TypeScript
arithmeticOpAsync(operation: ArithmeticOperation
range: OperationRange,

```
    range: Operatio
    operandSize?: OperandSize): Promise<void>;
C\#
C\# This method is not available in managed environment
c+
This method is not available in native environment

\section*{Parameters}

\section*{operation}

An arithmetic operation to perform.
range
Operation range.
value
Optional operand for an arithmetic operation. Required unless operation equals ArithmeticOperation.Negation.

\section*{operandSize}

Specifies the size of the operand. Defaults to current view's grouping mode.

\section*{Return Value}

A promise object that gets completed when the operation finishes.

\section*{Description}

Perform a given arithmetic operation. All arithmetic operations besides ArithmeticOperation.Negation require an operand.

\section*{Example}

TypeScript
TypeScript activeView.arithmeticOpAsync(ArithmeticOperation.Multiplication, OperationRange.WholeFile, 2, OperandSize.Word);

\section*{shiftOpAsync}
```

TypeScript
shiftOpAsync(operation: ShiftOperation,
range: OperationRange,
value: number,
operandSize?: OperandSize): Promise<void>;
C\#}\mathrm{ // This method is not available in managed environment
C++}\mathrm{ // This method is not available in native environment

```

\section*{Parameters}

\section*{operation}

A shift operation to perform.
range
Operation range
value
A number of bits to shift.

\section*{operandSize}

Specifies the size of the operand. Defaults to current view's grouping mode.
Return Value
A promise object that gets completed when the operation finishes.

\section*{Description}

Perform a given shift operation

\section*{Example}

TypeScript
await activeView.shiftOpAsync(ShiftOperation.RotateLeft, OperationRange.WholeFile, 3, OperandSize.Word);

\section*{caseOpAsync}
```

TypeScript
caseOpAsync(operation: CaseOperation,
range: OperationRange
encoding: Encodings): Promise<void>
C\#
// This method is not available in managed environment

```
```

C++
This method is not available in native environment

```

\section*{Parameters}

\section*{operation}

A case operation to perform.
range
Operation range.
encoding
A text encoding to use for case change operation
Return Value
A promise object that gets completed when the operation finishes

\section*{Description}

Perform a given case change operation.

\section*{Example}
ypescrip
await activeView.caseOpAsync(CaseOperation.Toggle, OperationRange.WholeFile, activeView.encoding);

\section*{reversepassme}

\section*{TypeScript}
reverseOpAsync(operation: ReverseOperation range: OperationRange operandSize?: OperandSize): Promise<void>;

C\#
// This method is not available in managed environment

C++ \(/\) This method is not available in native environment

\section*{Parameters}

\section*{operation}

A reverse operation to perform
range
Operation range.

\section*{operandSize}

Specifies the size of the operand. Defaults to current view's grouping mode.

\section*{Return Value}

A promise object that gets completed when the operation finishes

\section*{Description}

Perform a given bit or byte reverse operation

\section*{Example}

\section*{TypeScript}

TypeScript
await activeView.reverseOpAsync(ReverseOperation.Bits, OperationRange.WholeFile, OperandSize.Word)

IWindow IDocument ISelection Encodings IPattern

\section*{Process Object}

\section*{Process Object}

This object represents a running process. Use the findProcess and findProcesses methods to get a reference to the process object
Various running process properties are available via read-only properties IProcess.processId, IProcess.name and IProcess.commandLine
You can terminate a running process using the IProcess.kill method
Pass a reference to a process object to the openProcess method to get a IProcessWindow interface in order to obtain additional process properties, create and manage process snapshots, open process memory for editing, compare and debug process snapshots.

\section*{Process Interface}

\section*{Description}

This interface is implemented by a process object. Use the findProcess and findProcesses methods to get a reference to this interface.
```

TypeScript
interface IProcess
// Properties
readonly processId: number;
readonly name: string;
readonly commandLine: string;
// Methods
kill(): boolean
}
C\#}\mathrm{ // This interface is not available in managed environment
// This interface is not available in native environment

```

\section*{Process Properties}
processld

\section*{TypeScript}
readonly processId: number;

C\# This property is not available in managed environment
,
is property is not available in native environment

\section*{Description}

Process ID. This number is unique among all running processes.

NOTE
The system may reuse IDs after the process exits. This may happen when the process exits even if your script still holds a reference to a process object.
However, if you open the process using the openProcess method and store a IProcessWindow reference, the process ID will not be reused until process window is closed.
name

\section*{TypeScript}
readonly name: string;

C\#
// This property is not available in managed environment
(/ This property is not available in native environment

\section*{Description}

Process name

\section*{commandLine}

TypeScript
readonly commandLine: string;

C\#
// This property is not available in managed environment
\(\frac{\text { C++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

The full command line of a running process. This property may be empty if Hex Editor Neo does not have enough rights to obtain process's command line.

\section*{Process Method}

\section*{kill}
\(\frac{\text { TypeScript }}{\text { kill(): boolean; }}\)

C\# This method is not available in managed environment

C+
C++ This method is not available in native environment

\section*{Return Value}
true if termination was successful or false otherwise (usually this means that the Hex Editor Neo does not have enough rights to terminate a given process).

\section*{Description}

Terminate the process.

\section*{Process Window Object}

This object represents an opened process window. You get a reference to its IProcessWindow interface after calling the openProcess method. Use properties of this object to retrieve additional information about a running process and create and manage process snapshots.

IProcessWindow.processId, IProcessWindow.name, IProcessWindow.commandLine, IProcessWindow.path read-only properties retrieve information about the running process. IProcessWindow.isRunning property returns true if the process is still running and IProcessWindow.exitCode provides a process exit code.

IProcessWindow.snapshots property returns an array of all process snapshots. The snapshot with index zero always exists and represents the running process.
Call the IProcessWindow.createSnapshot method to create new process snapshot and IProcessWindow.deleteSnapshot to delete a snapshot. IProcessWindow.clearSnapshots method deletes all snapshots.

Call the IProcessWindow.compareSnapshots method to use the File Comparison feature to compare memory of the two process snapshots.

\section*{ProcessWindow Interface}

\section*{Description}

This interface is implemented by a process window object. Use the openProcess method to get a reference to this interface.

Declaration

TypeScript
interface IProcessWindow \{
// Properties
readonly processId: number;
readonly name: string;
readonly commandLine: string;
readonly path: string;
readonly isRunning: boolean
readonly exitcode?: number;
readonly snapshots: IProcessSnapshot[];
// Methods
deleteSnapshot(snapshot: IProcessSnapshot): void
clearSnapshots(): void;
compareSnapshots(snapshot1: IProcessSnapshot,
snapshot2: IProcessSnapshot,
arrange \(=\) SplitMode.Vertical): void;
\}

C\# This interface is not available in managed environment
// This interface is not available in native environment

\section*{ProcessWindow Properties}

\section*{processld}

TypeScript
readonly processId: number;

C\# This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Process ID. This number is unique among all running processes.
name

\section*{TypeScript \\ readonly name: string;}

C\# This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Process name
commandLine

TypeScript
readonly commandline: string
// This property is not available in managed environment

C++

\section*{Description}

The full command line of a running process. This property may be empty if Hex Editor Neo does not have enough rights to obtain process's command line.
path

TypeScript
readonly path: string;

C\# This property is not available in managed environment
++
property is not available in native environment

\section*{Description}

The full path to the process executable file.
isRunning

TypeScrip
readonly isRunning: boolean;

C\# This property is not available in managed environment

C++

\section*{Description}

Holds true while the process is running.
exitCode

TypeScript
readonly exitCode?: number;
// This property is not available in managed environment

C++
C++ This property is not available in native environment

\section*{Description}

Returns the process exit code (the meaning of this value is process-specific, however, zero is commonly used to signal successful exit). If the process is still running, this property is undefined.

\section*{snapshots}
```

TypeScript
readonly snapshots: IProcessSnapshot[];
C\#}\mathrm{ // This property is not available in managed environment
C++

```

\section*{Description}

Returns an array of all created process snapshots. This array has at least one element that describes the running process itself and it is always the first element in the array.
createSnapshot

TypeScript
createSnapshot(): IProcessSnapshot;
// This method is not available in managed environment

C++ This method is not available in native environment

\section*{Description}

Create new process snapshot object and return a reference to it

\section*{deleteSnapshot}

TypeScript
deleteSnapshot(snapshot: IProcessSnapshot): void;
// This method is not available in managed environment

C+
/ This method is not available in native environment

\section*{Parameters}
snapshot
The snapshot to delete.

\section*{Description}

Delete the given snapshot. Note that attempt to delete a snapshot object that represents the running process will silently fail.
clearSnapshots

TypeScript
clearSnapshots(): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Description}

Delete all process snapshots

\section*{compareSnapshots}

TypeScript
compareSnapshots(snapshot1: IProcessSnapshot,
snapshot2: IProcessSnapshot
arrange \(=\) SplitMode.Vertical): void;

C\#
// This method is not available in managed environment

C++
/ This method is not available in native environment

\section*{Parameters}
snapshot1
A reference to the first (old) snapshot.
snapshot2
A reference to the second (new) snapshot

\section*{arrange}

Optional window arrangement mode. Defaults to SplitMode.Vertical if omitted.

\section*{Description}

Open the entire address space of given snapshots as two editor windows and use the File Comparison feature to compare their contents. Both specified snapshots must represent actual process snapshots, not the running process. They should also reference the same object.

This object represents a process snapshot. Snapshot object references either a running process (use the 0-th element of an array returned by IProcessWindow.snapshots property to get that) or represents a created process snapshot (using the IProcessWindow.createSnapshot method).

Use the IProcessSnapshot.isSnapshot property to determine if this object represents a running process (false) or an actual snapshot (true). For a an object representing an actual snapshot, IProcessSnapshot.date returns a JavaScript Date object that tells the snapshot creation date and time.

Call the IProcessSnapshot.openRegion method to open a document in the editor to start viewing and editing process memory.
IProcessSnapshot.delete method may be used to delete the snapshot object.
Use the IProcessSnapshot.saveDump method to save a dump of a running process or process snapshot to a file and the IProcessSnapshot.debug method to launch an external debugger to debug a running process or process snapshot.

\section*{IProcessSnapshot Interface}

\section*{Description}

This interface is implemented by a process snapshot object. Use the IProcessWindow.createSnapshot method to create a snapshot and get a reference to this interface.

Declaration

TypeScript
\(\frac{\text { TypeScript }}{\text { interface IProcessSnapshot }\{ }\)
// Properties
readonly isSnapshot: boolean;
readonly date?: Date;
// Methods
openRegion(start?: number, size?: number, readOnly?: boolean): IDocument;
delete(): void;
saveDump (path: string, includeFullMemory = false): Promise<void>;
\} debug(debuggerType: DebuggerType, includeFullMemory = false): void
\}
C\#
// This interface is not available in managed environment

C++
\(\frac{\text { C++ }}{\text { // This interface }}\) is not available in native environment

ProcessSnapshot Properties
isSnapshot

TypeScript
readonly isSnapshot: boolean;

C\#
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

This property is true if the object represents a process snapshot and false if it represents a running process.
date

TypeScript
readonly date?: Date;

C\# This property is not available in managed environment
\(\xrightarrow[\text { C+ }]{1 /}\)
// This property is not available in native environment

\section*{Description}

The date and time of the snapshot creation. This property returns undefined if the object represents a running process
```

ProcessSnapshot Methods

```
openRegion
TypeScript \(\quad\) openRegion(start?: number, size?: number, readOnly?: boolean): IDocument;
C\# This method is not available in managed environment
C++ This method is not available in native environment

\section*{Parameters}
start
A starting offset in the process address space you want to open.
size
A size of the memory window you want to open.
readonly
true to open a read-only view on the process memory and false to enable write access. This parameter is not applicable if opening process snapshots (they are always opened as read-only).

\section*{Description}

Opens the specified process memory of a current snapshot or running process in the editor. If both start and size are omitted, then the whole process space is opened.
delete

TypeScript
delete(): void;

C\#
// This method is not available in managed environment
\(+\)
// This method is not available in native environment

\section*{Description}

Delete the current snapshot.
saveDump

TypeScript
saveDump(path: string, includeFullMemory = false): Promise<void>;

C\#
// This method is not available in managed environment

C+
// This method is not available in native environment

\section*{Parameters}
path
A full path to the dump file.

\section*{includeFullMemory}
true to include all process memory in the dump. false only includes private process memory. Setting this parameter to true results in much larger dump size and longer execution

\section*{Description}

Save a dump of the current process snapshot (or running process) to a given file.
debug

\section*{TypeScript}
debug(debuggerType: DebuggerType, includeFullMemory = false): void;

C\# This method is not available in managed environment

C++ \(/\) This method is not available in native environment

\section*{Parameters}

\section*{debuggertype}

The type of the debugger to use.

\section*{includeFullMemory}
true to include all process memory in the dump. false only includes private process memory. Setting this parameter to true results in much larger dump size and longer execution

\section*{Description}

Launch an external debugger and attach it to a given process snapshot or a running process.

IDocument

\section*{Support Objects}

IWindow Interface

\section*{Description}

IWindow interface is a base interface for all window objects. Currently, the document view object implements the IDocumentView interface, which inherits from IWindow

\section*{Declaration}
```

TypeScript
interface IWindow {
// Properties
readonly name: string;
// Methods
close(): void
}
C\#
// This interface is not available in managed environment
C++

```

\section*{IWindow Properties}

\section*{name}

\section*{TypeScript \\ readonly name: string;}

C\#

C++ This property is not available in native environment

\section*{Description}

Returns the name of the window.

\section*{IWindow Methods}

\section*{close}

TypeScript
close(): void;

C\# This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method }}\) is not available in native environment

\section*{Description}

Closes the window. If this is the last window of a document, this action may lead to a closing of a document.

\section*{Cursor Interface}

\section*{Description}

ICursor interface is implemented by a cursor object. Each document view has a single read-only instance of a cursor object, which can be used to manage editor window's multiple cursors.

Declaration

TypeScript
interface ICursor
// Properties
readonly offset: number;
readonly singleCursor: boolean;
readonly all: number[];
// Methods
move(action: CursorAction, flags = ActionFlags.None): void;
moveTo(offset: number, flags = ActionFlags.None): void;
\}
// This interface is not available in managed environment

C++
// This interface is not available in native environment

\section*{Cursor Properties}

\section*{TypeScript}
readonly offset: number;

C\#
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Returns the main cursor offset.

\section*{singleCursor}

TypeScript
readonly singleCursor: boolean;

C\#
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Returns true if the cursor object has only a single cursor and false otherwise
all

TypeScript
readonly all: number[];
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Returns all cursors in an array.

\section*{ICursor Methods}
move

TypeScript
move(action: CursorAction, flags = ActionFlags.None): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
action
A type of cursor movement.
flags
Optional key modifiers.

\section*{Description}

Emulates the pressing of a specified keyboard key. Optional flags may add a key modifier to the action.

\section*{Example}

TypeScript
// Move the cursor two cells down
activeView.cursor.move(CursorAction.Down);
activeView.Cursor.move(CursorAction.Down)
/ Select the next row of data
activeView.cursor.move(CursorAction.Down, ActionFlags.Shift);

\section*{moveTo}

\section*{TypeScript}
moveTo(offset: number, flags = ActionFlags.None): void;

C\# \(/\) This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method }}\) is not available in native environment

\section*{Parameters}

\section*{offset}

An offset within a document where you want the cursor to move to. May be beyond the current size of a document.
flags
Optional key modifiers.

\section*{Description}

Move the cursor to a given offset. If ActionFlags.Alt modifier is set, adds new alternate cursor. If ActionFlags.Shift modifier and optionally Actionflags.Ctrl are set modifies the current selection.

Example
Typascript
// Move the cursor to \(0 \times 100\), deleting all alternate cursors
activeView.cursor.moveTo(0x100);
ctiveView. Cursor moveTo( \(\theta \times 200\), ActionFlags.Alt)

\section*{Selection Interface}

\section*{Description}

This interface is implemented by the selection object. Each document view has a read-only selection object, which you can query using the IDocumentView.selection property Use the properties and methods of this interface to control the editor window's multiple selection.

Use the special wholeFile global variable to specify the "whole file" selection. This virtual selection object may be used with any document.

Declaration
```

TypeScript
interface ISelection
// Properties
// Properties
readonly endOfffset: number;
readonly singleRange: boolean;
readonly rangeCount: number;
readonly size: number;
// Methods
selectRange(startOffset: number, size: number, mode = SelectRangeMode.Add): void
invert(size: number): void;
invert(size:
}

```
C\#
// This interface is not available in managed environment
\(\xrightarrow[\text { C++ }]{\text { // This interface }}\) is not available in native environment

\section*{Selection Properties}

\section*{startoffset}

\section*{TypeScript \\ readonly startoffset: number;}

C\# This property is not available in managed environment
\(\frac{\text { c++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

The first offset in a selection. Throws an exception if selection is empty.

\section*{endoffset}
\[
\begin{aligned}
& \text { TypeScript } \\
& \text { readonly endOffset: number; } \\
& \text { C\# } \\
& \text { // This property is not available in managed environment } \\
& \text { C++ } \\
& \hline / \text { This property is not available in native environment }
\end{aligned}
\]

\section*{Description}

The last offset in a selection. Throws an exception if selection is empty

\begin{abstract}
singleRange
\(\frac{\text { TypeScript }}{\text { readonly singleRange: boolean; }}\)

C\#
// This property is not available in managed environment
\(\stackrel{C+}{1 /+}\)
// This property is not available in native environment
\end{abstract}

\section*{Description}
true if selection has single range, false otherwise

\section*{rangeCount}

TypeScript
readonly rangeCount: number;

C\# This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Returns the number of ranges in a selection
size

TypeScript
\(\frac{\text { Typescript }}{\text { readonly size: number; }}\)

C\# This property is not available in managed environment
\(\frac{\text { C++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

Total size of a selection (the sum of sizes of all ranges).
empty

TypeScript
Typescript readonly empty: boolean;

C\# This property is not available in managed environment

C++ \(/\) This property is not available in native environment

\section*{Description}
true if selection is empty, false otherwise

\section*{ISelection Methods}

\section*{selectRange}

TypeScript
selectRange(startOffset: number, size: number, mode \(=\) SelectRangeMode.Add): void;

C\#
// This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method is not available in native environment }}\)

\section*{Parameters}

\section*{startoffset}

Starting offset.
size

\section*{Number of bytes to select}
mode
Optional selection mode

\section*{Description}

Performs a selection operation. The result of the operation depends on the value of a mode parameter:
\begin{tabular}{ll} 
Value & Result \\
\hline Replace & Replace a current selection with a given range. \\
\hline Add & Adds the new range to a current selection. \\
\hline Subtract & Subtract the range from a current selection. \\
\hline Overlap & Overlap a given range with a current selection. \\
\hline
\end{tabular}
invert

\section*{TypeScript \\ invert(size: number): void;}

C\#
// This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}
size
The total size of a selection to use for the inversion operation.

\section*{Description}

Invert the current selection. A required size parameter is used to perform this operation.
clear

TypeScript
clear(): void;

C\# // This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Description}

Clear the current selection.

\section*{IPattern Interface}

\section*{Description}

This interface is implemented by a pattern object. Use the global createPattern method to create a pattern object.

\section*{Declaration}

TypeScript
interface IPattern
// Properties
readonly empty: boolean;
readonly size: number;
readonly data: Uint8Array;
\}

C\#
// This interface is not available in native environment

\section*{Pattern Properties}
empty

\section*{TypeScript}
readonly empty: boolean;

C\#
// This property is not available in managed environment
\(\frac{\text { C++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

Returns true if pattern is empty and false otherwise
size

TypeScrip
readonly size: number;
// This property is not available in managed environment

C++
/ This property is not available in native environment

\section*{Description}

Returns the total size of a pattern
data

TypeScript
readonly data: Uint8Array;
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Return a pattern's data array

\section*{AlertButtons Enumeration}

\section*{Description}

Specifies which buttons to place on a message box displayed by a call to alert method.
\begin{tabular}{|c|c|c|}
\hline Symbol & Value & Description \\
\hline Ok & 0 & Display a single button OK. \\
\hline OkCancel & 0x001 & Display OK and Cancel buttons. \\
\hline AbortRetryIgnore & 0x002 & Display Abort, Retry and Ignore buttons. \\
\hline YesNoCancel & 0x003 & Display Yes, No and Cancel buttons. \\
\hline YesNo & 0x004 & Display Yes and Nobuttons. \\
\hline RetryCancel & 0x005 & Display Retry and Cancel buttons. \\
\hline CancelTryContinue & 0x006 & Display Cancel, Try Again and Continue buttons. \\
\hline DefButton1 & 0 & Make the first button default. \\
\hline DefButton2 & 0x100 & Make the second button default. \\
\hline DefButton3 & 0x200 & Make the third button default. \\
\hline DefButton4 & \(0 \times 300\) & Make the fourth button default. \\
\hline
\end{tabular}

\section*{Alerticon Enumeration}

\section*{Description}

The icon to place on a message box, displayed with a call to the alert method
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline No & 0 & No icon to display. \\
\hline Error & \(0 \times 10\) & Display "Error" icon. \\
\hline Question & \(0 \times 20\) & Display "Question" icon. \\
\hline Warning & \(0 \times 30\) & Display "Warning" icon. \\
\hline Information & \(0 \times 40\) & Display "Information" icon. \\
\hline
\end{tabular}

\section*{AlertResult Enumeration}

\section*{Description}

The result of the alert method call.
\begin{tabular}{|c|c|c|}
\hline Symbol & Value & Description \\
\hline Ok & 1 & OK button was pressed. \\
\hline Cancel & 2 & Cancel button was pressed. \\
\hline Abort & 3 & Abort button was pressed. \\
\hline Retry & 4 & Retry button was pressed. \\
\hline Ignore & 5 & Ignore button was pressed. \\
\hline Yes & 6 & Yes button was pressed. \\
\hline No & 7 & No button was pressed. \\
\hline Close & 8 & Close button was pressed. \\
\hline TryAgain & 10 & Try Again button was pressed. \\
\hline Continue & 11 & Continue button was pressed. \\
\hline
\end{tabular}

\section*{CloseMode Enumeration}

\section*{Description}

Specifies the document closing mode.
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline Prompt & 0 & Prompt the user to save changes. \\
\hline Save & 1 & Save document changes. \\
\hline Discard & 2 & Discard document changes. \\
\hline
\end{tabular}

\section*{FindInFilesOperation Enumeration}

\section*{Description}

Type of the Find in Files operation to perform.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline FindFirstMatch & 0 & Only find the first match. \\
\hline FindAllMatches & 1 & Find all matches. \\
\hline
\end{tabular}

\section*{ReplaceInFilesOperation Enumeration}

\section*{Description}

Type of the Replace in Files operation to perform.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline ReplaceAndSave & 0 & Replace all occurrences and save modified files. \\
\hline ReplaceAndOpen & 1 & Replace all occurrences and open modified files in the editor. \\
\hline
\end{tabular}

\section*{FindInFilesFlags Enumeration}

\section*{Description}

Flags that modify the behavior of findInFilesAsync and replacelnFilesAsync operations. Can be a combination of the following values:
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline None & 0 & No special behavior. \\
\hline IncludeSubFolders & \(0 \times 1\) & Include sub-folders of given folders when searching. \\
\hline IgnoreCase & \(0 \times 2\) & Ignore case when matching a pattern or regular expression. \\
\hline DisplayErrors & \(0 \times 4\) & Display file opening errors. \\
\hline IncludeStreams & \(0 \times 8\) & Include alternate NTFS data streams when searching. \\
\hline RegularExpressionUnicode & \(0 \times 10\) & UNICODE (UTF-16) encoding for a regular expression match pattern. \\
\hline
\end{tabular}

\section*{FindInFilesResult Object}

\section*{Description}

This object holds the information produced by findInFilesAsync and replaceInFilesAsync methods.
```

Declaration
TypeScript
interface FindInFilesResult {
// Properties
readonly filesSearched: number;
readonly folderssearched: number;
sed: number;
readonly filesMatched: number;
}
C\#
// This interface is not available in managed environment
C++

```
FindInFilesResult Properties
```

filesSearched
TypeScript
C\#
// This property is not available in managed environment
$\frac{\text { C++ }}{\text { // This property }}$ is not available in native environment

```

\section*{Description}

Total number of files searched.

\section*{foldersSearched}
ypeScrip
readonly foldersSearched: number;

C\#
// This property is not available in managed environment
\(\frac{\text { C++ }}{\text { // This property is not available in native environment }}\)

\section*{Description}

Total number of folders searched

\section*{bytesProcessed}

TypeScript
readonly bytesProcessed: number;

C\#
// This property is not available in managed environment

C++
// This property is not available in native environment

\section*{Description}

Total bytes processed.
filesMatched

TypeScript
readonly filesMatched: number;

C\#
// This property is not available in managed environment

C++

\section*{Description}

Total files matched.

\section*{FindResult Object}

\section*{Description}

This object holds the information of a next pattern match. It BraetmanetMieenthiendAsync, IDocumentView.findRegExpAsync and IDocumentView.findNextAsync methods.

\section*{Declaration}
```

TypeScript
interface FindResult \{
// Properties
readonly offset: number;
readonly size: number;

```
\}

C\# This interface is not available in managed environment
// This interface is not available in native environment
```

offset
TypeScript
readonly offset: number;
C\#
// This property is not available in managed environment
C+t This property is not available in native environment

```

\section*{Description}

The starting offset of a match.
size

TypeScrip
readonly size: number;

C\#
// This property is not available in managed environment
\(\frac{\text { C++ }}{\text { // This property }}\) is not available in native environment

\section*{Description}

Match size.

\section*{ActionFlags Enumeration}

\section*{Description}

Specifies the key modifiers. May be a combination of the following values:
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline None & 0 & No key modifiers. \\
\hline Shift & \(0 \times 1\) & Shift key modifier. \\
\hline Alt & \(0 \times 2\) & Alt key modifier. \\
\hline Ctrl & \(0 \times 4\) & Ctrl key modifier. \\
\hline
\end{tabular}

\section*{CursorAction Enumeration}

\section*{Description}

Emulate a given keyboard button.
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline Up & 0 & Emulate the Up keyboard key. \\
\hline Down & 1 & Emulate the Down keyboard key. \\
\hline Left & 2 & Emulate the Left keyboard key. \\
\hline Right & 3 & Emulate the Right keyboard key \\
\hline Pgup & 4 & Emulate the PgUp keyboard key. \\
\hline PgDn & 5 & Emulate the PgDn keyboard key. \\
\hline Home & 6 & Emulate the Home keyboard key. \\
\hline End & 7 & Emulate the End keyboard key. \\
\hline
\end{tabular}

\section*{GotoMode Enumeration}

\section*{Description}

Specify the mode of operation for IDocumentView.goto method.
Symbol Value Description
Absolute \(0 \quad\) Specified value is an absolute offset.
Relative 1 Specified value is a relative offset. The resulting value is computed as a sum of the default cursor position and goto offset.
Percent 2 Specified value is percentage of the total document size.

\section*{SelectionUpdateMode Enumeration}

\section*{Description}

Sets the selection update mode for the IDocumentView.goto method
Symbol Value Description
\begin{tabular}{lll}
\hline NoChange & 0 & Do not modify selection. \\
\hline Replace & 1 & Replace the current selection with new range from the current cursor position to target offset. \\
\hline Add & 2 & Add the new range (from current cursor position to target offset) to the current selection. \\
\hline Remove & 3 & Remove the new range (from current cursor position to target offset) from the current selection. \\
\hline Invert & 4 & Invert the range (from current cursor position to target offset) in the current selection.
\end{tabular}

\section*{OpeningMode Enumeration}

\section*{Description}

Specify the file opening mode. See the openFile method for more information.
Symbol Value Description
Normal 0 Normal file opening mode. Locks opened file for read-only access by other applications.
Shared 1 Open file in shared mode. Allows other applications to continue making changes to the document. See shared files for more information

\section*{OpenVolumeMode Enumeration}

\section*{Description}

Specify the volume opening mode. See the openVolume method for more information.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Normal & 0 & Open the volume for read-write access. \\
\hline ReadOnly & 1 & Open the volume for read-only access. \\
\hline Snapshot & 2 & Create a volume snapshot and open it. A snapshot is discarded as soon as the document is closed \\
\hline
\end{tabular}
Encodings Enumeration
Default Encodings
\begin{tabular}{lll} 
\\
& Symbol & Value \\
\hline Defaultansi & 0 & Description \\
\hline DefaultoEM & 1 & Default ANSI (depends on the system code page). \\
\hline UTF16 & -5 & Default OEM. \\
\hline UTF8 & 65001 & UNICODE (UTF-16). \\
\hline
\end{tabular}

ANSI Encodings
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline ANSI_Arabic & 1256 & Arabic ANSI. \\
\hline ANSI_Baltic & 1257 & Baltic ANSI. \\
\hline ANSI_Central_European & 1250 & Central European ANSI. \\
\hline ANSI_Cyrillic & 1251 & Cyrillic ANSI. \\
\hline ANSI_Greek & 1253 & Greek ANSI. \\
\hline ANSI_Hebrew & 1255 & Hebrew ANSI. \\
\hline ANSI_Latin_I & 1252 & Latin I ANSI. \\
\hline ANSI_Turkish & 1254 & Turkish ANSI. \\
\hline
\end{tabular}

Arabic Encodings
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Arabic_ASMO449 & 709 & Arabic ASMO449. \\
\hline Arabic_ASMO708 & 708 & Arabic ASMO708. \\
\hline Arabic_Transparent & 710 & Arabic Transparent. \\
\hline Arabic_TransparentASMO & 720 & Arabic Transparent ASMO. \\
\hline
\end{tabular}

ISO Encodings
\begin{tabular}{|c|c|c|}
\hline Symbol & Value & Description \\
\hline ISO_Japanese_JIS & 50222 & Japanese JIS. \\
\hline ISO_Japanese_Katakana_HW & 50221 & Katakana HW ISO. \\
\hline ISO_Japanese_Katakana_NHW & 50220 & Katakana NHW ISO. \\
\hline ISO_Korean & 50225 & Korean ISO. \\
\hline ISO_Simplified_Chinese & 50227 & Simplified Chinese ISO. \\
\hline ISO_Traditional_Chinese & 50229 & Traditional Chinese ISO. \\
\hline ISO_6937_NonSpacing_Accept & 20269 & Non-spacing Accept ISO 6937. \\
\hline ISO_8859_1_Latin_I & 28591 & Latin I ISO 8859-1. \\
\hline ISO_8859_15_Latin_9 & 28605 & Latin 9 ISO 8859-15. \\
\hline ISO_8859_2_Central_Europe & 28592 & Central Europe ISO 8859-2. \\
\hline ISO_8859_3_Latin_3 & 28593 & Latin 3 ISO 8859-3. \\
\hline ISO_8859_4_Baltic & 28594 & Baltic ISO 8859-4. \\
\hline ISO_8859_5_Cyrillic & 28595 & Cyrillic ISO 8859-5. \\
\hline ISO_8859_6_Arabic & 28596 & Arabic ISO 8859-6. \\
\hline ISO_8859_7_Greek & 28597 & Greek ISO 8859-7. \\
\hline ISO_8859_8_Hebrew & 28598 & Hebrew ISO 8859-8. \\
\hline ISO_8859_8_i_Hebrew & 38598 & Hebrew ISO 8859-8i. \\
\hline ISO_8859_9_Latin_5 & 28599 & Latin 5 ISO 8859-9. \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|}
\hline Symbol & Value & Description \\
\hline EBCDIC_Arabic & 20420 & Arabic EBCDIC. \\
\hline EBCDIC_Cyrillic_Russian & 20880 & Cyrillic Russian EBCDIC. \\
\hline EBCDIC_Cyrillic_Serbian_Bulgarian & 21025 & Cyrillic Serbian/Bulgarian EBCDIC. \\
\hline EBCDIC_Denmark_Norway & 20277 & Denmark/Norway EBCDIC. \\
\hline EBCDIC_Denmark_Norway_Euro & 1142 & Denmark/Norway + Euro EBCDIC. \\
\hline EBCDIC_Finland_Sweden & 20278 & Finland/Sweden EBCDIC. \\
\hline EBCDIC_Finland_Sweden_Euro & 1143 & Finland/Sweden + Euro EBCDIC. \\
\hline EBCDIC_France & 20297 & France EBCDIC. \\
\hline EBCDIC_France_Euro & 1147 & France + Euro EBCDIC. \\
\hline EBCDIC_Germany & 20273 & Germany EBCDIC. \\
\hline EBCDIC_Germany_Euro & 1141 & Germany + Euro EBCDIC. \\
\hline EBCDIC_Greek & 20423 & Greek EBCDIC. \\
\hline EBCDIC_Hebrew & 20424 & Hebrew EBCDIC. \\
\hline EBCDIC_Icelandic & 20871 & Icelandic EBCDIC. \\
\hline EBCDIC_Icelandic_Euro & 1149 & Icelandic + Euro EBCDIC. \\
\hline EBCDIC_International & 500 & International EBCDIC. \\
\hline EBCDIC_International_Euro & 1148 & International + Euro EBCDIC. \\
\hline EBCDIC_Italy & 20280 & Italy EBCDIC. \\
\hline EBCDIC_Italy_Euro & 1144 & Italy + Euro EBCDIC. \\
\hline EBCDIC_Japanese_Katakana_Extended & 20290 & Japanese Katakana Extended EBCDIC. \\
\hline EBCDIC_Korean_Extended & 20833 & Korean Extended EBCDIC. \\
\hline EBCDIC_Latin_1_OS & 1047 & Latin 1 OS EBCDIC. \\
\hline EBCDIC_LA_Spain & 20284 & Latin Spain EBCDIC. \\
\hline EBCDIC_LA_Spain_Euro & 1145 & Latin Spain + Euro EBCDIC. \\
\hline EBCDIC_Latin_1_OS_Euro & 20924 & Latin 1 OS + Euro EBCDIC. \\
\hline EBCDIC_Modern_Greek & 875 & Modern Greek EBCDIC. \\
\hline EBCDIC_Multilingual_ROECE_Latin_2 & 870 & Multilingual ROECE Latin 2 EBCDIC. \\
\hline EBCDIC_Thai & 20838 & Thai EBCDIC. \\
\hline EBCDIC_Turkish & 20905 & Turkish EBCDIC. \\
\hline EBCDIC_Turkish_Latin_5 & 1026 & Turkish Latin 5 EBCDIC. \\
\hline EBCDIC_US_Canada & 37 & US/Canada EBCDIC. \\
\hline EBCDIC_US_Canada_Euro & 1140 & US/Canada + Euro EBCDIC. \\
\hline EBCDIC_UK & 20285 & UK EBCDIC. \\
\hline EBCDIC_UK_Euro & 1146 & UK + Euro EBCDIC. \\
\hline
\end{tabular}

\section*{SCII Encodings}
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline ISCII_Assamese & 57006 & Assamese ISCII. \\
\hline ISCII_Bengali & 57003 & Bengali ISCII. \\
\hline ISCII_Devanagari & 57002 & Devangari ISCII. \\
\hline ISCII_Gujarati & 57010 & Gujarati ISCII. \\
\hline ISCII_Kannada & 57008 & Kannada ISCII. \\
\hline ISCII_Malayalam & 57009 & Malayalam ISCII. \\
\hline ISCII_Oriya & 57007 & Oriya ISCII. \\
\hline ISCII_Punjabi & 57011 & Punjabi ISCII. \\
\hline ISCII_Tamil & 57004 & Tamil ISCII. \\
\hline ISCII_Telugu & 57005 & Telugu ISCII. \\
\hline
\end{tabular}
\begin{tabular}{|l|l|}
\hline Symbol & Value \\
\hline MAC_Arabic & 10004 \\
\hline Description \\
\hline MAC_Croatia & 10082 \\
\hline MAC_Cyrillic & 10007 \\
\hline MAC_Greek_I & 10006 \\
\hline Croatia MAC. \\
\hline MAC_Hebrew & 10005 \\
\hline MAC_Icelandic & 10079 \\
\hline MAC_Japanese & 10001 \\
\hline MAC_Korean & 10003 \\
\hline MAC_Latin_II MAC. & Hebrew I MAC. \\
\hline MAC_Roman & 10029 \\
\hline MAC_Romania & 10000 \\
\hline MAC_Simplified_Chinese & Japanese MAC. \\
\hline MAC_Thai & 10010
\end{tabular}

\section*{OEM Encodings}
\begin{tabular}{|l|l|}
\hline Symbol & Value \\
\hline OEM_Arabic & 864 \\
\hline Description \\
\hline OEM_Baltic & 775 \\
\hline Arabic OEM. \\
\hline OEM_Canadian_French & 863 \\
Baltic OEM. \\
\hline OEM_Greek & 855 \\
\hline Candian French OEM. \\
\hline OEM_Hebrew & 737 \\
\hline Cyrillic OEM. \\
\hline OEM_Latin_II & 862 \\
\hline OEM_Modern_Greek & 861 \\
\hline Hebrew OEM. \\
\hline OEM_Multilingual_Latin_I & 852 \\
\hline Icelandic OEM. \\
\hline OEM_Nordic & 869 \\
\hline Latin II OEM. \\
\hline OEM_Portuguese & 850 \\
\hline Modern Greek OEM. \\
\hline OEM_Russian & 858 \\
\hline Multilingual Latin I OEM. \\
\hline OEM_Turkish & 865 \\
\hline Multilingual Latin I + Euro OEM. \\
\hline & 860 \\
Nordic OEM. \\
\hline
\end{tabular}

\section*{Asian Encodings}
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline Japanese_Katakana_Extended & 50930 & Japanese Katakana Extended \\
\hline Japanese_Latin_Extended & 50939 & Japanese Latin Extended \\
\hline JIS_0208_0121 & 20932 & JIS 0208-121 \\
\hline Korean_Johab & 1361 & Korean Johab \\
\hline Korean_Extended & 50933 & Korean Extended \\
\hline Simplified_Chinese & 50936 & Simplified Chinese \\
\hline Simplified_Chinese_GB2312 & 20936 & Simplified Chinese GB2312 \\
\hline Simplified_Chinese_Extended & 50935 & Simplified Chinese Extended \\
\hline
\end{tabular}
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline KOI8_R & 20866 & KOI8-R \\
\hline T61 & 20261 & T61 \\
\hline TCA_Taiwan & 20001 & TCA Taiwan \\
\hline TeleText_Taiwan & 20004 & TeleText Taiwan \\
\hline Ukrainian_KOI8_U & 21866 & Ukrainian KOI8-U \\
\hline US_Canada_Japanese & 50931 & US/Canada Japanese \\
\hline US_Canada_Traditional_Chinese & 50937 & US/Canada Traditional Chinese \\
\hline US_ASCII_7bit & 20127 & US ASCII 7-bit \\
\hline Wang_Taiwan & 20005 & Wang Taiwan \\
\hline CNS_Taiwan & 20000 & CNS Taiwan \\
\hline Eten_Taiwan & 20002 & Eten Taiwan \\
\hline EUC_Japanese & 51932 & EUC Japanese \\
\hline EUC_Korean & 51949 & EUC Korean \\
\hline EUC_Simplified_Chinese & 51936 & EUC Simplified Chinese \\
\hline EUC_Traditional_Chinese & 51950 & EUC Traditional Chinese \\
\hline Europa_3 & 29001 & Europa 3 \\
\hline HZ_GB2312_Simplified_Chinese & 52936 & HZ GB2312 Simplified Chinese \\
\hline IA5_German_7bit & 20106 & IA5 German 7-bit \\
\hline IA5_IRV_International_Alphabet_No5_7bit & 20105 & IA5 IRV International Alphabet No5 7-bit \\
\hline IA5_Norwegian_7bit & 20108 & IA5 Norwegian 7-bit \\
\hline IA5_Swedish_7bit & 20107 & IA5 Swedish 7-bit \\
\hline IBM5550_Taiwan & 20003 & IBM5550 Taiwan \\
\hline
\end{tabular}

\section*{FindFlags Enumeration}

\section*{Description}

One or more of the following values may be combined to change the behavior of the following methods:
\begin{tabular}{ll} 
Find Methods & Replace Methods \\
\hline IDocumentView.findAsync & IDocumentView.replaceAsync \\
\hline IDocumentView.findAllAsync & IDocumentView.replaceAllAsync \\
\hline IDocumentView.findRegExpAsync & IDocumentView.replaceRegExpAsync \\
\hline IDocumentView.findAllRegExpAsync & IDocumentView.replaceAllRegExpAsync \\
\hline
\end{tabular}

Not all flags are applicable for all methods.
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline Forward & 0 & Search forward. \\
\hline Backwards & 1 & Search backwards. \\
\hline IgnoreCase & 2 & Ignore case when matching pattern. \\
\hline RegularExpression & 4 & Use regular expressions mode. \\
\hline ReplaceRegularExpression & 8 & The replace pattern is a regular expressions replace pattern. \\
\hline ResetReplace & 16 & Reset the replace mode. \\
\hline
\end{tabular}

\section*{ReplaceCommand Enumeration}

\section*{Description}

Specifies the action to perform with IDocumentView.replaceAsync and IDocumentView.replaceRegExpAsync methods.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Find & 0 & Find the next match. \\
Replace & 1 & Replace the current match. \\
\hline
\end{tabular}

\section*{FillFlags Enumeration}

\section*{Description}

Specifies the behavior of the IDocumentView.fillAsync method.
\begin{tabular}{lll} 
Symbol & \multicolumn{2}{l}{ Value } \\
\hline Noscription \\
\hline Transparent & 0 & No special behavior. \\
\hline Reverse & \(0 \times 2\) & \begin{tabular}{l} 
Use "transparent" fill mode. In this mode, pattern will not start from the beginning when filling next range in a multiple \\
selection.
\end{tabular} \\
\hline
\end{tabular}

\section*{ValueType Enumeration}

\section*{Description}

Specifies the type of the value. Used by the createPattern method.
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline Byte & 0 & Numbers should be interpreted as bytes (8-bit unsigned values). \\
\hline Word & 1 & Numbers should be interpreted as words (16-bit unsigned values). \\
\hline DoubleWord & 2 & Numbers should be interpreted as double words (32-bit unsigned values). \\
\hline QuadWord & 3 & Numbers should be interpreted as quad words (64-bit unsigned values). \\
\hline Float & 4 & Numbers should be interpreted as single-precision floating-point values. \\
\hline Double & 5 & Numbers should be interpreted as double-precision floating-point values. \\
\hline
\end{tabular}

\section*{HexFormat Enumeration}
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Intel & 0 & Intel Hex encoded file format. \\
Motorola & 1 & Motorola S-Records encoded file format. \\
\hline Enumeration & & \\
& & \\
&
\end{tabular}

Specifies the behavior of the IDocumentView.insertHexAsync and openHexAsync methods.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline None & 0 & No special behavior. \\
\hline InsertMode & 1 & Use insert mode when inserting the hex file in the document. Is not used for openHexAsync method. \\
\hline IgnoreDataOffset & 2 & Ignore data offsets in a hex file.
\end{tabular}

\section*{DisplayAs Enumeration}

\section*{Description}

Specifies the editor window cell view type. The script may control the current view's view type using the IDocumentView.displayAs property.
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline Hex & 0 & Display as hexadecimal numbers. \\
\hline Decimal & 1 & Display as decimal numbers. \\
\hline Octal & 2 & Display as octal numbers. \\
\hline Binary & 3 & Display as binary numbers. \\
\hline Float & 4 & Display as single precision floating-point numbers (IEEE 754). \\
\hline Double & 5 & Display as double precision floating-point numbers (IEEE 754). \\
\hline
\end{tabular}

\section*{AddressDisplayAs Enumeration}

\section*{Description}

Specifies the offset representation in an editor window. IDocumentView.addressDisplayAs property control this mode for a given document view.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Hex & 0 & Hexadecimal offsets. \\
\hline Decimal & 1 & Decimal offset. \\
\hline Octal & 2 & Octal offsets. \\
\hline
\end{tabular}

\section*{GroupBy Enumeration}
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Bytes & 0 & Group values by bytes (8-bit unsigned integers). \\
\hline Words & 1 & Group values by words (16-bit unsigned integers). \\
\hline DoubleWords & 2 & Group values by double words (32-bit unsigned integers). \\
\hline QuadWords & 3 & Group values by quad words (64-bit unsigned integers). \\
\hline
\end{tabular}

\section*{SelectionLoadType Enumeration}

\section*{Description}

Selection loading mode. See IDocumentView.loadSelectionAsync method for more information.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline New & 0 & Replace the current selection with loaded selection. \\
\hline Add & 1 & Add the loaded selection to the current selection. \\
\hline Subtract & 2 & Subtract the loaded selection from the current selection.
\end{tabular}

\section*{SelectRangeMode Enumeration}

\section*{Description}

Specifies the selection modification mode. See the ISelection.selectRange method.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Replace & 0 & Replace the current selection. \\
\hline Add & 1 & Add to the current selection. \\
\hline Subtract & 2 & Subtract from the current selection. \\
\hline Overlap & 3 & Overlap with the current selection.
\end{tabular}

\section*{BitwiseOperation Enumeration}

\section*{Description}

The type of the bitwise operation to perform. See the IDocumentView.bitwiseOpAsync for more information.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Not & 0 & Perform a bitwise NOT operation: \(x=\sim x\). \\
\hline or & 1 & Perform a bitwise OR operation: \(x=x \mid v\). \\
\hline And & 2 & Perform a bitwise AND operation: \(x=x \& v\). \\
\hline Xor & 3 & Perform a bitwise XOR operation: \(x=x^{\wedge} v\). \\
\hline
\end{tabular}

\section*{ArithmeticOperation Enumeration}

\section*{Description}

The type of the arithmetic operation to perform. See the IDocumentView.arithmeticOpAsync for more information.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Negation & 0 & Perform an arithmetic negation operation: \(x=-x\). \\
\hline Addition & 1 & Perform an arithmetic addition operation: \(x=x+v\). \\
\hline Subtraction & 2 & Perform an arithmetic subtraction operation: \(x=x-v\). \\
\hline Multiplication & 3 & Perform an arithmetic multiplication operation: \(x=x * v\). \\
\hline Division & 4 & Perform an arithmetic division operation: \(x=x / v\). \\
\hline Remainder & 5 & Perform an arithmetic remainder operation: \(x=x \% v\). \\
\hline SetMinimum & 6 & Perform an arithmetic minimum operation: \(x=\min (x, v)\). \\
\hline SetMaximum & 7 & Perform an arithmetic maximum operation: \(x=\max (x, v)\). \\
\hline
\end{tabular}

\section*{ShiftOperation Enumeration}

\section*{Description}

The type of the shift operation to perform. See the IDocumentView.shiftOpAsync for more information.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline LogicalLeft & 0 & Perform a logical left shift operation: \(x=x \ll v\). \\
\hline LogicalRight & 1 & Perform a logical right shift operation: \(x=x \gg v\). \\
\hline ArithmeticLeft & 2 & Perform an arithmetic left shift operation: \(x=x \ll v\). \\
\hline ArithmeticRight & 3 & Perform an arithmetic right shift operation: \(x=x \gg v\). \\
\hline RotateLeft & 4 & Perform a left rotation operation: \(x=\operatorname{rotl}(x, v)\). \\
\hline RotateRight & 5 & Perform a right rotation operation: \(x=\operatorname{rotr}(x, v)\). \\
\hline
\end{tabular}

\section*{CaseOperation Enumeration}

\section*{Description}

The type of the case operation to perform. See the IDocumentView.caseOpAsync for more information.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Lower & 0 & Make lowercase. \\
\hline Upper & 1 & Make uppercase. \\
\hline Toggle & 2 & Toggle case. \\
\hline
\end{tabular}

\section*{ReverseOperation Enumeration}

\section*{Description}

The type of the reverse operation to perform. See the IDocumentView.reverseOpAsync for more information.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Bits & 0 & Reverse bits. \\
Bytes & 1 & Reverse bytes.
\end{tabular}

\section*{OperationRange Enumeration}

\section*{Description}

Specifies the type of operation to perform
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline CurrentCell & 0 & Run the data operation on the current cell only. \\
\hline Selection & 1 & Run the data operation on the current selection. \\
\hline WholeFile & 2 & Run the data operation on the whole document. \\
\hline
\end{tabular}

\section*{OperandSize Enumeration}

\section*{Description}

Specifies the operand size for the data operation.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Byte & 0 & Byte-size operands (8-bit unsigned integers). \\
\hline Word & 1 & Word-size operands (16-bit unsigned integers). \\
\hline DoubleWord & 2 & Double word-size operands (32-bit unsigned integers). \\
\hline QuadWord & 3 & Quad word-size operands (64-bit unsigned integers). \\
\hline
\end{tabular}

\section*{SplitMode Enumeration}

\section*{Description}

Window arrangement mode.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline Horizontal & 0 & Arrange windows horizontally. \\
\hline Vertical & 1 & Arrange windows vertically. \\
\hline
\end{tabular}

\section*{DebuggerType Enumeration}

\section*{Description}

Specifies the type of an external debugger to launch.
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline RegisteredJit & 0 & Launch a registered JIT debugger. \\
\hline VisualStudio & 1 & Launch Visual Studio debugger. \\
\hline WinDbgX & 2 & Launch WinDBG Preview or WinDBG debugger. \\
\hline
\end{tabular}

\section*{Scripting}

Hex Editor Neo provides a programming interface to its basic file editing features. The interface provided by the Hex Editor Neo may be used by any code written in any automation-compatible language.

Three objects, File Document Object, Multiple Selection Object and Sequence Object are provided by the application. User code can create these objects, call methods they expose and query/set their properties. As a result, it may achieve the same functionality as provided by the Hex Editor Neo's user interface, with same performance characteristics. Hex Editor Neo's window does not appear on the screen when any of exposed objects is instantiated.

\section*{File Document Object}

File Document object represents a file opened for editing. It corresponds to a document in the Hex Editor Neo's user interface. A lot of editing actions may be performed on this object.

Such concepts as operation history, patch creation, Undo and Redo are fully accessible for user code via this object.
Creating File Document Object section describes how you create a File Document object and Working with File Document Object section illustrates the ways this created object may be used. The IFileDocument Interface section in Scripting Reference section contains in-depth documentation for all properties and methods exposed by the File Document object.

\section*{Multiple Selection Object}

Multiple Selection object represents a multiple selection in a document. This object is used by a number of File Document object's methods to define the portion of the document on which this method operates.

Creating Multiple Selection Object section describes how you create a Multiple Selection object and Working with Multiple Selection Object section illustrates the ways this created object may be used. The IMultiSelection Interface section in Scripting Reference section contains in-depth documentation for all properties and methods exposed by the Multiple Selection object.

\section*{Sequence Object}

Sequence object serves as a thin layer that helps scripting languages that lack support for pointers and arrays to construct and pass a pattern. For example, it is impossible to create pattern for a IFileDocument.Find method in JavaScript. In this case, the Sequence object comes to help.

Creating Sequence Object section describes how you create a Sequence object and Working with Sequence Object section illustrates the ways this object may be used.

\section*{Thread Safety}

Any single object taken from the Hex Editor Neo may not be used by multiple threads at the same time. User code should serialize access to objects' methods and properties if it needs to access the object from different threads. In such scenario, user code may call object's methods and access object's properties from any thread, provided the access is strictly serialized.
At the same time, user code may simultaneously access two different objects from two different threads. There is no conflict in this situation

\section*{Lengthy Operations Limitations}

Unlike the Hex Editor Neo's user interface, File Document and Multiple Selection objects do not provide you with a way to cancel any lengthy operation. When a thread calls a lengthy operation, it only has an option to wait until it's finished. Object performing this operation may not be accessed from another thread until operation has finished.

\section*{Document Object}

\section*{Creating Document Object}

File Document object has a following class ID: FileDocument. FileDocument. Use the method provided by your language to create an ActiveX object. For example, in JavaScript:

\section*{JavaScript}
var fdoc = new ActiveXObject("FileDocument.FileDocument");
// work with fdoc object
Created object is in uninitialized state. You cannot execute any method or access any property until you call the IFileDocument.New or IFileDocument.Open methods to initialize an object.

\section*{Working with Document Object}

This section illustrates several document object usage scenarios. All examples in this section suppose that you have a valid reference to a file document object, as described in Creating Document Object section.

All actions that lead to document modification create new operations in document's operation history

\section*{Initializing Document Object}

Before you start using the newly created document object, you must initialize it.
If you want to associate a document object with any existing file, call the IFileDocument.Open method. If you want to associate a document object with an empty file, call the IFileDocument.New method. After that, the document object enters the initialized state and is ready to process further requests.

To open a volume, call the IFileDocument.OpenVolume method. You may also open a physical disk (IFileDocument.OpenPhysicalDisk) and process memory (IFileDocument.OpenProcess).

User Interface Analogue: Creating New Documents and Opening Documents.

\section*{Closing Document Object}

The IFileDocument.Close method is used to close the document object. The object enters an uninitialized state after it is closed. All changes made to the document are silently discarded.

You don't need to explicitly call this method as it is automatically called when object is destroyed. Although, if you are going to reuse the document object (that it, call Open or New method again), this method must be called.

\section*{Saving Changes}

The IFileDocument.Modified property is TRUE if the document has been modified and FALSE otherwise. It can be used to determine whether the document must be saved. To save a document, call the IFileDocument.Save or IFileDocument.SaveAs methods. The first method overwrites the existing file while the second method creates a copy of the
file.
If the document object was initialized with a call to IFileDocument.New method, IFileDocument.Save method cannot be called until you call IFileDocument.SaveAs method that also updates the IFileDocument.FileName property. IFileDocument.FileName property contains the full path to the file this document is currently representing.

User Interface Analogue: Saving Documents.

\section*{Reading and Writing Data}

The IFileDocument.Read and IFileDocument.Write methods are used to read and write file's data. Calling the IFileDocument.Write method creates new operation in file's operation history

User Interface Analogue: Data Modification.

\section*{Inserting Data}

Several file document object's methods are used to insert data into the file. All insert methods increase the file's size and shift remaining data to free up space for inserted data.

IFileDocument.Insert, IFileDocument.InsertByte and IFileDocument.InsertPattern methods are used to insert data into the document.
User Interface Analogue: Data Modification and Insert command

\section*{Deleting Data}

IFileDocument.Delete and IFileDocument.DeleteMulti methods are used to delete data from the document. After deletion, the size of the file is decreased and remaining data is shifted.

User Interface Analogue: Delete command

\section*{Filling Data}

Unlike Insert group of methods, Fill methods overwrite original file's data with new data.
IFileDocument.Fill, IFileDocument.FillMulti, IFileDocument.FillByte and IFileDocument.FillByteMulti methods are used to fill data.
User Interface Analogue: Fill command.

\section*{Find and Find All}

Call the IFileDocument.Find method to find a pattern in a document. Call the IFileDocument.FindAll method to locate all occurrences of a pattern in a document.
User Interface Analogue: Find and Find All commands.

\section*{Replace and Replace All}

The IFileDocument.Replace method is used to replace a single occurrence of the pattern with another pattern. The powerful IFileDocument.ReplaceAll method automatically locates all occurrences of specified pattern in a file and replace them with another pattern

User Interface Analogue: Replace and Replace All commands.

\section*{Calculating Statistics}

The IFileDocument.GatherStatistics method calculates the General Statistics for the document (or selection) and IFileDocument.PatternStatistics method calculates the Pattern Statistics for it.

User Interface Analogue: Statistics Tool Window.

\section*{Inserting Files}

The IFileDocument.PutFile method is used to put the contents of a disk file to the current document.
User Interface Analogue: Insert File command.

\section*{Controlling File's Size}

The read-write IFileDocument.FileSize property can be used to query or change the file's size. If you use it to change the size, new operation in operation history is created.
User Interface Analogue: Change File Size command

\section*{Undo and Redo}

The IFileDocument.CanUndo and IFileDocument.CanRedo properties are TRUE if IFileDocument.Undo and IFileDocument.Redo methods may be called for a document correspondingly. The IFileDocument.Undo method undoes the previous operation, placing it into the operation tail. IFileDocument.Redo method makes the first operation from operation tail a current operation

User Interface Analogue: Edit » Undo and Edit » Redo commands.

\section*{Working with Operation History}

IFileDocument.Purge, IFileDocument.SaveHistory and IFileDocument.LoadHistory methods are used to purge an operation history, save it to a file or load from a file correspondingly.

User Interface Analogue: History » Clear History and History » Purge... commands.
\#\#Patch Creation
The IFileDocument.CreatePatch method is used to create a patch from the current state of the document.
User Interface Analogue: File » Create Patch... command.
Clipboard Methods

File document object's IFileDocument.Copy, IFileDocument.Cut and IFileDocument.Paste2 methods are used to put file's data to the Clipboard or paste data from the Clipboard into the document.

The IFileDocument.PasteText method is used to paste text from the Clipboard as it is, either in ANSI or UNICODE. You can also use thelFileDocument.PasteText2 method to paste the text from the Clipboard and convert it to a given encoding.

User Interface Analogue: Edit » Copy, Edit » Merge and Copy, Edit » Cut, Edit » Merge and Cut and Edit » Paste commands.

\section*{Bitwise and Other Operations}

Four methods, IFileDocument.BitwiseOp, IFileDocument.ArithmeticOp, IFileDocument.ShiftOp and IFileDocument.CaseOp, are used to perform bitwise, arithmetic, shift and case change operations correspondingly.

\section*{Regular Expressions}

The following methods accept regular expression as their parameters: IFileDocument.FindRegExp2, IFileDocument.FindAllRegExp2, IFileDocument.ReplaceAllRegExp2 and IFileDocument.PatternStatisticsRegExp2.

\section*{Multiple Selection Object}

\section*{Creating Multiple Selection Object}

An empty multiple selection object is created with a call to IFileDocument.CreateEmptySelection method. This means that you must have a valid File Document object in order to create a multiple selection object.

\section*{JavaScript}
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msel = fdoc.CreateEmptySelection()
// work with msel object

Created object is not related to the document object, may live longer that its creator and may be used in calls to methods provided by other document objects.

\section*{Working with Multiple Selection Object}

The created multiple selection object is empty. IMultiSelection.Empty property is TRUE for empty selection objects. You may use the IMultiSelection.AddRange method to add a contiguous range to the selection, IMultiSelection.RemoveRange method to remove (subtract) a range from the selection and IMultiSelection.ToggleRange method to "toggle" the given range in a selection.

IMultiSelection.InsertRange and IMultiSelection.DeleteRange are used to insert and delete new ranges into the selection. These methods differ from previously described in that the data beyond the inserted/deleted range is shifted. IMultiSelection.DeleteRange method is not currently implemented.

Call the IMultiSelection.Invert method to invert the current selection and IMultiSelection.Clear method to clear the current selection. Both methods perform in constant-time, not depending on the selection's complexity

IMultiSelection.Save and IMultiSelection.Load methods may be used to save (with compression) the current selection to a file or load it from a file correspondingly. IMultiSelection.Load method accepts the loading mode, allowing you to merge two selection objects using different algorithms.

IMultiSelection.IsIn method checks if the given offset is located within the selection.
The IMultiSelection.TotalSize property holds the total size of all ranges in the selection. IMultiSelection.Count property holds the total number of ranges in the selection.

\section*{Sequence Object}

\section*{Creating Sequence Object}

An empty sequence object is created with a call to IFileDocument.CreateSequence method. This means that you must have a valid File Document object in order to create a sequence object

\section*{JavaScript}
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var seq= fdoc.CreateSequence();
// work with seq object

Created object is not related to the document object, may live longer that its creator and may be used in calls to methods provided by other document objects.

\section*{Working with Sequence Object}

The created sequence object is empty. ISequence.Empty property is TRUE for empty sequences. You may use the ISequence.AddText and ISequence.AddData methods to add new data to a sequence, ISequence.Remove method to delete a part of the sequence and ISequence.Clear method to clear the sequence.

A copy of the sequence may be constructed with a call to the ISequence.Copy method. The ISequence.CreateSubSequence method, on the other hand, allows you to copy only part of the sequence.

Individual sequence bytes may be accessed and modified through the default ISequence.Value property. Current sequence size is retrieved through the ISequence.Length property. This same property may be used to change the size of the sequence, for example, to prepare the sequence before calling IFileDocument.Read method.

\section*{JavaScript}
// Reading data from the document
var read_seq = fdoc.CreateSequence()
read_seq.Length = 1024;
fdoc.Read(read_seq.Data, 0, read_seq.Length);
// Performing Search \& Replace
var find_seq \(=\) fdoc.CreateSequence(), rep_seq \(=\) fdoc.CreateSequence();
find_seq.AddData(SequenceDataBytes, 0x0d);
rep_seq.AddData(SequenceDataBytes, 0x0d, 0x0a);
var sel = fdoc.CreateEmptySelection();
sel.AddRange( \(\theta\),fdoc.FileSize)
var occurrences \(=\) fdoc.ReplaceAll(find seq.Data, find seq.Length, rep seq.Data, rep seq.Length, false, sel);
alert(occurrences + " replacements made.");

\section*{Parser Object}

Obtaining Parser Object

Parser object is an intrinsic object. It is provided under the name parser to the script called by the Structure Viewer. There is no other way to obtain a reference to this object. This object is a default global object for a running script, so its methods may be called directly, for example C++
alert("Message"); // equivalent to parser.alert("Message");

\section*{Working with Parser Object}

Parser Object exposes a number of methods which can be used by the script function called by Structure Viewer. Consult the IParser interface section for more information.

\section*{Reference}

Document Object

\section*{FileDocument Interface}

\section*{Description}

This interface is implemented by File Document Object. See the Creating File Document Object section to see how you create the object in your code.
```

Declaration
TypeScript
interface IFileDocument extends IDispatch (
// Properties
AutoCreateBranch: boolean;
readonly FileName: string;
ileSize: number;
readonly Modified: boolean
readonly Stream: boolean;
RegExpSyntaxError: string;
readonly CanRedo: boolean;
readonly CanUndo: boolean;
readonly CanPaste: boolean;
// Methods
vew(): void;
New(): void;
Open(Name: string, ReadOnly: boolean): void
OpenPhysicalDisk(PhysicalDiskName: string, FriendlyName: string, ReadOnly: boolean): void;
OpenVolume(VolumeName: string, FriendlyName:, Size: number, ReadOnly: boolean): void;
Save(ForceBackup: boolean): void;
SaveAs(FileName: string): void;
GetModifiedSel(Selection: IMultiSelection): void
Decrypt(Provider: string
ProviderType: numb
Password: string
Selection: IMultiSelection,
KeyLength?: number): void;
Delete(Offset: number, Size: number): void;
DeleteMulti(Selection: IMultiSelection): void;
Encrypt(Provider: string,
ProviderType: number
AlgoID: number,
Password: string
selection: IMultiSelection,
eylength?: number): void;
FillByte(FillValue: number, Offset: number, Size: number): void;
InsertByte(ByteFill: number, Offset: number, Size: number): void;
PutFile(FileName: string, Offset: , InsertMode: boolean): void;
CreatePatch(PatchName: string, Options: Patchoptions, PathToStubFile?: string): void;
Purge(Leve1: HistoryPurgeLeve1): void;
Redo(Steps?: number): void,
Undo(Steps?: number): void;
很(SergeBlocks: boolean): void
Paste(Offset: number InsertMode: boolean): void
Paste2(Offset: number, Type: TextType, InsertMode: boolean): void;
PasteText(Offset: number, Unicode: boolean, InsertMode: boolean): void;
PasteText2(Offset: number, CodePage: number, InsertMode: boolean): void
FindAllRegExp(RegExp: string,
SubExpression: number
Slar: RegExpFlags,
StartFrom: num
Found: IMulti
MultiSelection): void;
indAllRegExp2(RegExp: string,
CodePage: number,
SubExpression: number,
Flags: RegExpFlags,
StartFrom: number
Sound: IMultiSelection): void;
ReplaceAllRegExpWithRegExp(RegExp: string,
ReplacePattern: string,
胙age: number,
lags: RegExpFlags
StartFrom: number,
ArithmeticOp(Selection: IMultiSelection,
Type: ArithmeticOpType,
pSize: OperandSizeType
Operand?: any): void;
CaseOp(Selection: IMultiSelection
ype: CaseOpType
CodePage?: number): void;
ShiftOp(Selection: IMultiSelection, Type: ShiftOpType, OpSize: , Bits: number): void;
ReverseOp(Selection: IMultiSelection, Type: ReverseOpType, OpSize: OperandSizeType): void;
CreateEmptySelection(): IMultiSelection;
CreateSequence(): ISequence,
BitwiseOpS(Selection: IMultiSelection, Type: BitwiseOpType, Sequence: ISequence): void;
FillMultiS(Sequence: ISequence, Selection: IMultiSelection, Continue: boolean): void;
FillS(Sequence: ISequence, Offset: number, Size: number): void
FindAllS(Sequence: ISequence,

```
selection: IMultiSelection,
IgnoreCase: boolean,
Found: IMultiSelection): void
FindS(Sequence: ISequence,
Selection: IMultiSelection,
Searchup: boolean,
IgnoreCase: boolean): void;
InsertPatternS(Sequence: ISequence, Offset: number, Size: number): void;
Inserts(Sequence: ISequence, Offset: number): void;
ReadS(Sequence: ISequence, Offset: number): void;
ReplaceAllRegExpS(RegExp: string,
SubExpression: number,
Flags: RegExpFlags,
StartFrom: number,
Size: number): void;
ReplaceAllRegExp2S(RegExp: string,
CodePage: number,
Flags: RegExpFlags,
Sequence: ISequence,
StartFrom: number,
ReplaceAlls(SequenceFrom: ISequence,
SequenceTo: ISequence
Selection: IMultiSelection): void;
ReplaceS(Offset: number, Size: number, Sequence: ISequence): void;
ToNumber(Value: any): number;
WriteS(Sequence: ISequence, Offset: number): void;
\}
```

C\#
public interface IFileDocument : IDispatch
// Properties
bool AutoCreateBranch { get; set; }
string FileName { get; }
ulong MoleSize { get;
bool Stream { get; }
bool ReadOnly { get; }
string RegExpSyntaxError { get; set; }
bool CanRedo { get; }
bool CanPaste {get;}
// Methods
void Close();
void New();
void Open(string Name, bool ReadOnly);
void OpenPhysicalDisk(string PhysicalDiskName, string FriendlyName, bool ReadOnly)
void OpenProcess(uint ProcessId, long Start, ulong Size, bool ReadOnly)
void OpenVolume(string VolumeName, string FriendlyName, bool ReadOnly);
void SaveAs(string FileName);
void GetModifiedSel(IMultiSelection Selection);
void Decrypt(string Provider,
int ProviderType,
uint AlgoID,
tring Password,
MultiSelection Selection,
uint KeyLength);
void Delete(ulong Offset, ulong Size);
void Delete(ulong (IMultiSelection Selection);
void Encrypt(string Provider,
int ProviderType,
uint AlgoID,
IMultiSelection Selection,
uint KeyLength);
void Fill(ref byte Pattern,
uint PatternSize,
long Offset,
void FillByte(byte FillValue, ulong Offset, ulong Size);
void FillByteMulti(byte FillValue, IMultiSelection Selection);
void Insert(ref byte Data, ulong Offset, uint InsertSize);
void InsertByte(byte ByteFill, ulong Offset, ulong Size);
void InsertPattern(ref byte Pattern,
uint PatternSize,
ulong Offset,
ulong Size);
void PutFile(string FileName, ulong Offset, bool InsertMode);
void Read(ref byte Data, ulong Offset, uint ReadSize);
void CreatePatch(string PatchName, PatchOptions Options, string PathToStubFile);
void LoadHistory(IStream Stream);
void Purge(HistoryPurgeLevel Level)
void Redo(uint Steps);
void SaveHistory(IStream Stream),
void Undo(uint Steps);
void Copy(IMultiSelection Selection, bool MergeBlocks);
void Cut(IMultiSelection Selection, bool MergeBlocks);
void Paste(ulong Offset, bool InsertMode)
void Paste2(ulong Offset, TextType Type, bool InsertMode);
void Paste3(ulong Offset, TextType Type, Paste3Flags Flags);
void PasteText(ulong Offset, bool Unicode, bool InsertMode);
void PasteText2(ulong Offset, uint CodePage, bool InsertMode);
ulong Find(ref byte DataToFind,
uint DataSize,
IMultiSelection Selection,
ulong StartFrom
bool Searchup,
void FindRegExp(string RegExp,
uint SubExpression,
uint SubExpression,
ulong StartFrom,
ulong Size,
ref ulong FoundOffset,
ref ulong MatchSize);
void FindRegExp2(string RegExp,
uint CodePage,
*int Subexpression
*)
ulong StartFrom,

```
ref ulong Foundoffset,
ref ulong MatchSize);
void FindAll(ref byte DataToFind
uint DataSize,
Mool IgnoreCase,
IMultiselection Found);
void FindAllRegExp(string RegExp,
uint SubExpression,
RegExpFlags Flags,
ulong StartF
ulong Size,
ulong Size,
IMultiSelec
FindllRetirp (sund)
FindAllRegExp2(string RegExp,
uint CodePage,
int SubExpression
ulong StartFrom, ulong Size,
IMultiselection Found);
void Replace(ulong offset,
ref byte To,
uint SizeTo);
void ReplaceAll(ref byte From,
uint SizeFrom,
int SizeTo
bool IgnoreCase,
IMultiSelection Selection);
void ReplaceAllRegExp(string RegExp,
RegExpFlags Flags,
ref byte To,
uint SizeTo,
long StartFrom,
ulong Size);
void ReplaceAllRegExp2(string RegExp,
uint CodePage,
int SubExpression,
RegExpFlags Flags,
re
ulong StartFrom,
void ReplaceAllRegExpWithRegExp(string RegExp,
string Replac
uint CodePage
RegExpFlags Flags,
ulong StartFrom,
ulong Size);
void ArithmeticOp(IMultiSelection Selection
ArithmeticOpType Type,
object Operand);
void BitwiseOp(IMultiSelection Selection, BitwiseOpType Type, ref byte Data, uint DataSize);
void CaseOp(IMultiSelection Selection,
CaseOpType Type,
bool Unicode,
object CodePage);
void Shiftop(IMultiSelection Selection, ShiftopType Type, OpSize, uint Bits);
void Reverseop(IMultiSelection Selection, ReverseOpType Type, OperandSizeType OpSize)
ulong[] Gatherstatistics(IMultiselection Selection);
void
uint DataSize,
bool IgnoreCase,
ref ulong DataBuffer,
uint Blocks,
IMultiSelection Selection);
void PatternStatisticsRegexp(string RegExp,
uint SubExpression,
RegExpFlags Flags,
ref ulong DataBuffer,
uint Blocks,
ulong Start From,
ulong Size);
void PatternStatisticsRegExp2(string RegExp
uint CodePage,
int Subexpression,
RegExpFlags Flags,
ref ulong DataBuffer,
uint Blocks,
ulong Start From,
IMultiSelection CreateEmpty
ISequence CreateSequence()
struct IFileDocument : IDispatch
\(\{\)
```

    // Properties
    L/ // get set
    bstr_t FileName; // get
    unsigned long long FileSize; // get set
    VARIANT_BOOL Modified; // get
    VARIANT_BOOL Modified; // get
    VARIANT_BOOL ReadOnly; // get
    _bstr_t RegExpSyntaxError; // get set
    VARIANT_BOOL CanRedo; // get
    VARIANT_BOOL CanUndo; // get
    VARIANT_BOOL CanPaste; // get
    // Methods
    HRESULT Close();
    hrESULT New();
    HRESULT Open(_bstr_t Name, VARIANT_BOOL ReadOnly);
    HRESULT OpenPhysicalDisk(_bstr_t PhysicalDiskName, _bstr_t FriendlyName, VARIANT_BOOL ReadOnly)
    HRESULT OpenProcess(unsigned long ProcessId, long long Start, unsigned long long Size, VARIANT_BOOL ReadOnly);
    HRESULT OpenVolume(_bstr_t VolumeName, _bstr_t FriendlyName, VARIANT_BOOL ReadOnly);
    HRESULT Save(VARIANT_BOOL ForceBackup)
    HRESULT SaveAs(_bstr_t FileName);
    HRESULT GetModifiedSel(IMultiSelectionPtr Selection)
    HRESULT Decrypt(_bstr_t Provider,
        unsigned long AlgoID,
        bstr_t Password,
        IMultiSelectionPtr Selection,
    ```
            variant + Kavl enoth):
hresult delete (unsigned long long offset, unsigned long long Size);
HRESULT Delete(unsigned long long Offset, unsigned
HRESULT DeleteMulti(IMultiSelectionPtr Selection);
HRESULT Encrypt (_bstr_t Provider,
unsigned long ProviderType,
unsigned long AlgoID,
bstr_t Password,
IMultiselectionPtr Selection,
variant_t KeyLength);
HRESULT Fill(unsigned char * Pattern
unsigned long long Offset,
unsigned long long Size);
HRESULT FillByte(unsigned char FillValue, unsigned long long offset, unsigned long long Size);
HRESULT FillByteMulti(unsigned char FillValue, IMultiSelectionPtr Selection);
HRESULT FillMulti(unsigned char * Pattern, unsigned long Patternsize, Multiselectionptr selection, Continue);
HRESULT Insert(unsigned char * Data, unsigned long long Offset, unsigned long InsertSize);
HRESULT InsertByte(unsigned char ByteFill, unsigned long long Offset, unsigned long long Size);
HRESULT InsertPattern(unsigned char * Pattern,
unsigned long PatternSize,
unsigned long long Offset,
unsigned long long Size);
HRESULT PutFile(_bstr_t FileName, unsigned long long Offset, VARIANT_BOOL InsertMode);
HRESULT Write(unsigned char * Data, unsigned long long Offset, unsigned long ReadSize),
HRESULT CreatePatch(_bstr_t PatchName, PatchOptions Options, _variant_t PathToStubFile);
HRESULT LoadHistory(IStream * Stream);
HRESULT Purge(HistoryPurgeLevel Level);
HRESULT Redo(_variant_t Steps);
HRESULT SaveHistory(IStream
HRESULT Undo(_variant_t Steps);
HRESULT Copy(IMultiSelectionPtr Selection, VARIANT_BOOL MergeBlocks)
HRESULT Paste(unsigned long long Offset, VARIANT_BOOL InsertMode);
HRESULT Paste2 (unsigned long long Offset, TextType Type, VARIANT,BOOL InsertMode);
HRESULT Paste3(unsigned long long Offset, TextType Type, Paste3Flags Flags);
HRESULT PasteText(unsigned long long Offset, VARIANT_BOOL Unicode, VARIANT_BOOL InsertMode);
HRESULT PasteText2(unsigned long long Offset, unsigned long CodePage, VARIANT_BOOL InsertMode);
unsigned long long Find(unsigned char * DataToFind,
nsigned long DataSize,
MultiSelectionPtr Selection,
VARIANT_BOOL SearchUp,
VARIANT_BOOL ignore_case);
HRESULT FindRegExp(_bstr_t RegExp,
unsigned long SubExpression,
RegExpFlags Flags,
unsigned long long StartFrom,
unsigned long long Size,
unsigned long long * FoundOffset,
unsigned long long \(*\) MatchSize)
hResulz ind long (odePage, RegEx
unsigned long CodePage
RegExpFlags Flags,
unsigned long long StartFrom,
unsigned long long Size,
unsigned long long * FoundOffset,
unsigned long long * MatchSize);
HRESULT FindAll(unsigned char * DataToFind, nsigned long DataSize,
VARIANT BOOL Ignotion, IMultiSelectionPtr Found)
HRESULT FindAllRegExp(_bstr_t RegExp,
unsigned long SubExpression,
RegExpFlags Flags,
unsigned long long StartFrom, unsigned long long Size,
HRESULT FindAllRegExp2(_bstr_t RegExp, unsigned long CodePage,
nsigned long SubExpression,
RegExpFlags Flags,
unsigned long long StartFr
unsigned long long Size,
IMultiSelectionPtr Found)
hresult replace(unsigned long long Offset, unsigned long SizeFrom, unsigned char \({ }^{*}\) To,
unsigned long SizeTo);
HRESULT ReplaceAll(unsigned char * From, unsigned char \({ }^{*}\) To
unsigned long SizeTo,
VARIANT_BOOL IgnoreCase,
IMultiSelectionPtr Selection);
HRESULT ReplaceAllRegExp(_bstr_t RegExp, unsigned long SubExpression,
RegExpFlags Flags,
nsigned Char To,
unsigned long SizeTo,
unsigned long long StartFr
ULT ReplaceAllinegExp2 (
HRESULT ReplaceAll RegExp2(_bstr_t RegExp,
unsigned long CodePage,
RegExpFlags Flags,
unsigned char \(*\) To,
unsigned char * To,
unsigned long SizeTo,
unsigned long long Startfr
HRESULT ReplaceAllRegExpWithR
bstr_t ReplacePattern,
insigned long CodeP
RegExpFlags Flags,
unsigned long long Startfro
unsigned long long Size);
HRESULT ArithmeticOp(IMultiSelectionPtr Selection, ArithmeticOpType Type,
perandSizeType OpSize,
variant_t Operand);
HRESULT BitwiseOp(IMultiSelectionPtr Selection, BitwiseOpType Type, unsigned char * Data, unsigned long DataSize)
HRESULT CaseOp(IMultiSelectionPtr Selection,
CaseopType Type,
variant_t CodePage)
HRESULT ShiftOp(IMultiSelectionPtr Selection, ShiftOpType Type, OpSize, unsigned long Bits) HRESULT ReverseOp(IMultiSelectionPtr Selection, ReverseOpType Type, OperandSizeType OpSize); unsigned long long[256] GatherStatistics(IMultiSelectionPtr Selection); HRESULT PatternStatistics(unsigned char * Pattern,

IMultiSelectionPtr Selection)
HRESULT PatternStatisticsRegExp(_bstr_t RegExp,
unsigned long SubE
RegExpFlags Flags,
RegExpFlags Flags,
unsigned long long * DataBuffer,
unsigned long long Startfr
unsigned long long Size);
HRESULT PatternStatisticsRegExp2(_bstr_t RegExp,
unsigned long CodePage,
unsigned long SubExpression,
egExpFlags Flags,
unsigned long long * DataBuffer,
unsigned long blocks,
unsigned long long StartFro
unsigned long long Size);
IMultiSelectionPtr CreateEmptySelection()
ISequencePtr CreateSequence();
\};

IFileDocument Properties

AutoCreateBranch
\begin{tabular}{l} 
TypeScript \\
\begin{tabular}{l} 
AutoCreateBranch: boolean; \\
\hline C\# \\
\hline bool AutoCreateBranch \{ get; set; \} \\
\hline C++ \\
VARIANT_BOOL AutoCreateBranch; // get set \\
\hline
\end{tabular} \\
\hline
\end{tabular}

\section*{Description}
true if Auto Create Branches option enabled, false otherwise.
Query or set the AutoCreateBranch property value. See the History »Auto Create Branches switch description for more information about this property

FileName
\(\frac{\text { TypeScript }}{\text { readonly FileName: string; }}\)

C\#
string FileName \{ get; \}

C++
_bstr_t FileName; // get

\section*{Description}

The full file's path name

\section*{Example}

Using the FileName property
```

JavaScript
var fdoc=new ActiveXObject("FileDocument.FileDocument");
fdoc.open("c:<br>temp<br>file.bin");
alert("Full file name is " + fdoc.FileName);

```

Filesize

\section*{TypeScript}
fileSize: number;

C\#
c\# ulong FileSize \{ get; set; \}
\(+\)
nsigned long long FileSize; // get set

\section*{Description}

Retrieve or change the current file's size. When used to change the file's size, new operation is created in document's operation history.
Complexity: constant-time.
Example
Setting file's size
```

avaScript
var fdoc = new ActiveXObject("FileDocument.FileDocument");
fdoc.New(); // initialize a new, empty document
fdoc.SaveAs("c:<br>temp<br>file.bin");

```

\section*{Modified}
TypeScript
readonly Modified: boolean;
C\#
bool Modified \{ get; \}
C++
VARIANT_BOOL Modified; // get

\section*{Description}

Returns the modified document state.


\section*{Description}

This property is true if document represents an NTFS stream and false otherwise.

ReadOnly
\begin{tabular}{l} 
TypeScript \\
readonly ReadOnly: boolean; \\
\hline C\# \\
\hline bool ReadOnly \{ get; \} \\
\hline C++ \\
VARIANT_BOOL ReadOnly; // get
\end{tabular}

\section*{Description}

Retrieves the document's ReadOnly state.

\section*{RegExpSyntaxError}

TypeScrip
RegExpSyntaxError: string;

C\#

C++
\(\frac{\text { C++ }}{\text { bstr_t }}\) RegExpSyntaxError; // get set

\section*{Description}

Returns a textual representation (in English) of a regular expression syntax error, if any

\section*{CanRedo}
TypeScript

C\# bool CanRedo \{ get; \}

C++
VARIANT_BOOL CanRedo; // get

\section*{Description}

Determine if you can call the IFileDocument.Redo method.

\section*{Example}

Sample implementation of the Redo method
```

$\frac{\text { C++ }}{\text { void Redo(IFileDocument *pFileDocument) }}$
\{ Variant bool bCanRedo;
pFileDocument->get_CanRedo(\&bCanRedo);
if (bCanRedo == VARIANT_TRUE)
pFileDocument->Redo();
\}
Sample implementation of the Redo method

```
```

C\#

```
C#
public void Redo(IFileDocument fdoc)
public void Redo(IFileDocument fdoc)
    if (fdoc.CanRedo)
    if (fdoc.CanRedo)
    fdoc.Redo();
```

    fdoc.Redo();
    ```
CanUnd
\(\frac{\text { TypeScript }}{\text { readonly CanUndo: boolean; }}\)
C\# bool CanUndo \{ get; \}
\(\stackrel{C+}{\text { C+ }}\)
VARIANT_BOOL CanUndo; // get

Description
Determine if you can call the IFileDocument.Undo method.

\section*{Example}

Sample implementation of the Undo method
```

C++
VARIANT_BOOL bCanUndo;
pFileDocument->get_CanUndo(\&bCanUndo);
if (bCanUndo == VARIANT_TRUE)
pFileDocument->Undo();
}
Sample implementation of the Undo method

```
```

C\#

```
C#
{ if (fdoc.CanUndo)
{ if (fdoc.CanUndo)
    fdoc.Undo();
```

    fdoc.Undo();
    ```
CanPaste
TypeScript
readonly CanPaste: boolean;
C\#
bool CanPaste \{ get; \}
\(\frac{\text { C++ }}{\text { VARIANT_BOOL CanPaste; // get }}\)

\section*{Description}

Determine if you can paste data from the Clipboard.

\section*{Example}
```

C++
ARTANT BOOL bCanPaste;
pFileDocument->get_CanPaste(\&bCanPaste);
if (bCanPaste != VARIANT_FALSE)
if (bCanPaste != VARIANT_FALSE)
else
}
// Clipboard contains incompatible data

```

\section*{FileDocument Methods}
```

TypeScrip
Close(): void;
void Close();
C++
HRESULT Close()

```

\section*{Description}

Discard document's operation history and close the document. You may subsequently call the IFileDocument.New or IFileDocument.Open method for this document. Complexity: constant-time

\section*{Example}

Reusing document objects
C\#
var fdoc \(=\) new FileDocumentLib.FileDocument ();
fdoc.Open(@"c:\temp\file1");
// perform modifications to the file and save it
fdoc.Save(false);
doc.Save(false)
// prepare the object to work with a next document
fdoc.Close();
fdoc.Open(@"c:\temp\file2");
// ..

New

TypeScript
New(): void;

C\#
oid New();
++
hresult New();

\section*{Description}

Creates a new, empty document.
Complexity: constant-time.

Open

TypeScript
Open(Name: string, ReadOnly: boolean): void;

C\#
Open(string Name, bool ReadOnly)

HRESULT Open(_bstr_t Name, VARIANT_BOOL ReadOnly);

\section*{Parameters}

Full path to the opened file.
Readonly
Force read-only mode (the IFileDocument.Save method is disabled).

\section*{Description}

Opens an existing document. Hex Editor Neo tries to get a read-only access to the file. It also denies write access to all future open attempts, until the file is closed.
Complexity: constant-time
If supported by the file system, the name may be actually the name of the alternate data stream. Hex Editor Neo fully supports NTFS alternate data streams You may use any method provided by the object after opening the file stream. Note however, that there are few limitations that are described in the IFileDocument.Save method section.

\section*{Example}

Opening a document
```

C\#
var fdoc = new FileDocumentLib.FildDocument();
fdoc.Open(@"<br>server\share\somefile.bin", false);

```

OpenPhysicalDisk

TypeScript
OpenPhysicalDisk(PhysicalDiskName: string, FriendlyName: string, ReadOnly: boolean): void;
```

C\#
oid OpenPhysicalDisk(string PhysicalDiskName, string FriendlyName, bool ReadOnly);
HRESULT OpenPhysicalDisk(_bstr_t PhysicalDiskName, _bstr_t FriendlyName, VARIANT_BOOL ReadOnly);

```

\section*{Parameters}

\section*{PhysicalDiskName}

The string of form \(\backslash \backslash . \backslash\) PhysicalDriven, where N is a physical disk number (starting from 0 ),

\section*{FriendlyName}

Friendly name. Used only in the editor. You may pass an empty string to this parameter.

\section*{ReadOnly}

Specify true to open a disk as read-only, or false otherwise.
Description
Open physical disk

OpenProces
ypeScrip
OpenProcess(ProcessId: number, Start: number, Size: number, ReadOnly: boolean): void;

C\#
void OpenProcess(uint ProcessId, long Start, ulong Size, bool ReadOnly);

C++ HRESULT OpenProcess(unsigned long ProcessId, long long Start, unsigned long long Size, VARIANT_BOOL ReadOnly)

\section*{Parameters}

\section*{ProcessId}

Process Identifie
Start
Starting memory address
Size
Size of the range to open
ReadOnly
true to disable the Save command, false otherwise.

\section*{Description}

Open process memory.

OpenVolume

TypeScript
TypeScript OpenVolume(VolumeName: string, FriendlyName: string, ReadOnly: boolean): void;
"
void OpenVolume(string VolumeName, string FriendlyName, bool ReadOnly);

C++
HRESULT OpenVolume(_bstr_t VolumeName, _bstr_t FriendlyName, VARIANT_BOOL ReadOnly);

\section*{Parameters}

\section*{VolumeName}

The name of the volume to open. May be a string of form \(\backslash \backslash . \ c\) : to open a drive letter, or a full volume mount point name

\section*{FriendlyName}

The friendly name. Used only in the editor. You may pass an empty string in this parameter
ReadOnly
Specify true to open a volume as read-only, or false otherwise

\section*{Description}

Open a volume (logical disk)

Save

TypeScript
Save(ForceBackup: boolean): void

C\#
oid Save(bool ForceBackup);

HRESULT Save(VARIANT_BOOL ForceBackup);

\section*{Parameters}

\section*{ForceBackup}

Force creation of backup copy of the file.

\section*{Description}

Saves all document's changes and drop document's operation history. IFileDocument.FileName property must not be empty to successfully call this method. In particular, the Save method may not be called after the IFileDocument.New method. IFileDocument.SaveAs method always set the IFileDocument. FileName property, so you may call the Save method after the SaveAs method was once called for a document.

ForceBackup parameter is ignored (assumed to be false) if the current object represents the NTFS alternate data stream, physical disk, volume, process virtual memory or physical memory.

SaveAs
```

TypeScript
SaveAs(FileName: string): void;

```
void SaveAs(string FileName)

C++

\section*{Parameters}

\section*{FileName}

Full name of the file to write

\section*{Description}

Saves the file with a different name. The Save As command does not drop a document's operation history. Passing the name of an NTFS alternate data stream to this method is an error and leads to undefined behavior.

To save a document to alternate data stream, first create a stream (this function is outside the scope of this library), open the File Document Object for created stream and use the IFileDocument.Copy and IFileDocument. Paste methods to copy the contents. Then call the created object's IFileDocument.Save method to apply changes.


TypeScript
GetModifiedSel(Selection: IMultiSelection): void;
void GetModifiedSel(IMultiSelection Selection)

C++
HRESULT GetModifiedSel(IMultiSelectionPtr Selection);

\section*{Parameters}

\section*{Selection}

Multiple Selection object to be filled with modified ranges.

\section*{Description}

Fills a provided Multiple Selection object with ranges that describe the changed data in the document. The method clears the given multiple selection object before execution.

Complexity: linear-time, depending on the number of document modifications

\section*{Example}

Using GetModifiedSel method

C\# var fdoc = new FileDocumentLib.FileDocument();
var fdoc = new FileDocumentLib.
fdoc. FillByte (0x35, 100, 20) ;
doc.FillByte(0x36, 200, 30)
var msel = fdoc.CreateEmptyselection();
fdoc.GetModifiedSel(msel);
// msel is now: [100..120) U [200..230)

Decrypt

\section*{TypeScript}

Decrypt(Provider: string,
ProviderType: number,
AlgoID: number,
Password: string,
Selection: IMultiSelection
KeyLength?: number): void;
```

C\#
uint ProviderType
uint AlgoID,
string Password
String Password,
uint KeyLength);
C++

```

```

    unsigned long ProviderType
    bstr t Password,
    IMultiSelectionPtr Selection,
    _variant_t KeyLength)
    ```

\section*{Parameters}

\section*{Provider}

The name of the encryption provider. Consult the Microsoft Cryptography API documentation for more information.

\section*{ProviderType}

The provider type. Consult the Microsoft Cryptography API documentation for more information

\section*{Algoid}

Algortihm identifier
Password
String containing a password
Selection
Multiple selection object specifying data to encrypt

\section*{KeyLength}

Key length, for supported algorithms.

\section*{Description}

Decrypt the contents of the document

Delete

TypeScript
Delete(Offset: number, Size: number): void;
void Delete(ulong Offset, ulong Size);

C++
HRESULT Delete(unsigned long long Offset, unsigned long long Size);

\section*{Parameters}

Offset
Range offset.
Size
Range size.
Description
Deletes the given range from the document
Complexity: constant-time

\section*{Example}

Deleting document's data
```

f (SUCCEEDED(pFileDocument.CoCreateInstance(L"FileDocument.FileDocument")))
{
if (SUCCEEDED(pFileDocument->Open(L"c:<br>temp<br>file.bin", VARIANT_FALSE))
{
pFileDocument->Delete(0x50, 0x2a)
pFileDocument->Save(false);
}

```

DeleteMulti

\section*{ypeScript}

DeleteMulti(Selection: IMultiSelection): void;

C\#
oid DeleteMulti(IMultiSelection Selection);

RESULT DeleteMulti(IMultiSelectionPtr Selection)

\section*{Selection}

Multiple selection object, which describes the ranges to remove from the document

\section*{Description}

Deletes the given multiple selection from the document.
Complexity: linear-time, depending on the selection's complexity.

\section*{Example}

Deleting document's data
C++
\(\frac{\text { C++ }}{\text { CComPtr }<\text { IFileDocument }>~ p F i l e D o c u m e n t ; ~}\)
if (SUCCEEDED(pFileDocument.CoCreateInstance(L"FileDocument.FileDocument")))
\{
if (SUCCEEDED(pFileDocument->Open(L"c:\\temp\\file.bin", VARIANT_FALSE)))
\{ CComPtr<IMultiSelection> pMultiSelection
if (SUCCEEDED(pFileDocument->CreateEmptySelection(\&pMultiSelection)))
pMultiSelection->AddRange(100, 20);
pMultiSelection->AddRange(200, 30);
pFileDocument->DeleteMulti(pMultiSelection)
pFileDocument->Save(false);
\}
\(\}^{\}}\)

Encrypt
```

ypeScript
Encrypt(Provider: string,
ProviderType: number,
AlgoID: number,
AlgoID: number,
Selection: IMultiSelection,
Keylength?: number): void;

```
void Encrypt(string Provider,
    uint ProviderType
    uint AlgoID,
    string Password
    IMultiSelection Selection,
    uint KeyLength).
\(\frac{\text { C++ }}{\text { HRESULT Encrypt(_bstr_t Provider, }}\)
    unsigned long Providertype,
    unsigned long AlgoID,
    _bstr_t Password,
    IMultiselectionPtr Selection,
    _variant_t KeyLength);

\section*{Parameters}

Provider
The name of the encryption provider. Consult the Microsoft Cryptography API documentation for more information.

\section*{ProviderType}

The provider type. Consult the Microsoft Cryptography API documentation for more information
AlgoID
Algortihm identifier
Password
String containing a password

\section*{Selection}

Multiple selection object specifying data to encrypt.

\section*{KeyLength}

Key length, for supported algorithms

\section*{Description}

Decrypt the contents of the document.

Fill

\section*{TypeScript}
// This method is not available in scripting environment

C\#
```

id Fill(ref byte Pattern,
uint PatternSize,
ulong Offset

```
C++
HRESULT Fill(unsigned char * Pattern,
    unsigned long Patternsize,
    unsigned long long Offset,
    unsigned long long Size);

\section*{Parameters}

\section*{Pattern}

Pointer to a pattern.
PatternSize
Pattern size, in bytes.
Offset
Range starting offset.
Size
Range size.

\section*{Description}

Fills a given range with a given pattern. A pattern is repeated until the required number of bytes is written to the file. If file is shorter than the filling range, the file's size is increased.

Complexity: constant-time

\section*{Example}

Filling a range with a simple pattern

C++ pFileDocument is initialized elsewhere
unsigned char Pattern[]="Simple Pattern";
HRESULT hRes = pFileDocument->Fill(Pattern, sizeof(Pattern) - 1, 0x10000i64, 0x10ffi64);

Fill Byte

TypeScrip
FillByte(FillValue: number, Offset: number, Size: number): void;
void FillByte(byte FillValue, ulong Offset, ulong Size);

HRESULT FillByte(unsigned char FillValue, unsigned long long Offset, unsigned long long Size)

\section*{Parameters}

Fillvalue
The byte to put in the range.
offset
Range offset.
Size
Range size.

\section*{Description}

Fills a given range with a single byte value.
Complexity: constant-time
Example
Filling a range with a single byte
```

C\#
/l fdoc is initialized elsewher
fdoc.FillByte(0x34, 1000, 200);

```

\section*{FilisyteMult}

TypeScript
FillByteMulti(FillValue: number, Selection: IMultiSelection): void;

C\#
oid FillByteMulti(byte FillValue, IMultiSelection Selection);

HRESULT FillByteMulti(unsigned char FillValue, IMultiSelectionPtr Selection);

\section*{Parameters}

\section*{Fillvalue}

The byte to put in the selection.
Selection
The multiple selection object, which contains the ranges to fill in the document

\section*{Description}

Fills a given multiple selection with a single byte value
Complexity: linear-time, depending on the selection's complexity.

\section*{Example}

Filling a range with a single byte
```

C\#
// fdoc is initialized elsewhere
var msel = fdoc.CreateEmptySelection();
sel.AddRange(100, 20);
fdoc.FillByteMulti(0\times35, msel)

```
FillMulti

\section*{TypeScript}

TypeScript
c\# void FillMulti(ref byte Pattern, uint PatternSize, IMultiSelection Selection, Continue);

HRESULT FillMulti(unsigned char * Pattern, unsigned long PatternSize, IMultiSelectionPtr Selection, Continue);

\section*{Parameters}

\section*{Pattern}

Pointer to a pattern

\section*{PatternSize}

Pattern Size
Selection
The multiple selection object, which contains the ranges to fill in the document

\section*{Continue}

True to continue filling in a new range and False to start from the beginning of the pattern. See the description of the "Transparent fill" flag in the Fill user-interface command for more information

\section*{Description}

Fills a given multiple selection with a specified pattern. The pattern is repeated until all given range(s) are filled. bCont parameter specifies whether to continue filling a new range from the previous range, or from the beginning of the pattern

Complexity: linear-time, depending on the selection's complexity.

\section*{Example}

Filling a range with a single byte
```

C\#
// fdoc is initialized elsewhere

```
var msel = fdoc.CreateEmptySelection();
msel.AddRange(100, 20);
msel.AddRange(200, 30);
byte[] pattern = new byte[] \{ 0x31, 0×32, 0×33 \};
fdoc.FillMulti(ref pattern[0], pattern.Length, msel, false);
Insert
TypeScript
This method is not available in scripting environment
C\#

C++
HRESULT Insert(unsigned char * Data, unsigned long long Offset, unsigned long InsertSize);

\section*{Parameters}

Data
Pointer to a pattern.
offset
Insert offset

\section*{InsertSize}

Pattern size

\section*{Description}

Inserts a pattern into the document. Insert command shifts file's data to free up the space to fit an inserted pattern.
Complexity: constant-time.

\section*{Example}

Inserting a pattern

\footnotetext{
C\#
\# fdoc is initialized elsewhere
yte[] pattern = new byte[] \{ 0x0d, 0x0a \}
doc.Insert(ref pattern[0], 100, pattern.Length); // insert a given pattern at the offset 100
}
```

InsertByte
TypeScript
InsertByte(ByteFill: number, Offset: number, Size: number): void;
void InsertByte(byte ByteFill, ulong Offset, ulong Size);
C++
HRESULT InsertByte(unsigned char ByteFill, unsigned long long Offset, unsigned long long Size);

```

\section*{Parameters}
```

ByteFill
The byte value to insert.
Offset
Insert offset
Size
Insert size.

```

\section*{Description}

Inserts a block of a given size into the document and fill it with a given byte value
Complexity: constant-time

\section*{Example}

Inserting a pattern
```

C\#
fdoc is initialized elsewhere
fdoc.InsertByte('\#', 100, 10000)

```

\section*{Inserpater}

\section*{TypeScript}
// This method is not available in scripting environment

C\#
id InsertPattern(ref byte Pattern,
uint PatternSize, ulong Size);

HRESULT InsertPattern(unsigned char * Pattern,
unsigned long Patternsize,
unsigned long long Offset,
unsigned long long Size);

\section*{Parameters}

\section*{Pattern}

Pointer to a pattern.

\section*{Patternsize}

Pattern Size
offset
Insert offset
Size
Insert size

\section*{Description}

Inserts a given pattern into the document, repeating it to fill the given size. Insert command shifts file's data to free up the space to fit an inserted pattern Complexity: constant-time

\section*{Example}

Inserting a pattern
C\#
// fdoc is initialized elsewhere
byte[] pattern = new byte[] \{ 0x0d, 0x0a \}
doc.InsertPattern(ref pattern[0], pattern.Length, 100, 1000);

PutFile
```

ypeScript
PutFile(FileName: string, Offset: , InsertMode: boolean): void;
void PutFile(string FileName, ulong Offset, bool InsertMode);

```

C++ \(H R E S U L T\) PutFile(_bstr_t FileName, unsigned long long Offset, VARIANT_BOOL InsertMode);

\section*{Parameters}

\section*{FileName}

The full path name of the file
offset
Insert offset.

\section*{InsertMode}
true to insert a file's contents, false to overwrite the current document's data with data from the file.

\section*{Description}

Writes or inserts the contents of the file into the current document. InsertMode parameter governs the behavior of the function.
Complexity: constant-time

\section*{Example}

Concatenating Files

JavaScript
var fdoc = new ActiveXObject("FileDocument.FileDocument");
fdoc.Open("c:\\temp\\file1.bin");
fdoc.PutFile("c:\\temp\\file2.bin", fdoc.FileSize, false);
fdoc.Save(false);

Read

TypeScrip
// This method is not available in scripting environment
void Read(ref byte Data, ulong Offset, uint ReadSize);

C++
HRESULT Read(unsigned char * Data, unsigned long long Offset, unsigned long ReadSize);

\section*{Parameters}

Data
A pointer to the buffer that receives the data read from a file
offset
Read offset.
ReadSize
Total number of bytes to read.

\section*{Description}

Reads data from a document. If requested data exceeds the current document's size, any excess bytes are filled with zeros.
Complexity: constant-time

\section*{Example}

Reading Data
```

C++

```
nsigned char data[256]
pFileDocument->Read(data, 1000, 256);

Write

\section*{TypeScript}
// This method is not available in scripting environment

C\#
void Write(ref byte Data, ulong Offset, uint WriteSize);

HRESULT Write(unsigned char * Data, unsigned long long Offset, unsigned long WriteSize);

\section*{Parameters}

Data
Pointer to the buffer containing the data to be written to the file.
offset
Write offset
WriteSize
Total number of bytes to write.

\section*{Description}

Writes data to a document. The written data overwrites document's existing data and may also increase the file's size if it overlaps the current file's size. Write always creates a new operation in document's operation history

Complexity: constant-time.

\section*{Example}

Writing Data
// pFileDocument is initialized elsewhere
unsigned char data[256];
pFileDocument->Write(data, 1000, 256);

CreatePatch
ypeScript
CreatePatch(PatchName: string, Options: PatchOptions, PathToStubFile?: string): void;

C\#
void CreatePatch(string PatchName, PatchOptions Options, string PathToStubFile);
++
HRESULT CreatePatch(_bstr_t PatchName, Patchoptions Options, _variant_t PathToStubFile);

\section*{Parameters}

\section*{PatchName}

The full path of the patch file. If file already exists, it is overwritten. Options

Patch options. See [PatchOptions] enumeration for possible values

\section*{PathToStubFile}

An optional path to an executable stub file. Should be a full path to either PatchApply32.exe or PatchApply64.exe application, installed with the Hex Editor Neo.

\section*{Description}

Creates a patch file. See the Patch Creation section for more information on patches
Complexity: linear-time, depending on the number and complexity of operations in document's operation history
Example
Creating patch

C\#
// fdoc is initialized elsewhere
Perform a modication
doc CreatePatch (0"c: 1pat
fdoc.CreatePatch(@"c:\patch.hexpatch", FileDocumentLib.PatchOptions.PatchCalculateHash | FileDocumentLib.PatchOptions.PatchSaveFileName, null);

\section*{LoadHistor}

\section*{TypeScript}
// This method is not available in scripting environment
c\#
\(++\)
HRESULT LoadHistory(IStream * Stream);

Parameters

\section*{Stream}

Pointer to a stream object from which to load history.

\section*{Description}

Loads a document's operation history from a file.
Complexity: linear-time, depending on the saved history's complexity

Purge

TypeScrip
TypeScript
Purge(Level: HistoryPurgeLevel): void

C\#
oid Purge(HistoryPurgeLevel Level);

HRESULT Purge(HistoryPurgeLevel Level);

\section*{Parameters}

Level

Purge level. Can be one of the values described in the HistoryPurgeLevel topic.

\section*{Description}

Purges a document's operation history. See Purging History section for more information.
Complexity: constant-time for all purge levels except HistoryPurgeAllButCurrent and linear-time for HistoryPurgeAllButCurrent

\section*{Example}
```

Purging history
C\# fdoc is initialized elsewhere
fdoc.FillByte(0x35, 100, 40);
fdoc.Purge(FileDocumentLib.HistoryPurgeLevel.HistoryPurgeAll); // discard all operations
Redo
TypeScript
TypeScript
C\#
void Redo(uint Steps);
C++
HRESULT Redo(variant t Steps)

```

\section*{Parameters}

Steps
Optional number of operations to redo. If omitted, one operation is redone.

\section*{Description}

Redoes Steps recently undone operations.
Complexity: constant-time

SaveHistory

TypeScript
// This method is not available in scripting environment

C\#
void SaveHistory(IStream Stream)

C++
HRESULT SaveHistory(IStream * Stream)

\section*{Parameters}

Stream
Pointer to a stream object to write history to
Description
Saves the document's operation history to a given stream object.
Complexity: linear-time, depending on the history's complexity.

Undo

TypeScript
Undo(Steps?: number): void;
void Undo(uint Steps)

C++
HRESULT Undo(_variant_t Steps);

\section*{Parameters}

Steps
Optional number of operations to undo. If omitted, one operation is undone

\section*{Description}

Undoes last Steps operations.
Complexity: constant-time

TypeScript
Copy(Selection: IMultiSelection, MergeBlocks: boolean): void
void Copy(IMultiSelection Selection, bool MergeBlocks);
++
HRESULT Copy(IMultiSelectionPtr Selection, VARIANT_BOOL MergeBlocks);

\section*{Parameters}

Selection
The multiple selection object.
MergeBlocks
true to merge selection blocks, false to leave them as is

\section*{Description}

Copies the given selection to the Clipboard
Complexity: linear-time, depending on the selection's complexity.

\section*{Example}

Placing document's data to the Clipboard

C\# fdoc is defined and obtained elsewhere
var msel \(=\) fdoc.CreateEmptySelection(); // create an empty selection object
msel.AddRange (100, 20); // add a first range to the selection
sel.AddRange 200,20 ); // add a second range to the selection
fdoc.Copy(msel, false); // place data to the clipboard

Cut
ypeScript
Cut(Selection: IMultiSelection, MergeBlocks: boolean): void;

C\#
void Cut(IMultiSelection Selection, bool MergeBlocks);

C++
HRESULT Cut(IMultiSelectionPtr Selection, VARIANT_BOOL MergeBlocks)

\section*{Parameters}

Selection
The multiple selection object.
MergeBlocks
true to merge selection blocks, false to leave them as is

\section*{Description}

Copies the given selection to the Clipboard and removes it from the document
Complexity: linear-time, depending on the selection's complexity.

\section*{Example}

Cutting document's data to the Clipboard
// fdoc is defined and obtained elsewhere
var msel = fdoc.CreateEmptySelection(); // create an empty selection objec
msel.AddRange(100, 20); // add a first range to the selection
msel.AddRange(200, 20); // add a second range to the selection
foc.Cut(msel, false); // place data to the clipboard

\section*{Paste}

\section*{ypescript}

Paste(Offset: number, InsertMode: boolean): void;

C\#
void Paste(ulong Offset, bool InsertMode);

C++

\section*{Parameters}

Offset
Paste offset

\section*{InsertMode}
true to insert data, false to overwrite data

\section*{Description}

Pastes data from the Clipboard to the given position. Pasted data either overwrites the current document's data, or is inserted into the document, shifting existing data forward, depending on the InsertMode parameter.

This method is superseded by IFileDocument.Paste2 method, but is still supported.

\section*{Example}

Using Paste
var fdoc \(=\) new FileDocumentLib. FileDocument();
var msel = fdoc.CreateEmptySelection();
nsel.AddRange(0, 10);
fooc.Copy(msel, false);
fdoc.Paste(20, true); // Equivalent to fdoc.Paste2(20, TypeHexByte, true);
fdoc.Paste(100, false); // Equivalent to fdoc.Paste2(100, TypeHexByte, false);

Paste2

Typaste2(Offset: number, Type: TextType, InsertMode: boolean): void;

C\#
void Paste2(ulong Offset, TextType Type, bool InsertMode);

HRESULT Paste2(unsigned long long Offset, TextType Type, VARIANT_BOOL InsertMode)

\section*{Parameters}

\section*{offse}

Paste offset
Type
Type of text in the Clipboard. Used only if Clipboard contains textual data, otherwise ignored. Can be one of the values from TextType enumeration. A number may be prefixed by the 0x prefix which forces it to be in base 16

\section*{InsertMode}
true to insert data, false to overwrite data

\section*{Description}

Pastes data from the Clipboard to the given position. Pasted data either overwrites the current document's data, or is inserted into the document, shifting existing data forward, depending on the InsertMode parameter.

\section*{Example}

Using Paste
```

C\#
*ar Fdoc = new FileDocumentlib, FileDOCument();
var msel = fdoc.CreateEmptySelection();
sel.AddRange(0, 10);
doc.Copy(msel, false);
fdoc.Paste2(20, TextType.TypeHexByte, true);
foc.Paste2(100, TextType.TypeHexByte, false);

```

Paste3
```

TypeScript
Paste3(Offset: number, Type: TextType, Flags: Paste3Flags): void;

```
C\#
void Paste3(ulong Offset, TextType Type, Paste3Flags Flags);
HRESULT Paste3(unsigned long long Offset, TextType Type, Paste3Flags Flags);

\section*{Parameters}

\section*{offset}

Paste offset
Type
Type of text in the Clipboard. Used only if Clipboard contains textual data, otherwise ignored. Can be one of the values from TextType enumeration. A number may be prefixed by the ex prefix which forces it to be in base 16.

\section*{Flags}

Pasting flags. Can be one or more values from the Paste3Flags enumeration.

\section*{Description}

Pastes data from the Clipboard to the given position. Pasted data either overwrites the current document's data, or is inserted into the document, shifting existing data forward, depending on the InsertMode flag in Flags parameter

TypeScript
PasteText(Offset: number, Unicode: boolean, InsertMode: boolean): void;
void PasteText(ulong Offset, bool Unicode, bool InsertMode);

C++
HRESULT PasteText(unsigned long long Offset, VARIANT_BOOL Unicode, VARIANT_BOOL InsertMode);

\section*{Parameters}
offset
Paste offse
Unicode
Paste text as Unicode (true) or ANSI (false).
InsertMode
true to insert data, false to overwrite data

\section*{Description}

Pastes text from the Clipboard to the given position. Pasted data either overwrites the current document's data, or is inserted into the document, shifting existing data forward, depending on the InsertMode parameter.

Unicode parameter tells the Hex Editor Neo to interpret the text in the Clipboard as UNICODE or ANSI. IUnicode is true, each character occupies two bytes, otherwise, it occupies a single byte.
```

PasteText2
TypeScript
PasteText2(Offset: number, CodePage: number, InsertMode: boolean): void;
void PasteText2(ulong Offset, uint CodePage, bool InsertMode);
C++
RRESULT PasteText2(unsigned long long Offset, unsigned long CodePage, VARIANT_BOOL InsertMode);

```

\section*{Parameters}
```

offset
Paste offset

```

\section*{CodePag}
```

The code page to convert text to. Set to CP_UNICODE (hardcoded as -5 ) to specify UTF-16 encoding

```

\section*{InsertMode}
```

True to insert data, False to overwrite data.

```

\section*{Description}

Pastes text from the Clipboard to the given position. Pasted data either overwrites the current document's data, or is inserted into the document, shifting existing data forward, depending on the InsertMode parameter.

Hex Editor Neo uses the CodePage to convert the text in the Clipboard. Pass the CP_UNICODE (hardcoded as -5) to specify the UTF-16 encoding.
This method requires the CF_UNICODETEXT format to be present in the Clipboard in order to convert to the given encoding. If only CF_TEXT is available, than it operates exactly like the IFileDocument.PasteText method

Find

\section*{TypeScrip}
// This method is not available in scripting environment
\#
uint DataSize,
IMultiSelection Selection,
ulong StartFrom,
bool SearchUp,
bool ignore_case)

C++
unsigned long long Find(unsigned char * DataToFind,
signed long DataSize
Multiselectionptr Selection,
VARIANT_BOOL SearchUp,
VARIANT_BOOL ignore_case);

\section*{Parameters}

\section*{DataToFind}

Pointer to a pattern.

\section*{DataSiz}

Pattern size
Selection
Multiple selection object, which contains ranges in which you want to locate a pattern.

\section*{StartFrom}

\section*{Offset from which you want to start searching}

Searchup
true to search backwards and false to search forward.
ignore_case
true to ignore case and false to perform exact matching.

\section*{Return Value}

Offset of the located pattern or - 1 if pattern was not found

\section*{Description}

Searches for a pattern within a given selection. This method returns the offset of the matched pattern. To continue searching, call this method again, adjusting StartFron parameter appropriately. If pattern is not found in the specified range, -1 is returned

Complexity: linear-time, depending on the file's size and selection's complexity.

\section*{Example}

Searching for a pattern

C\#
var fdoc \(=\) new FileDocumentLib.FileDocument();
var msec = fdoc.CreateEmptySelection();
fdoc.Open(@"c:\temp\file.txt");
sec.AddRange(0, fdoc.FileSize); // We'll be searching in a whole file
byte[] pattern = new byte[] \{ 0xөd, 0xөa \}; // EOL pattern
ulong EOL_Offset \(=\) fdoc.Find(ref pattern[0], pattern.Length, msel, 0, false, false);
// continue searching
EOL_Offset = fdoc.Find(ref pattern[0], pattern.Length, msel, EOL_Offset + 1, false, false);

\section*{FindRegExp}

TypeScrip
// This method is not available in scripting environment
void FindRegExp(string RegExp,
uint SubExpression,
RegExpFlags Flags
ulong StartFro
ulong Size
FoundOffset,
ref ulong MatchSize);

C++
HRESULT FindRegExp (_bstr_t RegExp,
unsigned long SubExpression,
RegExpFlags Flags,
unsigned long long StartFrom,
unsigned long long Size,
unsigned long long * FoundOffset,
unsigned long long * MatchSize);

\section*{Parameters}

RegExp
A string that contains a regular expression. A regular expression must by in ECMAScript syntax
SubExpression
The number of sub-match to search for. 0 means the entire expression.
Flags
Flags that change the behavior of the function. May be one or more of values from RegExpFlags enumeration.

\section*{StartFrom}

A start offset of the range. Must be a multiple of 2 if RegExpTypeUNICODE flag is specified.
Size
Range's size. Must be a multiply of 2 if RegExpTypeUNICODE flag is specified.
Foundoffset
On output, the found offset is stored in this parameter

\section*{MatchSize}

On output, the match's size is stored in this parameter

\section*{Description}

Searches for an occurrence of a regular expression within a given range. To continue searching, call this method again, adjusting Startfrom field appropriately. If a pattern is not found in the specified range, this method returns an error

Complexity: depends on the range's size and regular expression complexity.
If regular expression syntax is invalid, the method returns (or throws) an error E_INVALIDARG. A IFileDocument.RegExpSyntaxError property contains the description of the syntax error.

When match is not found, the method returns (or throws) a NOT_FOUND error
This function is obsolete. Use the IFileDocument.FindRegExp2 instead.
```

FindRegExp2
TypeScrip
// This method is not available in scripting environment
C\#
id FindRegExp2(string RegExp,
uint CodePage,
uint SubExpression,
RegExpFlags Flags,
ulong Start
rof ulong Foundoffset,
ref ulong MatchSize);
C++
RESULT FindRegExp2(_bstr_t RegExp,
unsigned long CodePage,
unsigned long SubExpression,
RegExpFlags Flags
unsigned long long StartFrom,
unsigned long long Size,
unsigned long long Size,
unsigned long long * MatchSize);

```

\section*{Parameters}

\section*{RegExp}
```

A string that contains a regular expression. A regular expression must by in ECMAScript syntax
CodePage
Code page for regular expression pattern encoding or CP_UNICODE (-5) for UTF-16
SubExpression
The number of sub-match to search for. 0 means the entire expression.

```

\section*{Flags}
```

Flags that change the behavior of the function. May be a combination of values from the RegExpFlags enumeration.

```

\section*{StartFrom}
```

A start offset of the range. Must be a multiple of 2 if RegExpTypeUNICODE flag is specified.

```

\section*{Size}
```

Range's size. Must be a multiply of 2 if RegExpTypeUNICODE flag is specified.

```

\section*{foundoffse}
```

On output, the found offset is stored in this parameter

```

\section*{MatchSize}
```

On output, the match's size is stored in this parameter

```

\section*{Description}

Searches for an occurrence of a regular expression within a given range. To continue searching, call this method again, adjusting Startfrom field appropriately. If a pattern is not found in the specified range, this method returns an error.

Complexity: depends on the range's size and regular expression complexity.
If regular expression syntax is invalid, the method returns (or throws) an error E_INVALIDARG. A IFileDocument.RegExpSyntaxError property contains the description of the syntax error.

When match is not found, the method returns (or throws) a NOT_FOUND error

FindAll

\section*{TypeScrip}
// This method is not available in scripting environment

C\#
oid FindAll(ref byte DataToFind
int DataSize,
Multiselection Selection,
IMultiSelection Found)

C++
RESULT FindAll(unsigned char * DataToFind,
unsigned long DataSize,
IMultiSelectionPtr Selection,
VARIANT_BOOL IgnoreCase,

Parameters
DataToFind
Pointer to a pattern.

\section*{DataSize}

Pattern size

\section*{Selection}

Multiple Selection object, which contains ranges in which a pattern need to be located

\section*{IgnoreCase}
true to ignore case, false to perform exact matching
Found

Multiple Selection object, which contains, on method's return, all located ranges.

\section*{Description}

Locates all occurrences of a given pattern within a given multiple selection. You provide the function with an output Multiple Selection Object Found, in which it stores all located ranges. This object is automatically cleared before the operation begins. When the method returns, check the IMultiSelection.Empty property to see if there were any pattern occurrences.

Complexity: linear-time, depending on the file's size and selection's complexity.

\section*{Example}

Searching all pattern occurrences
C\#
var fdoc = new FileDocumentLib.FileDocument();
= fdoc.CreateEmptySelection();
var mselOutput \(=\) fdoc.CreateEmptySelection();
sel.AddRange(0, fdoc.FileSize); // We'll be searching in a whole file
byte[] pattern = new byte[] \{ 0x0d, 0x0a \}; // EOL pattern
fdoc.FindAll(ref pattern[ 0 ], pattern. Length, msel, false, mseloutput);
if (!mseloutput.Empty) // check if there are any matches
fdoc.Copy(mselOutput, false); // copy them to the Clipboard

FindAlliRegExp
```

TypeScript
FindAllRegExp(RegExp: string
SubExpression: number
Flags: RegExpFlags,
tartFrom: number
Size: number
Fize: number,
C\#
oid FindAllRegExp(string RegExp
uint SubExpression,
RegExpFlags Flags,
ulong StartFro
ulong Size,
IMultiSelection Found);

```
```

C++
RESULT FindAllRegExp(_bstr_t RegExp,

```
    unsigned long SubExpression,
    RegExpFlags Flags,
    unsigned long long StartFrom,
    unsigned long long Size,
    IMultiSelectionPtr Found);

\section*{Parameters}

\section*{RegExp}

A string that contains a regular expression. A regular expression must by in ECMAScript syntax.

\section*{Subexpression}

The number of sub-match to search for. 0 means the entire expression

\section*{Flags}

Flags that change the behavior of the function. May be one or more values from the RegExpFlags enumeration.

\section*{StartErom}

A start offset of the range. Must be a multiple of 2 if RegExpTypeUNICODE flag is specified.
Size
Range's size. Must be a multiply of 2 if RegExpTypeUNICODE flag is specified
Found
Multiple Selection object, which contains, on method's return, all located ranges.

\section*{Return Value}

A number of occurrences

\section*{Description}

Locates all matches of a given regular expression in a given range. You provide the function with an output Multiple Selection Object Found, in which it stores all located ranges. This object is automatically cleared before the operation begins. When the method returns, check the IMultiSelection.Empty property to see if there were any pattern occurrences.

Complexity: depends on the range's size and regular expression complexity.
This method is obsolete. Use the IFileDocument.FindAllRegExp2 instead.

\section*{FindAllRegExp2}
```

TypeScript
FindAllRegExp2(RegExp: string,
CodePage: number,
SubExpression: number,
Flags: RegExpFlags,
StartFrom: number,
Size: number
Found: IMultiSelection): void;

```
```

C\#
uint CodePage
uint CodePage,
uint SubExpression,
RegExpFlags Flags,
ulong StartFrom,
ulong Size,
IMultiSelection Found)

```
C++ HRESULT FindAllRegExp2(_bstr_t RegExp
    unsigned long CodePage
    unsigned long CodePage,
unsigned long SubExpression,
    RegExpFlags Flags,
    Regexprlags
unsigned long long
lontart
    unsigned long long Size,
    IMultiSelectionPtr Found);

\section*{Parameters}

\section*{RegExp}

A string that contains a regular expression. A regular expression must by in ECMAScript syntax.

\section*{CodePage}

Code page for regular expression pattern encoding or CP_UNICODE (-5) for UTF-16.

\section*{SubExpressio}

The number of sub-match to search for. 0 means the entire expression.

\section*{Flags}

Flags that change the behavior of the function. May be one or more values from the RegExpFlags enumeration

\section*{StartFrom}

A start offset of the range. Must be a multiple of 2 if RegExpTypeUNICODE flag is specified.
Size
Range's size. Must be a multiply of 2 if RegExpTypeUNICODE flag is specified
Found
Multiple Selection object, which contains, on method's return, all located ranges.

\section*{Return Value}

A number of occurrences.

\section*{Description}

Locates all matches of a given regular expression in a given range. You provide the function with an output Multiple Selection Object Found, in which it stores all located ranges. This object is automatically cleared before the operation begins. When the method returns, check the IMultiSelection. Empty property to see if there were any pattern occurrences.

Complexity: depends on the range's size and regular expression complexity.

Replace

TypeScrip
// This method is not available in scripting environment
void Replace(ulong Offset,
uint SizeFrom,
ref byte To,
uint SizeTo)
```

+++
unsigned long SizeFrom,
unsigned char * To,
unsigned long SizeTo)

```

\section*{Parameters}

\section*{offset}

Replace offset.
SizeFrom
Size of the original pattern
то
Pointer to a replace pattern.
sizeTo
Size of the replace pattern

\section*{Description}

Replaces the given block with a pattern. This method replaces a range described by offset and SizeFrom parameters with a pattern.
Complexity: constant-time

\section*{Example}

Replacing a pattern
```

C\#
var fdoc = new FileDocumentLib.FileDocument();
var msec = fdoc.CreateEmptySelection();
msec.AddRange(0, fdoc.FileSize); // We'll be searching in a whole file
byte[] pattern1 = new byte[] { 0xөd, 0xөa }; // EOL pattern
byte[] pattern2 = new byte[] { 0x\otimesd, 0xөa, 0xөd, 0xөa }; // Double-EOL pattern
ulong EOL_Offset = fdoc.Find(ref pattern1[0], pattern1.Length, msel, 0, false, false);
!= -1
fdoc.Replace(EOL_Offset, pattern1.Length, ref pattern2[0], pattern2.Length);

```

\section*{ReplaceAII}

\section*{TypeScrip}
// This method is not available in scripting environment

C\#
id ReplaceAll(ref byte From,
uint SizeFrom,
uint SizeTo,
bool IgnoreCase
IMultiSelection Selection);

C++
RESULT ReplaceAll(unsigned char * From
unsigned long SizeFrom
unsigned char * To,
VARIANT BOOL IgnoreCas
IMultiSelectionPtr Selection)

\section*{Parameter}

From
Pointer to a search pattern

\section*{SizeFron}

Search pattern size
To
Pointer to a replace pattern.
SizeTo
Replace pattern size.

\section*{IgnoreCase}
true to ignore case, false to perform exact matching

\section*{selection}

Multiple Selection Object, describing the ranges in which to perform search and replace.

\section*{Return Value}

A number of replacements done.

\section*{Description}

Finds and replaces all occurrences of the given pattern with another pattern
Complexity: varying.

\section*{Example}

Searching all pattern occurrences

C\#
ar fdoc \(=\) new FileDocumentLib. FileDocument();
var msel \(=\) fdoc.CreateEmptySelection();
doc.Open(@"c:\temp\file.txt")
.AddRange( 0 , fdoc.FileSize); // We'll be searching in a whole file
byte[] pattern1 = new byte[] \{ 0xөd, 0x0a \}; // EOL pattern
byte[] pattern2 = new byte[] \{ 0x0d, 0x0a, 0x0d, 0x0a \}; // Double-EOL pattern
long nReplacements \(=\) fdoc.ReplaceAll(ref pattern1[0], pattern1.Length, ref pattern2[0], pattern2.Length, false, msel);

\section*{ReplaceAliregExp}

\section*{Typescrip}
// This method is not available in scripting environment
c\#
id ReplaceAllRegExp(string RegExp
uint Subexpression,
RegExpflags
ref byte To,
ref byte to,
uint SizeTo,
ulong StartFrom,
ulong Size);
```

HRESULT ReplaceAllRegExp(_bstr_t RegExp,
unsigned long SubExpression
RegExpFlags Flags
unsigned char To,
unsigned long SizeTo,
unsigned long long StartFrom,
unsigned long long Size)

```

\section*{Parameters}

RegExp
A string that contains a regular expression. A regular expression must by in ECMAScript syntax

\section*{SubExpressio}

The number of sub-match to search for. 0 means the entire expression.
Flags
Flags that change the behavior of the function. May be one or more of the values from RegExpFlags enumeration
To
Pointer to a replace pattern.
SizeTo
Replace pattern size.

\section*{Startfrom}

A start offset of the range. Must be a multiple of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.
Size
Range's size. Must be a multiply of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified

\section*{Return Value}

A number of occurrences

\section*{Description}

Finds and replaces all occurrences of regular expression matches with a given pattern
Complexity: varying
This method is obsolete. Use the IFileDocument.ReplaceAllRegExp2 instead.

\section*{ReplaceAllRegExp2}

\section*{ypeScrip}

This method is not available in scripting environment
```

void ReplaceAllRegExp2(string RegExp,
uint CodePage,
RegExpFlags Flags,
ref byte To,
uint SizeTo,
ulong StartFrom,
ulong Size);
C++
ESULT ReplaceAllRegExp2(_bstr_t RegExp
nsigned long CodePage
signed long SubExpression,
egExpFlags Flags,
unsigned long SizeTo,
unsigned long long StartFrom,
unsigned long long Size)

```

\section*{Parameters}

\section*{RegExp}

A string that contains a regular expression. A regular expression must by in ECMAScript syntax
CodePage
Code page for regular expression pattern encoding or CP_UNICODE (-5) for UTF-16.

\section*{SubExpressio}

The number of sub-match to search for. 0 means the entire expression.
Flags
Flags that change the behavior of the function. May be one or more of the values from the RegExpFlags enumeration.
To
Pointer to a replace pattern.
SizeTo
Replace pattern size.

\section*{StartErom}

A start offset of the range. Must be a multiple of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.
Size
Range's size. Must be a multiply of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified

\section*{Return Value}

A number of occurrences.

\section*{Description}

Finds and replaces all occurrences of regular expression matches with a given pattern.
Complexity: varying.

ReplaceAllRegExpWithRegExp
```

ypeScript
ReplaceAllRegExpWithRegExp(RegExp: string
ReplacePattern: string,
odePage: number
Fags: RegExpFlags,
ize: number): voi
C\#
id ReplaceAllRegExpWithRegExp(string RegExp,
string ReplacePattern,
uint CodePage,
RegExpFlags Flags,
log Size);
C++
HRESULT ReplaceAllRegExpWithRegExp(_bstr_t RegExp
bstr_t ReplacePattern,
unsigned long CodePage
RegExpFlags Flags,
inged long long StartFrom,
unsigned long long Size);

```

Parameters

RegExp
A string that contains a regular expression. A regular expression must by in ECMAScript syntax

\section*{ReplacePattern}

A string that contains a replace regular expression.

\section*{CodePag}

Code page for both regular expression patterns encoding or CP_UNICODE (-5) for UTF-16.
Flags
Flags that change the behavior of the function. May be one or more of the values from RegExpFlags enumeration

\section*{StartFrom}

A start offset of the range. Must be a multiple of 2 if UTF-16 encoding is used.
Size
Range's size. Must be a multiply of 2 if UTF-16 encoding is used

\section*{Return Value}

A number of occurrences.

\section*{Description}

Finds and replaces all occurrences of regular expression matches with a given regular expression pattern.
Complexity: varying.

ArithmeticOp

TypeScript
ArithmeticOp(Selection: IMultiSelection
Type: ArithmeticOpType,
OpSize: OperandSizeType
Operand?: any): void;
```

C\#
ArithmeticOpType Type,
perandSizeType OpSize
object Operand);
C++
RESULT ArithmeticOp(IMultiSelectionPtr Selection,
ArithmeticOpType Type,
OperandSizeType OpSiz
_variant_t Operand);

```

\section*{Parameters}

\section*{Selection}

The multiple selection object which contains ranges for this operation to work on.
Type
The operation to perform. Can be one of the values from the ArithmeticOpType enumeration.
OpSize
Specifies the operand size. Can be one of the values from the OperandSizeType enumeration.
Operand

Second operand. Ignored for ArithmeticOpNegation operation type.

\section*{Description}

Performs an arithmetic operation on a given selection.
Complexity: linear-time, depending on selection's complexity

\section*{Bitwise \(_{p}\)}
```

TypeScript
TypeScript This method is not available in scripting environment

```
C\#

C++
HRESULT BitwiseOp(IMultiSelectionPtr Selection, BitwiseOpType Type, unsigned char * Data, unsigned long DataSize);
Parameters

\section*{Selection}

The multiple selection object which contains ranges for this operation to work on.
Type
The operation to perform. Can be one of the values from the BitwiseOpType enumeration
Data
Pointer to an operand pattern.
DataSize
Operand pattern size in bytes. Must be zero for BitwiseOpNOT operation type.

\section*{Description}

Performs a bitwise operation on a given selection.
Complexity: linear-time, depending on selection's complexity

Caseop

\section*{ypeScript}

CaseOp(Selection: IMultiSelection,
Type: CaseOpType,
CodePage?: number): void;

C\#
void CaseOp(IMultiSelection Selection
CaseOpType Type,
bool Unicode,
object CodePage);

HRESULT CaseOp(IMultiSelectionPtr Selection,
CaseOpType Type,
CaseOpType Type,
VARIANT BOOL Unicode,
_variant_t CodePage);

\section*{Parameters}

\section*{Selection}

The multiple selection object which contains ranges for this operation to work on.
Type
The operation to perform. Can be one of the values from CaseOpType enumeration
Unicode
true to treat selection data as UNICODE, false otherwise. If this parameter is true, all blocks in a given selection must have alignment of 2 bytes
CodePage
Optional code page that describes the encoding used in the selection. Ignored if Unicode parameter is True.

\section*{Description}

Performs a case change operation on a given selection.
Complexity: linear-time, depending on selection's complexity

Shiftop
```

ypeScript
ShiftOp(Selection: IMultiSelection, Type: ShiftOpType, OpSize: , Bits: number): void;
\#
void Shiftop(IMultiSelection Selection, ShiftOpType Type, OpSize, uint Bits)

```

HRESULT ShiftOp(IMultiSelectionPtr Selection, ShiftOpType Type, OpSize, unsigned long Bits);

\section*{Parameters}

Selection
The multiple selection object which contains ranges for this operation to work on.
Type
The operation to perform. Can be one of the values from ShiftOpType enumeration
OpSize
Specifies the operand size. Can be one of the values from OperandSizeType enumeration
Bits
The number of bits to shift

\section*{Description}

Performs a shift operation on a given selection.
Complexity: linear-time, depending on selection's complexity

\section*{ReverseOp}

TypeScript
TypeScript
ReverseOp(Selection: IMultiSelection, Type: ReverseOpType, OpSize: OperandSizeType): void;
void ReverseOp(IMultiSelection Selection, ReverseOpType Type, OperandSizeType OpSize);

C++
HRESULT ReverseOp(IMultiSelectionPtr Selection, ReverseOpType Type, OperandSizeType OpSize);

\section*{Parameters}

Selection
The multiple selection object which contains ranges for this operation to work on.
Type
The operation to perform. Can be one of the values from ReverseOpType enumeration
OpSize
Specifies the operand size. Can be one of the values from OperandSizeType enumeration

\section*{Description}

Performs a bit or byte reverse operation on a given selection.
Complexity: linear-time, depending on selection's complexity

\section*{GatherStatistics}

TypeScript
// This method is not available in scripting environment

C\#
ulong[] GatherStatistics(IMultiSelection Selection);

C++

\section*{Parameters}

Selection
Multiple Selection object that describing ranges, in which you want to calculate statistics.

\section*{Return Value}

Statistics array.

\section*{Description}

Calculates file statistics. See the General Statistics section for more information. On method return, each of 256 elements in an array contains the number of corresponding byte occurrences.

Complexity: linear-time, depending on the file's size

\section*{Example}

Calculating General Statistics
// pFileDocument and pMultiselection are initialized elsewhere
unsigned long long data[256];
pFileDocument->GatherStatistics(pMultiSelection, data);

\section*{PatternStatistic}

TypeScrip
/ This method is not available in scripting environment
```

C\# void PatternStatistics(ref byte Pattern
uint DataSize,
uint DataSize,
ref ulong DataBuffer,
uint Blocks,
IMultiSelection Selection)

```
C+
    SULT PatternStatistics(unsigned char * Pattern
    unsigned long DataSize,
VARIANT BOOL
    unsigned long long * DataBuffer
    unsigned long Blocks,
    IMultiSelectionPtr Selection);

Parameters

\section*{Pattern}

Pointer to a pattern.

\section*{DataSize}

Pattern Size

\section*{IgnoreCase}
true to ignore case, false to perform exact matching

\section*{DataBuffer}

Pointer to the statistics buffer.
Blocks
Number of blocks.
Selection
Multiple Selection Object, containing ranges in which you want to calculate statistics

\section*{Description}

Calculates file statistics. See the Pattern Statistics section for more information. On method return, the DataBuffer buffer is filled with calculated statistics values.
Complexity: linear-time, depending on the file's size.

\section*{Example}

Calculating Pattern Statistics
C++
// pFileDocument and pMultiSelection are initialized elsewhere
unsigned int64 data[1024];
nsigned char pattern[] \(=\{\) '0x0d, exөa \};
pFileDocument->PatternStatistics(pattern, sizeof(pattern), false, data, 1024, pMultiSelection);
```

ypescript
// This method is not available in scripting environment

```

C\#
oid PatternStatisticsRegExp(string RegExp
uint SubExpression,
RegExpFlags Flags,
uint Blocks,
uint Blocks,
ulong Size);

C++
RESULT PatternStatisticsRegExp(_bstr_t RegExp
unsigned long SubExpression,
RegExpFlags Flags,
unsigned long long * DataBuffer
unsigned long Blocks,
unsigned long long Startfrom,
unsigned long long Size);

\section*{Parameter}

RegExp
A string that contains a regular expression. A regular expression must by in ECMAScript syntax

\section*{SubExpression}

The number of sub-match to search for. 0 means the entire expression.
Flags
Flags that change the behavior of the function. May be one or more of the values from the RegExpFlags enumeration.

\section*{DataBuffer}

Pointer to the statistics buffer.

\section*{Blocks}

Number of blocks.

\section*{StartFrom}

A start offset of the range. Must be a multiple of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.
Size
Range's size. Must be a multiply of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.

\section*{Description}

Calculates file statistics for a given regular expression. See the Pattern Statistics section for more information. On method return, the DataBuffer buffer is filled with calculated statistics values.

Complexity: linear-time, depending on the file's size and regular expression complexity
This method is obsolete. Use the IFileDocument.PatternStatisticsRegExp2 instead

\section*{PatternStatisticsRegExp2}

\section*{ypeScrip}
// This method is not available in scripting environment

C\#
uint CodePage,
uint SubExpression,
reg ulong DataBuffer,
uint Blocks,
ulong StartFrom,
ulong Size);
C++
RESULT PatternStatisticsRegExp2(_bstr_t RegExp,
unsigned long CodePage
nsigned long SubExpression,
unsigned long long
unsigned long Blocks,
unsigned long long StartFrom
unsigned long long Size);

\section*{Parameters}

RegExp
A string that contains a regular expression. A regular expression must by in ECMAScript syntax.

\section*{CodePage}

Code page for regular expression pattern encoding or CP_UNICODE (-5) for UTF-16.

\section*{SubExpressio}

The number of sub-match to search for. 0 means the entire expression.
Flags
Flags that change the behavior of the function. May be one or more of the values from RegExpFlags enumeration

\section*{DataBuffer}

Pointer to the statistics buffer

\section*{Blocks}

Number of blocks.

\section*{StartFron}

A start offset of the range. Must be a multiple of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.
Size
Range's size. Must be a multiply of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified

\section*{Description}

Calculates file statistics for a given regular expression. See the Pattern Statistics section for more information. On method return, the DataBuffer buffer is filled with calculated statistics values.

Complexity: linear-time, depending on the file's size and regular expression complexity.

\section*{CreateEmptySelection}

\section*{ypeScrip}

CreateEmptySelection(): IMultiSelection;

C\#
IMultiSelection CreateEmptySelection();
\(\xrightarrow{\text { C++ }}\) IMultiSelectionPtr CreateEmptySelection();

\section*{Return Value}

Returns the newly created Multiple Selection Object.

\section*{Description}

Sel Pointer to a variable that receives a pointer to the IMultiSelection interface of created multiple selection object.
Creates an empty selection object. Complexity: constant-time

\section*{Example}

Creating an empty selection object
```

C++
/ pFileDocument is obtained lsewh
MultiSelection *pMultiSelection {};
f (SUCCEEDED(pFileDocument->CreateEmptySelection(\&pMultiSelection)))
// work with a selection
pMultiSelection->Release(); // release object
}

```
CreateSequence
ypeScript
CreateSequence(): ISequence
C\#
ISequence CreateSequence();
C++
ISequencePtr CreateSequence();

\section*{Description}

Create an empty Sequence Object.

BitwiseOps
\(\frac{\text { TypeScript }}{\text { BitwiseOpS(Selection: IMultiSelection, Type: BitwiseOpType, Sequence: ISequence): void }}\)

C\# // This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

Selection
The multiple selection object which contains ranges for this operation to work on.
Type
The operation to perform. Can be one of the values from BitwiseOpType enumeration

\section*{Sequence}

Sequence object containing operand

\section*{Description}

Performs a bitwise operation on a given selection.
Complexity: linear-time, depending on selection's complexity
This method is a replacement for IFileDocument.BitwiseOp when used in scripting languages.

FillMultis

TypeScript
FillMultis(Sequence: ISequence, Selection: IMultiSelection, Continue: boolean): void;

C\#
// This method is not available in managed environment

C++
/ This method is not available in native environment

\section*{Parameters}

Sequence
Sequence object containing the data to fill.

\section*{Selection}

The multiple selection object, which contains the ranges to fill in the document.

\section*{Continu}

True to continue filling in a new range and False to start from the beginning of the pattern. See the description of the "Transparent fill" flag in the Fill user-interface command for more information.

\section*{Description}

Fills a given multiple selection with a specified pattern. The pattern is repeated until all given range(s) are filled. Continue parameter specifies whether to continue filling a new range from the previous range, or from the beginning of the pattern.

Complexity: linear-time, depending on the selection's complexity.
This method is a replacement for IFileDocument.FillMulti when used in scripting languages
Example
Filling a range with a single byte

JavaScript
var msel = fdoc.CreateEmptySelection();
msel.AddRange(100, 20);
msel.AddRange (200, 30);
var pattern = fdoc.CreateSequence()
pattern.AddData(HexBytes, 0x31, 0x32, 0x33);
fdoc.FillMultis(pattern, msel, false);

Fills

TypeScript
FillS(Sequence: ISequence, offset: number, Size: number): void;
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{Sequence}

Sequence containing data to fill
offset
Range starting offset
Size
Range size

\section*{Description}

Fills a given range with a given pattern. A pattern is repeated until the required number of bytes is written to the file. If file is shorter than the filling range, the file's size is increased.

Complexity: constant-time
This method is a replacement for IFileDocument.Fill when used in scripting languages.

\section*{Example}

Filling a range with a simple pattern
```

JavaScript
//foc is initialized elsewhere
var pattern = fdoc.CreateSequence();
pattern.AddText("Simple Pattern", false);
fdoc.FillS(pattern, 0x10000, 0x10ff);

```

FindAlls

\section*{ypeScrip}
indAllS(Sequence: ISequence,
Selection: IMultiSelection
IgnoreCase: boolean,
Found: IMultiSelection): void;

C\# This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{Sequence}

Sequence object containing a find pattern.

\section*{Selection}

Multiple Selection object, which contains ranges in which a pattern need to be located

\section*{IgnoreCase}
true to ignore case, false to perform exact matching
Found
Multiple Selection Object, which contains, on method's return, all located ranges.

\section*{Description}

Locates all occurrences of a given pattern within a given multiple selection. You provide the function with an output Multiple Selection Object Found, in which it stores all located ranges. This object is automatically cleared before the operation begins. When the method returns, check the IMultiSelection.Empty property to see if there were any pattern occurrences.

Complexity: linear-time, depending on the file's size and selection's complexity

This method is a replacement for IFileDocument.FindAll when used in scripting languages.

\section*{Example}

Searching all pattern occurrences
```

JavaScript
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msel = fdoc.CreateEmptySelection();
var mselOutput = fdoc.CreateEmptySelection();
doc.Open("c:<br>temp<br>file.txt");
msel.AddRange(0, fdoc.FileSize); // We'll be searching in a whole file
var pattern = fdoc.CreateSequence();
attern.AddData(HexBytes, 0xӨd, 0xӨa); // EOL pattern
doc.FindAllS(pattern, msel, false, mselOutput);
fdoc Copy(mselOutput, false); /| copy them to the Clipboard

```
Finds
TypeScript
indS(Sequence: ISequence
    Selection: IMultiSelection,
    StartFrom: number
    Searchup: boolean,
C\#
// This method is not available in managed environment
C++
// This method is not available in native environment

\section*{Parameters}

\section*{Sequenc}

Sequence object containing a find pattern.

\section*{Selection}

Multiple selection object, which contains ranges in which you want to locate a pattern

\section*{StartFrom}

Offset from which you want to start searching
Searchup
true to search backwards and false to search forward.

\section*{gnoreCase}
true to ignore case and false to perform exact matching

\section*{Return Value}

Offset of the located pattern or - 1 if pattern was not found

\section*{Description}

Searches for a pattern within a given selection. This method returns the offset of the matched pattern. To continue searching, call this method again, adjusting StartFrom field appropriately. If pattern is not found in the specified range, -1 is returned

Complexity: linear-time, depending on the file's size and selection's complexity

This method is a replacement for IFileDocument.Find when used in scripting languages.

\section*{Example}

Searching for a pattern
JavaScript
ar fdoc = new ActiveXObject("FileDocument.FileDocument")
var \(\mathrm{msec}=\) fdoc.CreateEmptySelection();
fdoc.Open("c:\\temp\\file.txt");
msec.AddRange(0, fdoc.FileSize); // We'll be searching in a whole file
var pattern = fdoc.CreateSequence();
var EOL Offset = fdoc.ToNumber(fdoc.Finds(pattern, msel, 0, false, false))
// ...
EOL_Offset = fdoc.ToNumber(fdoc.FindS(pattern, msel, EOL_Offset + 1, false, false));

InsertPatternS
ypeScrip
InsertPatternS(Sequence: ISequence, Offset: number, Size: number): void;
// This method is not available in managed environment

C++ This method is not available in native environment

\section*{Parameters}

\section*{sequence}

Sequence object containing data to insert.
offset
Insert offset

Size
Insert size.

\section*{Description}

Inserts a given pattern into the document, repeating it to fill the given size. Insert command shifts file's data to free up the space to fit an inserted pattern
Complexity: constant-time
This method is a replacement for IFileDocument.InsertPattern when used in scripting languages

\section*{Example}

Inserting a pattern
```

JavaScript
/ fdoc is initialized elsewhere
var pattern = fdoc.CreateSequence(),
attern.AddData(HexBytes, 0xӨd, 0xӨa)
fdoc.InsertPatternS(pattern, 100, 1000)

```

Inserts

TypeScript
InsertS(Sequence: ISequence, Offset: number): void;

C\#
C\# This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{Sequence}

Sequence object containing the data to insert.
offset
Insert offset

\section*{Description}

Inserts a pattern into the document. Insert command shifts file's data to free up the space to fit an inserted pattern.
Complexity: constant-time
This method is a replacement for IFileDocument.Insert when used in scripting languages

\section*{Example}

Inserting a pattern

JavaScript
// fdoc is initialized elsewhere
var pattern = fac.Createsequence();
pattern.AddData(HexBytes, 0x0d, 0x日a)
fdoc.InsertS(pattern, 100); // insert a given pattern at the offset 100

\section*{Reads}

TypeScript
ReadS(Sequence: ISequence, Offset: number): void;
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{Sequence}

Sequence object into which data is read
offset
Read offset

\section*{Description}

Reads data from a document. If requested data exceeds the current document's size, any excess bytes are filled with zeros.
Complexity: constant-time
This method is a replacement for IFileDocument.Read when used in scripting languages.

\section*{Example}

Reading Data
```

JavaScript
// fdoc is initialized elsewhere
var seq = fdoc.CreateSequence();
*).Length = 256;

```

\section*{ReplaceAllRegExpS}

\section*{TypeScript}

ReplaceAllRegExpS(RegExp: string
SubExpression: number,
Flags: RegExpFlags,
sequence: ISequence
Size: number) void

C\#
// This method is not available in managed environment

C+
// This method is not available in native environment

\section*{Parameters}

RegExp
A string that contains a regular expression. A regular expression must by in ECMAScript syntax.

\section*{SubExpression}

The number of sub-match to search for. 0 means the entire expression.

\section*{Flags}

Flags that change the behavior of the function. May be one or more of the values from the RegExpFlags enumeration.

\section*{sequence}

Sequence object containing replace pattern

\section*{StartFrom}

A start offset of the range. Must be a multiple of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.

\section*{size}

Range's size. Must be a multiply of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified

\section*{Return Value}

A number of occurrences.

\section*{Description}

Finds and replaces all occurrences of regular expression matches with a given pattern.
Complexity: varying.
This method is a replacement for IFileDocument.ReplaceAllRegExp when used in scripting languages. This method is obsolete. Use the IFileDocument.ReplaceAllRegExp2S instead.

\section*{ReplaceAllRegExp2S}

TypeScrip
ReplaceAllRegExp2S(RegExp: string
CodePage: number,
SubExpression: number
Flags: RegExpFlags,
Sequence: ISequence
StartFrom: number,
Size: number): void
C\#
// This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}

\section*{RegEx}

A string that contains a regular expression. A regular expression must by in ECMAScript syntax
CodePage
Code page for regular expression pattern encoding or CP_UNICODE (-5) for UTF-16

\section*{SubExpressio}

The number of sub-match to search for. 0 means the entire expression.

\section*{Flags}

Flags that change the behavior of the function. May be one or more of the values from the RegExpFlags enumeration.

\section*{sequence}

Sequence object containing replace pattern

\section*{StartFrom}

A start offset of the range. Must be a multiple of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.
Size

Range's size. Must be a multiply of 2 if RegExpFlags.RegExpTypeUNICODE flag is specified.

\section*{Return Value}

A number of occurrences.

\section*{Description}

Finds and replaces all occurrences of regular expression matches with a given pattern
Complexity: varying.
This method is a replacement for IFileDocument.ReplaceAllRegExp2 when used in scripting languages.

\section*{ReplaceAlls}
```

TypeScript
ReplaceAllS(SequenceFrom: ISequence,
sequenceTo: ISequence,
gnoreCase: boolean

```
    Selection: IMultiSelection): void

C\#
// This method is not available in managed environment
// This method is not available in native environment

\section*{Parameters}

\section*{SequenceFrom}

Sequence object containing a find pattern.
SequenceTo
Sequence object containing a replace pattern.

\section*{IgnoreCase}
true to ignore case, false to perform exact matching

\section*{Selection}

Multiple Selection object, describing the ranges in which to perform search and replace

\section*{Return Value}

A number of replacements done.

\section*{Description}

Finds and replaces all occurrences of the given pattern with another pattern
Complexity: varying.
This method is a replacement for IFileDocument.ReplaceAll when used in scripting languages

\section*{Example}

Searching all pattern occurrences
JavaScrip
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var fdoc = new ActiveXObject( FileDocum
doc.Open("c:\\temp\\file.txt");
msel.AddRange(0, fdoc.FileSize); // We'll be searching in a whole file
var pattern1 = fooc.CreateSequence();
ar pattern2 = fdoc.CreateSequence();


var nReplacements = fdoc.ToNumber(fdoc.ReplaceAlls(pattern1, pattern2, false, msel);

\section*{Replaces}

ReplaceS(Offset: number, Size: number, Sequence: ISequence): void;

C\#
// This method is not available in managed environment

C++
This method is not available in native environment

\section*{Parameters}

\section*{Offse}

Replace offset
Size
Size of the original pattern.

\section*{Sequenc}

Sequence object containing the pattern to replace.

\section*{Description}

Replaces the given block with a pattern. This method replaces a range described by offset and Size parameters with a pattern
Complexity: constant-time
This method is a replacement for IFileDocument.Replace when used in scripting languages

\section*{Example}

Replacing a pattern

\section*{JavaScript}
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msec \(=\) fdoc.CreateEmptySelection();
fdoc.Open("c:\\temp\\file.txt");
mec.AddRange(0, fdoc.FileSize); // We'll be searching in a whole file
var pattern1 \(=\) fdoc.CreateSequence();
var pattern2 \(=\) fdoc.CreateSequence();
(HexBytes, 0x0d,0x日a), // E0L pattern
pattern2.AddDate(HexBytes,0x0d,0x0a,0x0d,0x0a); // Double-EOL pattern
ar
(EOL_Offset !=-1)
fdoc.ReplaceS(EOL_Offset, pattern1.Length, pattern2);

ToNumber
ypeScrip
ToNumber(Value: any): number;
// This method is not available in managed environment
c++
This method is not available in native environment

\section*{Parameters}

Value
A value to convert

\section*{Return Value}

Value, converted to decimal type.

\section*{Description}

Converts 64-bit integer into scripting compatible integer. Hex Editor Neo extensively uses 64 -bit integer numbers. Scripting languages do not directly support such values, so this method may be used to convert them to supported decimal type.

\section*{Example}

Value conversion

\section*{JavaScript}
// document1 and document2 are defined elsewhere
document1.FileSize \(=\) document2.FileSize; // valid, no processing in Javascript.
// Passing 64-bit integers is supported
document1.FileSize = document2.FileSize / 2; // invalid, "Not a Number" exception is thrown
document1.FileSize \(=\) document2.ToNumber(document2.FileSize) / 2; // valid

Writes

TypeScript
WriteS(Sequence: ISequence, Offset: number): void;

C\#
I/ This method is not available in managed environment
// This method is not available in native environment

\section*{Parameter}

\section*{Sequence}

Sequence object that contains the data to write.

\section*{offset}

Write offset

\section*{Description}

Writes data to a document. The written data overwrites document's existing data and may also increase the file's size if it overlaps the current file's size. Write always creates a new operation in document's operation history

Complexity: constant-time
This method is a replacement for IFileDocument.Write when used in scripting languages.

\section*{Example}

Writing Data
```

JavaScript
// fdoc is initialized elsewhere
lol
fdoc.WriteS(seq, 1000);

```

PatchOptions Enumeration
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline PatchNoOptions & 0x000000000 & No options specified. \\
\hline PatchSaveFileName & 0x00000001 & Save file's name for later verification. \\
\hline PatchCalculateHash & 0x000000002 & Calculate and save file's hash. \\
\hline PatchRemoveReferences & 0x000000004 & Remove external references (currently not implemented). \\
\hline PatchExecutable & 0x00000008 & Create a self-installing patch. PathToStubFile parameter in IFileDocument.CreatePatch method is \\
required if this option is used.
\end{tabular}

\section*{HistoryPurgeLevel Enumeration}
\begin{tabular}{llll} 
Symbol & \multicolumn{1}{l}{ Value Description } \\
\hline HistoryPurgeTail & 0 & All phantom operations (or operation tail) are dropped and removed. \\
\hline HistoryPurgeBranches & 1 & All branches except the current one are dropped and removed. \\
\hline HistoryPurgeAllButCurrent & 2 & \begin{tabular}{l} 
All operations in a history are deleted. The document state is NOT changed, all modifications are \\
"merged" into the single operation, which becomes an operation list root.
\end{tabular} \\
\hline HistoryPurgeAll & 3 & Remove all operations except the very first one. \\
\hline
\end{tabular}

\section*{TextType Enumeration}
\begin{tabular}{lll}
\hline Symbol & Value & Description \\
\hline TextTypeHexByte & 0 & Hexadecimal numbers (from 0 to 0xFF) \\
\hline TextTypeHexWord & 1 & Hexadecimal numbers (from 0 to 0xFFFF) \\
\hline TextTypeHexDword & 2 & Hexadecimal numbers (from 0 to 0xFFFF FFFF) \\
\hline TextTypeHexQword & 3 & Hexadecimal numbers (from 0 to 0xFFFF FFFF FFFF FFFF) \\
\hline TextTypeDecByte & 4 & Decimal numbers (from 0 to 255 ) \\
\hline TextTypeDecWord & 5 & Decimal numbers (from 0 to 65535 ) \\
\hline TextTypeDecDword & 6 & Decimal numbers (from 0 to \(4,294,967,295\) ) \\
\hline TextTypeDecQword & 7 & Decimal numbers (from 0 to \(18,446,744,073,709,551,615\) ) \\
\hline
\end{tabular}

\section*{Paste3Flags Enumeration}
\begin{tabular}{|c|c|c|}
\hline Symbol & Value & Description \\
\hline Paste30verwriteMode & 0x00000000 & Overwrite existing document data with data from the Clipboard. May not be combined with Paste3InsertMode. \\
\hline Paste3InsertMode & 0x00000001 & Insert Clipboard data at the given position, extending document's size. May not be combined with Paste3OverwriteMode. \\
\hline Paste3LittleEndian & 0x00000000 & Treat text in Clipboard as integers that have little-endian byte order. Used only if Clipboard contains textual data. May not be combined with Paste3BigEndian. \\
\hline Paste3BigEndian & 0x00000002 & Treat text in Clipboard as integers that have big-endian byte order. Used only if Clipboard contains textual data. May not be combined with Paste3LittleEndian. \\
\hline
\end{tabular}

RegExpFlags Enumeration
Symbol Value Description

RegExpTypeANSI 0x00000000 Treat the given range as a stream of single-byte characters. May not be used with RegExpTypeUNICODE flag. RegExpTypeUNICODE 0x00000001 Treat the given range as a stream of 16 -bit unicode characters. May not be used with RegExpTypeANSI flag. RegExpIgnoreCase \(0 \times 00000002\) If this flag is specified, ignore case during search operation.

\section*{ArithmeticOpType Enumeration}
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline ArithmeticOpNegation & 0 & Arithmetic negation (change sign): \(x[i]=-x[i] ;\) \\
\hline ArithmeticOpAddition & 1 & Arithmetic addition: \(x[i]+=\) operand; \\
\hline ArithmeticOpSubrtaction & 2 & Arithmetic subtraction: \(x[i]-=\) operand; \\
\hline ArithmeticOpMultiplication & 3 & Arithmetic multiplication: \(x[i] *=\) operand; \\
\hline ArithmeticOpDivision & 4 & Arithmetic division: \(x[i] /=\) operand; \\
\hline ArithmeticOpRemainder & 5 & Arithmetic division remainder: \(x[i] \%=\) operand; \\
\hline
\end{tabular}

\section*{OperandSizeType Enumeration}
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline OperandSizeByte & 0 & 8 -bit integer (BYTE) \\
\hline OperandSizeWord & 1 & 16-bit integer (WORD) \\
\hline OperandSizeDword & 2 & 32 -bit integer (DWORD) \\
\hline OperandSizeQword & 3 & 64 -bit integer (QWORD)
\end{tabular}

\footnotetext{
BitwiseOpType Enumeration
}
\begin{tabular}{|c|c|c|}
\hline Symbol & Value & Description \\
\hline BitwiseOpNOT & 0 & Bitwise NOT: \(x[i]=\sim x[i]\); \\
\hline BitwiseOpOR & 1 & Bitwise OR: \(x\) [ i\(]\) | \(=\) operand[ i \% DataSize]; \\
\hline BitwiseOpAND & 2 & Bitwise AND: \(x\) [ \(i\) ] \& \(=\) operand[ i \% DataSize]; \\
\hline BitwiseOpXOR & 3 & Bitwise XOR: \(\times\) [ i\(]^{\wedge}=\) operand[ i \% DataSize]; \\
\hline
\end{tabular}

\section*{CaseOpType Enumeration}
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline CaseOpLower & 0 & Make selection lowercase \\
\hline CaseOpUpper & 1 & Make selection uppercase \\
\hline CaseOpToggle & 2 & Invert selection \\
\hline
\end{tabular}

\section*{ShiftOpType Enumeration}
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline ShiftOpLogicalLeft & 0 & Logical left shift: \(\times[i] \ll=\) operand; \\
\hline ShiftOpLogicalRight & 1 & Logical right shift: \(\times[i] \gg=\) operand; \\
\hline ShiftOpArithmeticLeft & 2 & Arithmetic left shift (equivalent to ShiftopLogicalLeft): \(\times[i] \ll=\) operand; \\
\hline ShiftOpArithemticRight & 3 & Arithmetic right shift: \(\times[i] \gg=\) operand; // performs sign extension \\
\hline ShiftOpRotateLeft & 4 & Rotate left. \\
\hline ShiftOpRotateRight & 5 & Rotate right.
\end{tabular}

\section*{ReverseOpType Enumeration}
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline ReverseOpBits & 0 & Reverse bits in a selection \\
ReverseOpBytes & 1 & Reverse bytes in a selection \\
\hline
\end{tabular}

\section*{Multiple Selection Object}

\section*{MultiSelection Interface}

\section*{Description}

This interface is implemented by the Multiple Selection Object Its methods and properties are directly accessed by the low-level (native) languages like C++ and indirectly by automation-compatible languages, such as JavaScript and managed languages. Refer to Creating Multiple Selection Object and Working with Multiple Selection Object sections for usage information.

\section*{Declaration}

\section*{TypeScript}
interface IMultiSelection extends IDispatch \{
// Properties
readonly Count: number;
readonly TotalSize: numb
// Methods
AddRange(Offset: number, Size: number): void;
Clear(): void;
DeleteRange(Offset: number, Size: number): void;
InsertRange(Offset: number, Size: number): void;
Invert(Size: number): void;
RemoveRange(Offset: number, size: number): void
RemoveRange(Offset: number, Size: number): void;
ToggleRange(Offset: number, Size: number): void;
\}
public interface IMultiSelection : IDispatch
```

    // Properties
    ulong Count \{ get; \}
    ```
    bool Empty \{ get; \}
ulong TotalSize \(\{\) get; \}
    // Methods
    void AddRange(ulong Offset, ulong Size)
    void Clear();
    void DeleteRange(ulong Offset, ulong Size);
    void Insertrange (ulong Offset, ulong Size);
    void Invert (ulong Size);
    void IsIn(ulong Offset);
    void Load(IStream Stream, LoadMode LoadMode);
    void RemoveRange(ulong Offset, ulong Size);
    oid Save(IStream Stream).
    void ToggleRange(ulong Offset, ulong Size);
\}
```

C++
struct IMultiSelection : IDispatch
// Properties
unsigned long long Count; // get
VARIANT_BOOL Empty; // get
unsigned long long TotalSize; // get
// Methods
HRESULT AddRange(unsigned long long Offset, unsigned long long Size);
HRESULT Clear()
HRESULT DeleteRange(unsigned long long Offset, unsigned long long Size);
HRESULT InsertRange(unsigned long long offset, unsigned long long Size);
HRESULT Invert(unsigned long long Size);
HRESULT IsIn(unsigned long long Offset)
HRESULT Load(IStream * Stream, LoadMode LoadMode);
HRESULT RemoveRange(unsigned long long Offset, unsigned long long Size)
HRESULT Save(IStream * Stream);
};

```

IMultiSelection Properties
coun
\begin{tabular}{l} 
TypeScript \\
readonly Count: number; \\
\hline \begin{tabular}{l} 
C\# \\
ulong Count \(\{\) get; \} \\
\hline C++ \\
unsigned long long Count; // get \\
\hline
\end{tabular} \\
\hline
\end{tabular}

\section*{Description}

Returns the number of ranges in a selection object
Complexity: constant-time
Example
Using the Count property

\section*{JavaScript}
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msel = fdoc.CreateEmptySelection();
lert(msel.Count); // displays "
sel.AddRange(100, 20); // selection: [100..120)
sel.AddRange(150, 20); // selection: [100..120) U [150..17
alert(msel.Count); // displays "2"
msel.AddRange(80, 80); // selection: [80..170)
alert(msel.Count); // displays "1"

Empty

TypeScript
readonly Empty: boolean;

C\#
bool Empty \{ get;

C++
VARIANT BOOL Empty; // get

\section*{Description}

This property is true if the selection object is empty.

\section*{Example}

Using the Empty property
```

JavaScript
var fdoc = new ActiveXObject("FileDocument.FileDocument")
var msel = fdoc.CreateEmptySelection();
lert(msel.Empty); // displ m
sel.AddRange(100, 20); Sl selection"
msel.clear();
alert(msel.Empty); // displays "true"

```

TotalSize
```

TypeScript
readonly TotalSize: number;

```
ulong TotalSize \{ get; \}

C++
unsigned long long TotalSize; // get

\section*{Description}

Returns the total selection size. This is a sum of sizes of all selection ranges.
Complexity: constant-time

\section*{Example}

Using TotalSize property

JavaScript
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msel = fdoc.CreateEmptySelection();
msel.ToggleRange(100, 50); // selection: [100.. 150
alert(msel.TotalSize); // displays "50"
sel.ToggleRange(100, 10); \(/ /\) selection: [110..150
sel.ToggleRange(0, 10); // selection:
sel.ToggleRange( 0,10 ); // selection: [0..10) U [110..150)
msel.ToggleRange(140, 20); // selection: [0..10) U [110..140) U [150..160
alert(msel.TotalSize); // displays "50"
msel.ToggleRange(10, 100); // selection: [0..140) U [150..160
alert(msel.TotalSize); // displays " 150 "

\section*{Mukiselection Methods}

AddRange
ypeScript
AddRange(Offset: number, Size: number): void

C\#
void AddRange(ulong Offset, ulong Size)

C++
HRESULT AddRange(unsigned long long Offset, unsigned long long Size);

\section*{Parameters}

Offset
Starting offse
Size
Range size
Description
Adds a given range to a selection. If the range intersects any existing ranges, they are combined with a given range. A number of ranges in a selection may change after adding a new range.

Complexity: linear-time, depending on the current selection configuration

\section*{Example}

Adding ranges to a selection

C\# \(/\) msel is initialized elsewhere and is currently empty
msel.AddRange(100, 20); // selection: [100..120)
msel.AddRange (150, 20); // selection: [100..120) U [150..170)
msel.AddRange(80, 80); // selection: [80..170)

Clear

TypeScript
Clear(): void;
void Clear();

C++
HRESULT Clear() ;

\section*{Description}

Clears the selection. After this function returns, selection object contains no ranges.
Complexity: constant-time

DeleteRange
```

TypeScript
TypeScript (Offset: number, Size: number): void;
c\#

```
ct+
HRESULT DeleteRange(unsigned long long Offset, unsigned long long Size);

\section*{Parameters}

\section*{Offset}

N/A
Size
N/A
Description
This method is not currently implemented.

InsertRange

TypeScript
InsertRange(Offset: number, Size: number): void;
void InsertRange(ulong Offset, ulong Size);

C++
HRESULT InsertRange(unsigned long long Offset, unsigned long long Size);

\section*{Parameters}

\section*{Offset}

Insert offset.
Size
Insert size.

\section*{Description}

Inserts a range into the selection object. As a result of this operation, existing ranges with offsets larger than offset are shifted forward by Size bytes.
Complexity: linear-time, depending on the number of ranges to shift

\section*{Example}

Using the InsertRange method
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msel = fdoc.CreateEmptySelection();
msel.AddRange(100, 20); // selection: [100..120
msel.AddRange(150, 20); // selection: [100..120) U [150..170)
msel.InsertRange (130, 10); // selection: [100..120) U [160..180
msel.InsertRange ( 0,20 ); // selection: [120..140) U [180..200)

Invert

TypeScript
Invert(Size: number): void;

C\#
void Invert(ulong Size)

HRESULT Invert(unsigned long long Size)

\section*{Parameters}

Size
The total size of an object. Used to properly invert selection.

\section*{Description}

Inverts the current selection. All defined ranges become gaps and all gaps become ranges. This method accepts the total size which is usually the current file's size. Calling Invert method two times with a same size value will leave the selection object intact.

Complexity: constant-time.

\section*{Example}

Using the Invert method

\section*{JavaScript}
var fdoc = new ActiveXObject("FileDocument.FileDocument");
\(\mathrm{var} \mathrm{msel}=\) fdoc.CreateEmptySelection();
msel.AddRange(100, 20); // selection: [100..120)
sel.Invert(200); // selection: [0..100) U [120..200)
msel.Invert(200); // selection: [100..120)

TypeScript
TypeScript
IsIn(Offset: number): void;

C\#
void IsIn(ulong Offset);
++
RESULT IsIn(unsigned long long offset)

\section*{Parameters}
offset
Offset to check.

\section*{Return Value}

A boolean value indicating whether the given offset lies within one of the selection's ranges.

\section*{Description}

Check if the given offset lies within the selection. Returns true if the offset is inside of selection's ranges or false otherwise.
Complexity: constant-time

\section*{Example}

Using the IsIn method
```

JavaScript
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msel = fdoc.CreateEmptySelection();
msel.AddRange(100, 20); // selection: [100..120)
alert(msel.IsIn(0)); // displays "false"
䘖䘖((mel.IsIn(100)); // displays "true"
alert(msel.IsIn(120)); // displays "false"

```
Load
TypeScript
// This method is not available in scripting environment
C\#
void Load(IStream Stream, LoadMode LoadMode);
C++
HRESULT Load(IStream * Stream, LoadMode LoadMode);

\section*{Parameter}

Stream
Pointer to a stream object
LoadMode
Load mode. Can be one of the values from LoadMode enumeration.

\section*{Description}

Loads the selection from a file. This method may be instructed to merge the loaded selection with a current one, using one of supported algorithms. Due to implementation details, the Load method usually tries to read more data from the stream object, than Save method wrote to it. This is not a problem if nothing is written beyond the selection's data in a stream. Otherwise, you should prepend the selection's data in a stream with a size value and adjust the stream pointer appropriately.

Complexity: linear-time, depending on current and loaded selections complexity.

\section*{Example}

Loading and Saving selection

JavaScript
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var msel \(=\) fdoc.CreateEmptySelection();
msel.AddRange(100, 50); // selection: [100..150)
// create a file and obtain a stream object for it
nsel.Save(stream);
msel.Clear(); // selection: empty
msel.AddRange(50, 50); // selection: [50..100)
abtream object for it
msel.Load(stream, SelectionAdd); // selection: [50..150)

RemoveRange
```

ypeScript
RemoveRange(Offset: number, Size: number): void;
C\#
id RemoveRange(ulong Offset, ulong Size)
HRESULT RemoveRange(unsigned long long Offset, unsigned long long Size);

```
```

Offset
Starting offset
Size
Range size
Description

```

Removes a given range from the selection. The specified range is "subtracted" from any intersected selection's ranges. If it does not intersect any range, nothing happens The number of ranges in selection object may change after this method returns.

Complexity: linear-time, depending on the current selection configuration

\section*{Example}

Removing ranges from a selection
C\#
/ msel is initialized elsewhere and is currently empty
sel.AddRange (100, 50); // selection: [100..150)
msel.RemoveRange(0, 10); // selection: [110..150) - nothing changed
msel.RemoveRange(150, 20); // selection: [110..150) - nothing changed
msel.RemoveRange(120, 10); // selection: [110..120) U [130..150)

Save

TypeScrip
// This method is not available in scripting environment

C\#
void Save(IStream Stream);

C+
RRESULT Save(IStream * Stream)

\section*{Parameters}

\section*{Stream}

Pointer to a stream object

\section*{Description}

Compresses and saves the selection object to a given stream
Complexity: linear-time, depending on number of ranges in a selection object.

ToggleRange

TypeScrip
ToggleRange(Offset: number, Size: number): void;
void ToggleRange(ulong Offset, ulong Size);

C++
HRESULT ToggleRange(unsigned long long Offset, unsigned long long Size);

\section*{Parameters}

Offset
Starting offset.
Size
Range size.

\section*{Description}

Current selection ranges are intersected with a specified range. Those portions of the specified range that fall onto gaps are appended to the selection, and those portions that fall to existing ranges, are subtracted from the selection.

Complexity: linear-time, depending on the current selection configuration
Complexity of this method is not worse than that of IMultiSelection.AddRange and IMultiSelection.RemoveRange

\section*{Example}

Using ToggleRange method
C\#
// msel is initialized elsewhere and is currently empty
sel.ToggleRange(100, 50); // selection: [100.. 150
sel.ToggleRange (100, 10); // selection: [110.. 150
mel.ToggleRange(0, 10); // selection: [0..10) U [110..150)
msel.ToggleRange(10, 100); // selection: [0..140) U [150..160)
\begin{tabular}{lll} 
Symbol & Value & Description \\
\hline SelectionNew & 0 & Replaces the current selection with a one loaded from a file. \\
\hline SelectionAdd & 1 & Combines the loaded selection with a current one. \\
\hline SelectionSubtract & 2 & Subtracts the loaded selection from a current one. \\
\hline
\end{tabular}

\section*{Sequence Object}

\section*{Sequence Interface}

\section*{Description}

This interface is implemented by a sequence object created through a File Document object. Use the methods of this interface from a scripting languages to construct a pattern to be passed to various methods of the IFileDocument interface.
```

Declaration
TypeScript
interface ISequence extends IDispatch {
// Properties
readonly Empty: boolean;
Length: number;
[Value: number]: number;
// Methods
AddData(Type: SequenceDataType, Values: number[]): void
AddText(Text:, Unicode: ): void;
Clear(): void;
Copy(): ISequence;
}

```
// This interface is not available in managed environment
\(\xrightarrow{\text { C++ }}\) // This interface is not available in native environment

ISequence Properties

Empty

\section*{TypeScript \\ readonly Empty: boolean}

C\#
/ This property is not available in managed environment
// This property is not available in native environment

\section*{Description}
true if the sequence is empty or false otherwise.

Length

TypeScript
Length: number;

C\#
/ This property is not available in managed environment
// This property is not available in native environment

\section*{Description}

Retrieve or change the current sequence length. Sequence length is expressed in bytes. When you change the sequence length, it is either truncated or extended. If the sequence was extended, it may contain garbage.

Value

\section*{TypeScript}
[Value: number]: number;

C\#
C\# This property is not available in managed environment
C++
// This property is not available in native environment

\section*{Description}

Retrieve the byte from sequence.

\section*{ISequence Methods}

\section*{AdD}

\section*{TypeScript}

AddData(Type: SequenceDataType, Values: number[]): void;

C\# // This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method is not available in native environment }}\)

\section*{Parameters}

Type
Type of the data to add. See SequenceDataType enumeration for more information.
Values
Any number of additional parameters which are added to a sequence. The type parameter describes the way they are interpreted.

\section*{Description}

Add numeric data to a sequence. This method is used to add bytes, words, double words, quad words, floats and doubles to a sequence. Data is added to the end of the sequence.

\section*{Example}

Adding values
var fdoc = new ActiveXObject("FileDocument.FileDocument");
var seq = fdoc.CreateSequence();
eq.AddData(SequenceDataBytes, 0x0d, 0x0a); // add two bytes to a sequence
seq.AddData(SequenceDataDwords | SequenceDataBigEndian,
0x12345678, 0xfedcba98, 0, 70000); // add a number of big-endian double words to a sequence

\section*{AddText}

TypeScript
AddText(Text: , Unicode: ): void

C\# This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

Text
A string to be added.
Unicode
true to treat string characters as UNICODE or false to treat them as ANSI.

\section*{Description}

Add textual data to a sequence. New data is appended to the end of the sequence.

\section*{Example}

Adding text
```

avaScrip
far fdoc = new ActiveXObject("FileDocument.FileDocument");
var seq = fdoc.CreateSequence();

```
seq.AddText("<p>", false)

Clear

TypeScript
Clear(): void;
// This method is not available in managed environment
\(\frac{\text { C++ }}{/ /}\) This method is not available in native environment

\section*{Description}

Clear the sequence
```

TypeScript
Copy(): ISequence;
C\#
This method is not available in managed environment
// This method is not available in native environment

```

\section*{Description}

Create a copy of the sequence.

CreateSubSequence
```

TypeScript
CreateSubSequence(Index: number, Length: number): ISequence;
C\#
This method is not available in managed environment
// This method is not available in native environment

```

\section*{Parameters}

\section*{Index}

Starting index of a sub-sequence.
Length
Length of the sub-sequence.

\section*{Description}

Create a sub-sequence of this sequence.

\section*{ypescript}

Remove(Index: number, Length: number): void;

C\#
// This method is not available in managed environment

C+
// This method is not available in native environment

\section*{Parameters}

Index
Index of the starting byte to remove
Length
The number of bytes to remove

\section*{Description}

Remove a part of the sequence

SequenceDataType Enumeration
\begin{tabular}{llll}
\hline Symbol & Value & Description \\
\hline SequenceDataBytes & 0 & Treat subsequent method parameters as bytes. \\
\hline SequenceDataWords & 1 & Treat subsequent method parameters as words. \\
\hline SequenceDataDwords & 2 & Treat subsequent method parameters as double words. \\
\hline SequenceDataQwords & 3 & Treat subsequent method parameters as quad words. \\
\hline SequenceDataFloats & 4 & Treat subsequent method parameters as floats (single precision floating-point numbers). \\
\hline SequenceDataDoubles & 5 & Treat subsequent method parameters as doubles (double precision floating-point numbers). \\
\hline SequenceDataLittleEndian & 0x00000000 & Treat numbers as having little-endian byte order. This option is on by default. \\
\hline SequenceDataBigEndian & 0x00010000 & Treat numbers as having big-endian byte order. May be combined with other values. Ignored when \\
\hline
\end{tabular}

\section*{Parser Object}

\section*{Parser Interface}

\section*{Description}

This interface is implemented by the intrinsic parser object, provided to scripts running as part of the Structure Viewer structure binding under the name parser. The name may be omitted and the methods below may be called directly.
```

TypeScript
interface IParser
// Methods
abort(message: string): void;
alert(message: string): void;
bind(typeName: string, varName: string, offset: number): void;
eval(Expression: string): boolean | number | string;
add_coloring_scheme(name: string, frontColor: object, backColor: object, outlineColor: object, roundEdges: boolean): void;
}
// This interface is not available in managed environment
// This interface is not available in native environment

```

\section*{Parser Methods}

\section*{TypeScript}
abort(message: string): void;
c\#
/ This method is not available in managed environment

C+
/ This method is not available in native environment

\section*{Parameters}

\section*{message}

A message to be displayed in the Structure Viewer Tool Window.

\section*{Description}

Abort current binding. Behaves like the \$assert directive.

\section*{alert}

\section*{TypeScript \\ alert(message: string): void}

C\#
// This method is not available in managed environment
\(\frac{\text { C++ }}{\text { // This method is not available in native environment }}\)

\section*{Parameter}

\section*{message}

String to display

\section*{Description}

Display informational message in a popup message box. Should be used only for debugging purposes.

\section*{Example}

Expression evaluation
```

JavaScript
function f()
var a = eval("pe_header.e_lfanew.sections[0].Name");
return 0;

```
\}

\section*{bind}

TypeScript
\(\frac{\text { yypeScript }}{\text { bind(typeName: string, varName: string, offset: number): void; }}\)
// This method is not available in managed environment

C++
/ This method is not available in native environment

\section*{Parameters}
typeName
The name of the user-defined type. The type must be fully defined.

\section*{varName}

The name of the variable. Must be unique
offset
Structure starting address (absolute value).

\section*{Description}

Bind a given structure to a given address. Hex Editor Neo performs delayed binding, that is, a new structure is bound only after the current one is successfully finished binding.

\section*{Example}

Binding a structure

JavaScript
function \(f(\)
bind("struct A","a",0);
\}
eval

TypeScript
eval(Expression: string): boolean | number | string;
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}

\section*{Expression}

The expression to evaluate

\section*{Return Value}

Expression evaluation result.

\section*{Description}

Calculate expression value. You may reference global constants, enumerations, field values of already bound types and so on.

\section*{Example}

Expression evaluation
```

JavaScript
function f()
{
var a = eval("pe_header.e_lfanew.sections[0].Name");
alert("First PE file section name is \"" + a + "\"");
return 0;
}

```

\section*{print}

\section*{TypeScript}
print(varName: string, varValue: string): void;

C\#
This method is not available in managed environment

C++ This method is not available in native environment

\section*{Parameters}

\section*{varName}

Name of the variable. Must be unique in the current scope.

\section*{varValue}

Variable's value, as string.

\section*{Description}

Add a virtual field to the output.

\section*{Example}

Adding virtual fields
```

JavaScript
function f(
print("var1",10);
print("var2","Hello, World")

```
\}
ypeScrip
add_coloring_scheme(name: string, frontColor: object, backColor: object, outlineColor: object, roundEdges: boolean): void;

C\#
// This method is not available in managed environment

C++
// This method is not available in native environment

\section*{Parameters}
name
Scheme's name. Must not match any of existing coloring schemes.

\section*{frontColor}

An object that specifies the font color. See Remarks for details.

\section*{backColor}

An object that specifies the background color. See Remarks for details.
outlineColor
An object that specifies the outline color. See Remarks for details.

\section*{roundEdges}
true to round outline edges, false otherwise

\section*{Description}

Create new binding-local coloring scheme. FrontColor, BackColor and OutlineColor must be JavaScript objects with r , g , b and a properties. If any of the property is omitted, its value is defaulted to 0 , except for \(a\), which value is defaulted to 255 . All values must be within [0..255] range.

\section*{Example}

Creating new color scheme
```

JavaScript
unction f()
fun
Create a coloring scheme Red on Blue, no outline (0 alpha = full transparent), no round edge
parser.add_coloring_scheme("my scheme",{ r:255, g:0, b:0, a:255 },
{r:0, g:0, b:255, a:255 }, { a:0 }, false);
}

```

\section*{Customization}

Hex Editor Neo is a highly customizable application. You can customize almost every feature or module. This section describes several basic customization mechanisms.
In addition, other customizable features are described in other sections. This includes:

\section*{Editor Window Customization}

You can customize the placement and layout of an editor window.

\section*{Structure Viewer Settings Page}

Structure Viewer allows you to define a list of directories that are used to search for included files. It also allows you to define an association between a structure scheme and file extension.

In addition, most interface windows, especially those that encapsulate a Pattern Window, allow you to change their size. The size is then saved and restored next time you use this window. Most window fields also store the values you entered into them and provide you with a list of last recently used values.
This section describes General Settings, Toolbar Customization and Keyboard Customization.
Settings Window
The Settings window is invoked using the Tools » Settings... command.


The Settings window contains a list of pages on the left. Currently selected configuration page is displayed on the right. The changes you make usually take effect immediately. You can press the OK button to apply and retain changes, or Cancel button to discard all changes you made and close the Settings window.

Settings window is resizable and memorizes its size the next time you open it.
Search settings field can be used to search for a specific setting. The search term you enter is searched for on every settings page. This filters the list of settings pages.
All settings pages are divided into the following categories:

\section*{Visual}

Contains the Themes Settings Page, which allows you to configure colors for all Hex Editor Neo's UI elements.

\section*{User Interface}

Contains settings pages for Languages, Keyboard and Commands.

\section*{Network \& Support}

Contains Beta Settings Page, Crash Reporting Settings Page and Proxy Settings Page.

\section*{Binary Editor}

Contains the following settings pages, which allow you to set different settings related either to the Hex Editor Neo in general, or to theeditor window behavior specifically:
- Tasks Settings Page
- Regular Expressions Settings Page
- Editor Settings Page
- Data Annotations Settings Page

\section*{Structure Viewer}

Allows you to configure the Structure Viewer and Structure Editor.

\section*{Other}

Contains the General Settings Page with application-wide settings and Settings Manager Settings Page, which allows you to manage the location of settings.

\section*{Scripts}

Scripts may use the displaySettings method to open the Settings Window, optionally focusing on specific settings page.

\section*{Visual}

\section*{Themes Settings}

Themes settings page allows you to customize the Hex Editor Neo's colors.


A given color set can be saved as a theme. In order to save a theme, enter theme's name into the Theme field and press the Save button. Switch themes by choosing the required theme from a list. Press the Delete button to delete a scheme and Reset All button to restore the color theme.

To modify a color, select it from the list and use controls on the bottom to change its value.
Current shows the previous version of the selected color and Selected shows the new version of the color.
Use \(R, G, B\) color controls to change color's RGB or \(H, S\), and \(L\) controls to change color as HSL triplet. Use the \(A\) control to change color's transparency level. Alternatively, you can directly change the color HEX value.

You can also click on the color wheel to select the desired color. Change the \(L\) control to update wheel's luminosity. When \(L\) is zero, the wheel is totally black, when \(L\) is 100 , the wheel is totally white.

When an Automatic is checked for a given color, Hex Editor Neo uses special algorithm to compute the color value. An automatic color is usually algorithmically related to another color, or a system defined color.

The default color scheme consists entirely of "automatic" colors and adapts to current operating system color settings. For Windows 10 and later the default color scheme adapts to a current Accent Color and to the system settings for Dark mode for applications.

Press the Export button to export the current colors into the file and press the Import button to load colors from a file.

\section*{User Interface}

\section*{Language Settings}

This page allows you to choose the Hex Editor Neo's interface language. The default interface language is English.


Installed languages are listed in the first choice box. Select the one you want and press the OK button. Hex Editor Neo will need to be restarted in order to apply the changes.

\section*{Online Languages}

More languages may be available on-line. When this page is displayed, Hex Editor Neo contacts the HHD Software web server and downloads a list of all available interface languages. It then allows you to download and install them. You may also manually refresh the list by clicking the Refresh button.

We can also publish updates to language packs. If Hex Editor Neo detects an update is available for installed language pack, Downloader's notification window is displayed with a prompt to download the update.

\section*{Keyboard Settings}

Keyboard page allows you to customize keyboard shortcuts used by the Hex Editor Neo.


The list of all commands is displayed on the left．Select a command to see the list of shortcuts assigned to it．They are displayed in a Current keys field．Each command may have several shortcuts，but a shortcut may not be shared between two or more commands．

To delete a shortcut，select it in a list and press the Remove button．
To define new shortcut，activate the Press new shortcut key field and then press the key combination．Use the Assign button to assign the combination to a selected command． Press the Keyboard Map button to open a Keyboard Map．

\section*{Keyboard Map}

The Keyboard Map window displays all currently defined keyboard shortcuts．
\begin{tabular}{|c|c|c|c|c|c|}
\hline fig Keyboard Map & & & & & \\
\hline Command & Shortcuts & Description & & & \\
\hline 監Bookmarks 》 Next Bookmark（BookNext） & Ctri＋B & Go to the next bookmark & & & \\
\hline 氖Bookmarks 》 Previous Bookmark（BookPrev） & Ctri＋Shilt + B & Go to the previous bookmark & & & \\
\hline F Bookmarks 》 Toggle Bookmark（BookToggle） & Ctril \(\mathrm{F}^{2}\) & Put or remove a bookmark on the curren & & & \\
\hline \(\Leftrightarrow\) Edit＂Change File Size．．．（EditSetSize） & Ctri＋Alt＋S & Change the size of the file & & & \\
\hline Edit» Copy（Editopopy） & \(\mathrm{Ctrl}+\mathrm{C}, \mathrm{CrIT}+\mathrm{Ins}\) & Copy the selection and put it on the Clip & & & \\
\hline \％\({ }^{\text {dedit } \geqslant \text { Cut（EditCut）}}\) & Ctri＋X，Shilt＋Del & Cut the selection and put it on the Clipb & & & \\
\hline －Edit » Delete（Editclear） & Del & Delete selection & & & \\
\hline －Edit » Edit Cell（EditEditCell） & F2 & Edit the current cell & & & \\
\hline \％Edit＞Fill．．．（EditFill） & Ctri＋Shift + F & Fill selection with byte or pattern & & & \\
\hline QEdit＞Find Next（EditFindNext） & F3 & Find next pattern occurrence & & & \\
\hline 8Edit＞Find Previous（EditFindPrev） & Shitt＋F3 & Find previous pattern occurrence & & & \\
\hline OEdit＞Find．．．（Editfind） & Ctril + & Find pattern or all pattern occurrences & & & \\
\hline 廌Edit》 Go to Offset．．．（EditGoto） & Ctri＋G & Go to the specific offset in file & & & \\
\hline 倌Edit 》 Insert File．．．（EditinsertFile） & Ctri＋Shilt＋L & Insert the contents of the disk file to the & position & ts ．．． & \\
\hline  & Ctrl + Shilt + H & Insert AsCII－encoded binary fie & & & \\
\hline （il）Edit 》 Insert Mode（EditInsertMode） & Ins & Turn the Insert mode on or off & & & \\
\hline ／\({ }^{\text {Edit }}\) 》 Insert．．．（Editinsert） & Ctri＋Shift +1 & Insert byte or pattern & & & \\
\hline \％edit 》 Merge and Copy（EditCopyMerged） & Ctrl＋Shift + c & Copy the selection，merge each block and & result & pbo．．． & \\
\hline \％Edit » Merge and Cut（EditCutMerged） & Ctrl＋Shilt +X & Cut the selection，merge each block and & result on & board & \\
\hline & & Print & Copy & Close & \\
\hline
\end{tabular}

You can change the size of the window，print it or copy its contents to the Clipboard．

\section*{Commands Settings}

Commands page lets you customize the main application toolbars as well as tool windows toolbars．In addition，you can customize the contents of the File » New and File » Open menus．


Select the toolbar from the Toolbar：list．It lists both application toolbars and tool window toolbars．Toolbar＂New＂corresponds to contents of the File＂New menu and toolbar＂Open＂corresponds to the contents of the File » Open menu．

\section*{Creating New Toolbar}

To create a new, empty toolbar press the New... button and enter the toolbar's name. Now you can add new commands to a created toolbar

\section*{Deleting Toolbar}

To delete a toolbar, select it in the list and click the Delete button. You cannot delete any predefined toolbar or tool window toolbar.

\section*{Configuring a Toolbar}
- To remove a command from the toolbar, click on it and drag away from the toolbar. You can also press the Remove > button.
- To move a command to another location, click and drag it. Dragging the command beyond itself adds a separator. You can also use the Move Up and Move Down buttons.
- To add a command to a toolbar, click on the command in the All Commands list and drag it to the toolbar. The All Commands list may be filtered to a specific command category by changing the current category in the Category: field.
- To add a separator, select the command and press the New Group button to insert a separator before it. Dragging command beyond itself also creates a separator. Separators may be dragged as any other commands.
- Press the Clear button to remove all commands from the toolbar

\section*{Other Options}

You can change the size of the toolbar icons with the Button size control. The Hex Editor Neo's unique vector icon technology renders command icons at any size without artifacts. This slider controls the size of the icons on application toolbars only. The size of tool window toolbars is fixed.

\section*{Network \& Support}

\section*{Beta Settings}

The Beta Settings page allows you to browse, download and install specific testing components published by developers.
We sometimes publish new pre-release features, bug fixes or updates as beta versions to prevent them from affecting a lot of customers. Users may also be instructed to try specific beta components if they experience any particular issue.

In addition to publicly available beta components, we may provide you with a specific private component. In this case, press the Advanced button and enter the component ID to download and install it.

\section*{Crash Reporting Settings}

We have worked hard to eliminate all possible bugs in the product. Nevertheless, as in each software product, bugs occur from time to time. The worst of bugs are general protection faults or other types of exceptions that cause the operating system to forcefully close the application, possibly causing user data losses.

Hex Editor Neo includes the built-in crash reporter. If application crashes, a small crash report, calledminidump is generated. You are offered to send a compressed report to HHD Software web site for analysis.

Crash Reporting Configuration
The Crash Reporting settings pages, accessible through the Tools» Settings... menu, allows you to configure the following information:
\begin{tabular}{|c|c|}
\hline 앙 Settings: Crash Reporting & \(\times\) \\
\hline Search settings & Settings \\
\hline \begin{tabular}{l}
Visual \\
Themes \\
User Interface \\
Language \\
Keyboard \\
Commands \\
Network \& Support \\
Beta \\
Crash Reporting \\
Proxy \\
Binary Editor \\
Regular Expressions \\
Tasks \\
Editor \\
Data Annotations \\
Structure Viewer \\
General \\
Structure Editor \\
Other \\
General \\
Settings
\end{tabular} & \begin{tabular}{l}
If you would like to be notified of the status of your crash report submission and want your reports \\
to be processed with higher priority, please provide your e-mail address in the box below. Your \\
e -mail address will not be used for anything else beyond discussion of your submission. \\
user@example.com \\
Send report automatically \\
Report verbosity level \\
- Minidump \\
Minidumps usually have a size less than a megabyte and contain only limited information about a crash. Sometimes a minidump is enough to diagnose the problem, but often it lacks necessary information required to reproduce the crash.
Full dump \\
Full dumps contain the compressed view of all process address space. They are best to allow developers to reproduce the situation that caused the crash and fix it as soon as possible. However, the size of a full dump may easily reach hundreds of megabytes.
\end{tabular} \\
\hline & \begin{tabular}{ll|l} 
OK Close & Help
\end{tabular} \\
\hline
\end{tabular}

You can optionally enter your e-mail address which will be submitted with crash dumps generated on your computer. If you provide your e-mail address, you might be contacted to provide additional information and notified when any progress is made concerning your report.

You can also choose the type of generated dump: minidump or full dump. Minidumps are usually very small (less than a megabyte) and contain only essential information. They are sometimes enough to diagnose and reproduce the problem. However for analyzing complex problems, a full dump is preferable, because it contains a copy of the whole process memory and allows to view the values of any local and global variables. The downside of a full dump is its size, which can easily exceed hundreds of megabytes.

Remember that the more crash dumps we receive the faster we release an update that fixes the bug.
All information is always submitted anonymously.

\section*{Crash Report Processing}

All received crash dumps are automatically analyzed and grouped according to the problem type.
HHD Software developers browse the crash report database on a regular basis. Reports received from maximum number of user computers receive the highest attention. They are fixed at the earliest time. Less frequent bugs may take longer to be fixed.

The lack of full dumps for complex problems may also postpone the fix of the bug

Having users' e-mails may actually speed up fixing particular bugs, even if they do not fall into "most frequent" category.

\section*{Proxy Settings}

This page allows you to configure the proxy server used by features Check for Updates, Downloader and Crash Dump Uploader. In most cases you don't need to make any changes, as defaults will work on majority of installations.

When Hex Editor Neo establishes a connection to remote server, it always does it over HTTP protocol over the 80 port. If your computer must use HTTP proxy when accessing HTTP resources, you may configure the proxy settings on this page. In addition, if your proxy server requires authentication, you may also configure credentials to use.


\section*{Server Settings}

There are four options you can use to configure a proxy server:

\section*{Default}

Hex Editor Neo will use the proxy server configured in your default browser. This is a default setting

\section*{Autodetect proxy settings}

Hex Editor Neo will try to automatically detect proxy server settings.

\section*{Do not use proxy}

Hex Editor Neo will bypass any configured proxy.

\section*{Use the following proxy (HTTP):}

Specify the address (and optionally port number) of the proxy server to use. Examples:
myserver.com
will use the HTTP proxy server myserver.com on port 80 (the default).
```

myserver.com:8080

```
will use the HTTP proxy server myserver.com on port 8080

\section*{Proxy Server Authentication}

There are three authentication options you can use

\section*{Authentication is not required}

Proxy Server does not require authentication

\section*{Authenticate me as logged on user}

Hex Editor Neo will use the currently logged on user to authenticate on the proxy server.

\section*{Authenticate me with the following credentials:}

Enter the user name and password. You must enter the same password in both Password and Re-type password boxes. See the Security Considerations section below.

\section*{Security Considerations}

When you configure the Hex Editor Neo to use the entered credentials, it stores entered password in registry under the HKEY_CURRENT_USER key. It uses encryption to store the password. Only the same user is able to decrypt the password.

That is, if encrypted password is copied to another computer, an attacker will not be able to get the plain text password

\section*{Binary Editor}

\section*{Tasks Settings}

This page allows you to configure tasks displayed in the Windows Jump List for the Hex Editor Neo application icon. This list is displayed when you right-click the pinned (or running) application icon and is also available in Start menu

There is a predefined list of tasks that may be configured to appear in a jump list. You are free to select the items to be included in the list. Remember that more tasks you select, less items will be left for recently opened documents and pinned documents.


Select a task to view its description in the right pane. Put a checkmark next to task to enable it.

\section*{Regular Expressions Settings}

Regular Expressions Settings page has several global options related to regular expressions, as well as allows you to manage named regular expression classes.


\section*{'.' (Dot) metacharacter does not match null character ('\0')}

Specify that the . metacharacter will not match a null character
' \(\quad\) ' (Dot) metacharacter does not match newline character (' \(\backslash \mathbf{n}\) ')
Specify that the . metacharacter will not match a newline character.

\section*{Non-escaped white-space is not significant}

Specify that non-escaped white-space characters in a regular expression are ignored.
See also the Creating and Managing Classes section for more information.

\section*{Editor Settings}

An editor window provides you with a large set of customizable properties. The Editor settings page in the Tools » Settings... is used to configure these properties.


\section*{Editor Schemes}

Hex Editor Neo allows you to associate a particular set of options with a specific file pattern. File pattern is specified using the regular expression. When a file is opened in the editor, its full path is matched against a regular expression and if match succeeds, the specific scheme is used. Otherwise, the scheme called "All documents" is used.

\section*{Configurable Options}

\section*{Font and Font Size}

You can select the font and font size to be used by an editor window. Only monospace fonts can be selected. Hex Editor Neo automatically scans your system for all compatible fonts and presents them in a Font combo box

\section*{Density}

This field lists all font density levels. At least regular and bold densities are usually present, while some fonts may provide with more intermediate density levels.

\section*{Line Spacing}

Allows fine-tuning of line height in the editor window.

\section*{Interlacing}

It is well known that humans better understand the screen table full of numbers if each odd table's row or column has different background color than even row or color. Hex Editor Neo supports both row and column interlacing. In addition, it allows you to specify odd and even row/column color separately.
Interlacing provides you with a choice from three options: "None", "Interlace columns" (the default one) and "Interlace rows".

\section*{Offset}

Specify the offset display mode. Choose from Hex, Decimal and Octal modes.

\section*{Display as}

Specify the default display mode. Choose from Hex, Decimal, Octal, Binary, Float and Double modes

\section*{Group by}

Specify the default grouping mode. Choose from Bytes, Words, Double words and Quad words modes.

\section*{Columns}

Specify the default number of columns. Choose one of the predefined values or enter any number from 1 to 9999.

\section*{Byte order}

Specify whether little-endian or big-endian byte order is used by the editor window.
Encoding
Select the default editor window encoding.

\section*{Smart Indentation}

When this option is ON, additional small gaps are inserted between columns to improve readability.


Smart indentation is applied only when the number of columns in a window is a power of 2 .

\section*{Code Pane}

Specify whether the Code Pane is visible.

\section*{Text Pane}

Specify whether the Text Pane is visible.
Any changes you make in this window automatically updates all applicable opened editor windows. If you have changed the specific options in the editor window, you can use the Take from current window button to copy those changes into this window.

Press the Reset to Default button to cancel your changes.

\section*{Default Values}

By default, when the new editor window is created, it gets the following settings:
\begin{tabular}{ll} 
Setting & Value \\
\hline Font & Lucida Console, Regular, 12pt \\
Interlacing & Columns \\
\hline Offset (address) display mode & Hex \\
Data display mode & Hex \\
Grouping & Byte \\
Columns & 16 \\
\hline Byte order & Little-Endian \\
Encoding & Default ANSI \\
Smart indentation & ON \\
Code pane & ON \\
Text pane & ON \\
\hline
\end{tabular}

\section*{Data Annotations Settings}

This settings page allows you to set the location where document's data annotations are stored.


The following storage locations are available:

\section*{NTFS Stream}

Data annotations are stored as an alternative NTFS data stream. It is usually a recommended option.

\section*{WARNING}

Data Annotations stored in an alternate NTFS data stream may be lost if file is copied to another file system. Windows Explorer usually displays a warning if such copy operation is attempted, however, other tools may not display such warning.

\section*{External file}

Data annotations are stored in an external file. Its name is calculated by adding a specific extension to original file name.

\section*{Custom folder}

All data annotations are stored in a custom folder

\section*{Auto select}

Let Hex Editor Neo automatically choose the storage location. The editor usually chooses the NTFS data stream, it if is available.

\section*{Other}

\section*{General Settings}

General Settings page contains application-wide settings


All settings on this page are divided into several groups.

\section*{General Group}

This group contains the following settings:

\section*{Always create backups}

Tells the editor to always create backup copies of the file before saving changes to it. See the Always create backups Option for more information.

\section*{Lowercase hexadecima}

If enabled, hexadecimal letters (' A ' to ' F ') are displayed in lower case, otherwise, in upper case.

\section*{Default byte order}

Specified the default byte order used by new editor windows. Byte Order can then be changed for each individual editor window.

\section*{Byte order change affects floating-point types}

Specifies whether byte order change affects encoding of floating point types, float and double. The encoding of floating point types may be different for different
platforms and files, so Hex Editor Neo provides you both methods.

\section*{Window switching style}

Specifies the window switching style. See the Window Switching section for more information.

\section*{Last recently used documents to display: \(\mathbf{N}\)}

Specify the number of last recently used documents to display in the File menu.

\section*{Close affects the current tool window only}

When enabled, clicking the Close button on the tool window closes only this window, if disabled, an entire tool window frame is closed

\section*{Restore main window position on startup}

If enabled, Hex Editor Neo restores the last position of its main window on startup.

\section*{Check for updates}

Can be "Every launch" (default), "Every week" or "Never". Specifies the frequency of checking for availability of new versions of the Hex Editor Neo.

\section*{Check for low disk space before performing operations}

Hex Editor Neo may require significant free space on one of your volumes for differnt "heavy" operations. It makes a "best guess" on the required hard disk space amount before processing the operation and warns you if there is too little free space. You may switch this option off, but do it on your own risk!

\section*{Foreground downloading}

Use all available bandwidth when downloading updates.

\section*{Editor Group}

This group contains the following settings:

\section*{Single Selection mode}

Change the way multiple selection works in Hex Editor Neo. See the Single Selection Mode Option for more information.

\section*{Adjust selection on Insert operation}

The current selection is shifted if you type new data with Insert Mode ON. This is a potentially slow operation and is not performed if you have a complex selection.

\section*{Digit grouping}

Affects the visualization of long numbers in decimal view types. Uses the thousands separator defined in the current profile to increase the readability of long decimal numbers.

\section*{Display cursor address on the left}

Specifies whether the cursor's address is displayed on the beginning of the current line.

\section*{Selection range export type}

Choose the way selection export works: use hexadecimal or decimal values when copying/exporting selection data.

\section*{Move cursor after pase}

Specifies whether to move the cursor after pasting data.

\section*{Notifications Group}

Hex Editor Neo has several notifications that are displayed to inform you about an important event. All these notifications have "Do not display anymore" checkbox. If you checked this box and want to restore the default behavior, use the options in this group to do it.

\section*{NTFS Streams}

This group contains the following settings:

\section*{Open stream after creation}

When enabled, the editor automatically opens a named stream after it is created (using NTFS Streams » Create Stream... command).

\section*{Printing}

This group contains the following settings:

\section*{Ignore Coloring}

When switched on, the editor ignores all coloring and prints the document in black and white. All highlighting is ignored and only selection is displayed on the paper (by inverting colors). When this option is switched off all highlighting is printed.

\section*{Standard header}

Print standard page header, which includes the document's name and current/total number of pages.

\section*{Standard footer}

Print standard page footer, which includes the name of the application printed the document.

\section*{Copy\&Export}

This group contains the following settings:

\section*{Float copy/export number of digits}

Specify the number of digits to be used while exporting single-precision floating-point numbers. See Formatted Data Format topic for more information

\section*{Double copy/export number of digits}

Specify the number of digits to be used while exporting double-precision floating-point numbers. See Formatted Data Format topic for more information.

\section*{Disassembler}

\section*{Display prefixes}

Set to display instruction prefixes

\section*{Display decimal values}

Set to display decimal values (hexadecimal otherwise)
Parse symbols

Set to parse module symbols, if available

\section*{Use AT\&T syntax}

Set to switch to displaying of AT\&T syntax for x86 and x86-64 disassembly. If not set, uses Intel syntax.

\section*{Show grid lines}

Improves window readability by drawing grid lines.

\section*{Process}

\section*{Preferred external debugger}

Select the preferred debugger to use for the Debug Snapshot command
- Registered JIT-debugger (default)
- Visual Studio
- WinDBG

\section*{Include full memory when debugging snapshots}

Set this option to ON if you want the debugger to access all snapshot memory when debugging snapshots.

\section*{Structure Viewer}

\section*{CSV Export Separator}

Choose what separator to use when exporting Structure Viewer's data into CSV file.

\section*{Maximum string length}

Limit the maximum number of characters in strings displayed in Structure Viewer.
Maximum number of inline fields or array items
Limit the amount of data in corresponding elements.

\section*{Purchasing Hex Editor Neo}

\section*{Choosing Appropriate Edition}

Hex Editor Neo is shipped in one of four different editions starting from Free Edition, which includes only basic functionality, up to Ultimate Edition, which combines all the power of advanced Hex Editor Neo features.

The detailed feature comparison chart is provided on-line
n addition, the Hex Editor Neo has its own built-in "feature assistance" engine. Every time you try to use a feature which is not a part of the current edition, Hex Editor Neo brings a popup window. The window contains short feature description and displays the edition this feature is implemented in.

\section*{Feature not Availble (closing in 27 seconds...)}

Disassembler feature you are trying to use is not available in the
current edition and is part of Ultimate Edition. Select an action be
to continue:
- Upgrade Now...
- Compare Editions
- Stalt 14 -days trial

Disassember module alows you to dicassember acter in fis or in
Disasserry. It supports several instruction sets and is capable of using symbol files.
memorn

If you have just started using Free Edition, you may select the "Start 14-days trial" option to get an Ultimate Edition for the next 14 days. All Hex Editor Neo features become available and you receive the full power of the Ultimate Edition. When the trial period is over, Hex Editor Neo switches back to the Free Edition. Trial period may not be started again on this computer, but you can always upgrade to Standard, Professional or Ultimate Edition.

\section*{Choosing Additional Options}

Several additional options and services are offered for those who purchase HHD Software products

\section*{Lifetime Upgrades Service}

According to our upgrade policy, all product bug fixes and minor upgrades (upgrades that do not change the major version number) are free for all customers. Major upgrades are available at special prices (major upgrades are free for customers owning subscription license types)

Our exclusive Lifetime Upgrade offer allows you to automatically receive ALL future product versions for free.
Please note that purchasing a product with Lifetime Upgrades service is cheaper and much more advantageous than upgrading product at a later time.
If you want to automatically receive all future versions of the purchased product for free, just check the Lifetime Upgrade option for your product in the shopping cart. If you are satisfied with bug fixes and minor upgrades only, uncheck this box during the order process.

\section*{How to Buy}

Online Orders
Online order is the fastest and cheapest way to get our products!
- Credit or Debit Card (Virtual Credit Cards are also supported).
- PayPal.
- Apple Pay.
- Microsoft Pay.
- Google Pay.

\section*{Offline Orders}
- Bank/Wire Transfer

We collect extra fee for such orders. This type of payment requires manual processing. Bank/Wire transfer usually takes from 2 to 5 days to complete.

\section*{Purchase Orders}

\section*{This option is not available for resellers}

We do not accept Purchase Orders directly. You may still purchase our products using this method from one of our official reseller partners. This is the slowest way to get our products. If you are still going to use this option, please do the following:
1. Add the products and services you are interested in to the Shopping Cart
2. Open the Shopping Cart
3. Select the required edition and license type for each product in the cart
4. Enter the quantity for each product in the cart
5. Check all required additional services
6. Enter coupon code(s), if you have them
7. Press the Quote button
8. Forward this quote to one of the official resellers

\section*{Ordering from Local Resellers and Distributors}

If you can't find suitable payment methods above, you may still purchase from one of our official resellers or distributor partners. However, it usually takes a lot of time to proceed due to the large amount of extra communications with significant delays. We recommend you to choose direct online ordering because it's much faster.

\section*{NOTE}

HHD Software may have an updated version of these purchasing instructions. To stay up to date, always refer to the online version of this page at HHD Software Web Site

\section*{My Account Web Site Section}

If you have ever purchased any of HHD Software products, you already have an account. It is automatically created at the time your shopping cart is transferred to the order processing server.
n My Account section you can get detailed information about your orders and purchased products, manage your personal data, download license files for purchased products, upgrade your products and so on.

The Password Hint function may be used to recover lost passwords
If you experience any difficulties logging into your personal account, please contact us. We will try to resolve your issue as soon as we can. Providing us with your Purchase ID (Order REF\#) will speed up the problem fixing.

\section*{Downloading License Files}

For security reasons Hex Editor Neo license file automatically expires in about a week. This of course does not apply to an installed license.
If you need another license file, you can always download it for free after logging into the My Account section of our web site.
License downloading procedure:
1. Go to the Products section (https://www.hhdsoftware.com/Account/products). You'll be asked for a login and password
2. Click on the product name in a list.
3. Click on the "Download License..." link.
4. Click the Open button to immediately install the license or the Save button to save the license file to disc.

See the Installing License Files section for information on how to install downloaded license file

\section*{Installing License Files}

This section describes how you can install the license file stored on the disc. Before using the license file make sure you have the most recent version of the Hex Editor Neo installed on your computer

Locate the downloaded license file in the Windows Explorer and double-click it. This will activate your copy of the Hex Editor Neo and unlock all features provided by the purchased license.

Alternatively, run the product, execute the Help » License Management... command and press the Install License... button. Locate the license file and press the Open button.


\section*{Upgrading Hex Editor Neo}

Hex Editor Neo is distributed in four different editions, including Free, Standard, Professional and Ultimate.
The powerful Hex Editor Neo functionality is flexibly divided between all these editions, allowing you to wisely choose the one appropriate for you. Please refer to the Choosing Appropriate Edition section for more information. Nevertheless, Hex Editor Neo may always be easily upgraded to a more powerful edition. Such upgrade is not free.

\section*{Upgrading from the Free Edition}

Free Edition, as told by its name, costs nothing to install and use. Although there are a lot of features included in the Free Edition, a number of powerful and advanced features are not, and therefore, you may find it useful to upgrade either to Standard, Professional or Ultimate edition.

The "Upgrade" in this case corresponds to the product purchase. Product purchasing options are described in more detail in corresponding section.
To start a purchase, you may use the Help " License Management... command to open up a License Management Window. Click the "Upgrade..." link next to the selected edition to start the order process.

\section*{Upgrading from Standard or Professional Editions}

If you currently have Hex Editor Neo Standard Edition licensed to you, you may upgrade either to the Professional or Ultimate Edition. Licensed Professional Edition may be upgraded to the Ultimate Edition. To perform an upgrade, navigate to the HHD Software Web Site'sUpgrade Section. You will be asked to log in, using your login and password. After that you will be able to choose the edition you want to upgrade to and proceed to the checkout.

As soon as your order is processed, the license file is e-mailed to you. Applying this license file activates the upgraded license and unlocks all its features.

\section*{Getting Support}

\section*{How to Get Support}

At HHD Software, we are committed to providing quality, bug-free software. If you experience any problems with our software or find any bugs, please don't hesitate to contact us using our contact page. Please provide a detailed description of your problem. Be sure to include which operating system you are running and the current version of the Hex Editor Neo (found in the Help » About).

Support can also be obtained by visiting our Knowledge Base. This page contains a list of frequently asked questions (FAQs) that might solve your problem.
If you have any other inquires, comments, ideas, or suggestions on things you would like Hex Editor Neo to do, you can provide your feedback using the Help » Feedback... option.

\section*{Built-In Updater}

Hex Editor Neo includes the built-in updating mechanism. It can automatically detect new product versions, as well as new language packs and their updates.
By default, the updater is configured to check for updates on each run. It downloads about 500 bytes from the HHD Software's web server and does not send any private information. You can change the frequency of update checks as well as completely disable it.

You can also always manually check for updates using the Help » Check for Updates command.

\section*{Component Updates Window}

This window is displayed when there new versions of product or installed language packs.


\footnotetext{
It lists all detected updates, with their currently installed version number and new version number. You can download selected updates, view changes or discard them. If you choose to discard updates, Hex Editor Neo will no longer notify you about those updates. It will, however, notify you of any future updates.
This window also allows you to change the default frequency of update checking. Please note that this option is also available on the General Settings page.
}

\section*{Settings Manager}

Settings Manager centralizes application settings storage and management. Settings Manager, introduced in version 6.20 and greatly improved in version 7.01 , stores settings for all Hex Editor Neo components. It supports the following storage locations:

\section*{Registry Settings Storage}

This mode is the default one for all installations except portable installations. In this mode, all settings are stored in per-user location in system registry.
Default folder for downloaded languages is \%LOCALAPPDATA\%\HHD Software\Hex Editor Neo\Localization (with backup folder \%INSTALLDIR\%\Localization).

\section*{Portable Settings Storage}

This mode is the default one for portable installations. All settings are stored in portable.hexset7 file, located in program startup folder. The presence of this file indicates the portable mode. Hex Editor Neo does not use or even accesses registry storage if started in portable mode.
Downloaded languages location is \%LAUNCHDIR\%\Localization where \%LAUNCHDIR\% is program startup folder.

\section*{External File Settings Storage}

This mode is similar to portable mode, but uses any given external file for storing application storage. In this mode, location of external file is still stored in per-user registry location (no other settings are stored in registry). The main purpose of this mode is to put external settings storage file into shared location, like OneDrive or Dropbox shared folder and have Hex Editor Neo automatically share its settings across all user's devices.

Default folder for downloaded languages is \%LOCALAPPDATA\%\HHD Software\Hex Editor Neo\Localization (with backup folder \%INSTALLDIR\%\Localization).

\section*{Exporting \& Importing Settings}

In addition to different storage locations, Settings Manager allows the user to manually export application settings to a given file and import the settings from a file. This may be used to copy application settings from one installation to another.

See the Settings Page topic for more information.

\section*{Settings Page}

This page allows the user to configure and use the Settings Manager.


The upper half of the window allows the user to select the settings storage location. See the Settings Manager topic for in-depth description of each location. For External File location mode a path to external file must be specified.

\section*{Settings Management}

Press the Export Settings... button to export application settings.
Press the Import Settings... button to import application settings from a file. It is recommended to restart the application after importing settings.
Press the Reset Settings... button to reset application settings. Application restart is required to complete settings reset.```


[^0]:    - ANSI - Arabic
    - ANSI - Baltic
    - ANSI - Central European
    - ANSI - Cyrillic
    - ANSI - Greek
    - ANSI - Hebrew
    - ANSI - Latin I
    - ANSI - Turkish
    - ANSI/OEM - Japanese, Shift-JIS
    - ANSI/OEM - Korean (Unified Hangul Code)
    - ANSI/OEM - Simplified Chinese (PRC, Singapore)
    - ANSI/OEM - Thai (same as 28605, ISO 8859-15)
    - ANSI/OEM - Traditional Chinese (Taiwan; Hong Kong SAR, PRC)
    - ANSI/OEM - Vietnamese
    - Arabic - ASMO 449+, BCON V4
    - Arabic - ASMO 708
    - Arabic - Transparent Arabic
    - Arabic - Transparent ASMO
    - CNS - Taiwan
    - Eten - Taiwan
    - EUC - Japanese
    - EUC - Korean
    - EUC - Simplified Chinese
    - EUC - Traditional Chinese
    - Europa 3
    - HZ-GB2312 Simplified Chinese
    - IA5 German (7-bit)
    - IA5 IRV International Alphabet No. 5 (7-bit)
    - IA5 Norwegian (7-bit)
    - IA5 Swedish (7-bit)

[^1]:    - OEM - Hebrew
    - OEM - Icelandi
    - OEM - Latin II
    - OEM - Modern Greek
    - OEM - Multilingual Latin I + Euro symbo
    - OEM - Multilingual Latin
    - OEM - Nordic
    - OEM - Portuguese
    - OEM - Russian
    - OEM - Turkish
    - OEM - United States
    - Russian - KOI8-R
    - Simplified Chinese (GB2312)
    - Simplified Chinese Extended and Simplified Chinese
    - Simplified Chinese
    - T. 61
    - TCA - Taiwan
    - TeleText - Taiwan
    - Ukrainian (KOI8-U)
    - US/Canada and Japanese
    - US/Canada and Traditional Chinese
    - US-ASCII (7-bit)
    - Wang - Taiwan

