

Experion PKS SafeView User's Guide

EPDOC-X120-en-431A
February 2015

Release 431

| Document | Release | Issue | Date |
|--------------------|---------|-------|---------------|
| EPDOC-X120-en-431A | 431 | 0 | February 2015 |

Disclaimer

This document contains Honeywell proprietary information. Information contained herein is to be used solely for the purpose submitted, and no part of this document or its contents shall be reproduced, published, or disclosed to a third party without the express permission of Honeywell International Sàrl.

While this information is presented in good faith and believed to be accurate, Honeywell disclaims the implied warranties of merchantability and fitness for a purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

In no event is Honeywell liable to anyone for any direct, special, or consequential damages. The information and specifications in this document are subject to change without notice.

Copyright 2015 - Honeywell International Sàrl

Contents

| | |
|---|-----------|
| 1 About This Document | 11 |
| 2 References | 13 |
| 3 Intended audience | 15 |
| 4 SafeView overview | 17 |
| 4.1 Managing windows | 18 |
| 4.1.1 Standard windows | 18 |
| 4.1.2 Problems with managing windows manually | 18 |
| 4.1.3 Wasted time | 18 |
| 4.1.4 Loss of view | 18 |
| 4.1.5 Unnecessary distractions | 18 |
| 4.1.6 Solving window management problems | 18 |
| 4.1.7 Crashed or terminated displays | 18 |
| 4.1.8 Terminating application processes | 18 |
| 4.2 Main features | 19 |
| 4.2.1 ViewPorts | 19 |
| 4.2.2 Functional summary | 19 |
| 4.3 Exclusions and special considerations | 20 |
| 4.3.1 Excluded applications and displays | 20 |
| 4.3.2 Test your applications | 20 |
| 4.3.3 Task Manager | 20 |
| 4.3.4 Using application titles | 20 |
| 4.3.5 For GUS scripts only: Using the Viewport. Open script | 20 |
| 4.3.6 Applications with pseudo-dialogs may close main display | 21 |
| 5 SafeView concepts | 23 |
| 5.1 Terminology | 24 |
| 5.1.1 Application | 24 |
| 5.1.2 Application window | 24 |
| 5.1.3 Category | 24 |
| 5.1.4 Constant expression | 24 |
| 5.1.5 Control panel | 24 |
| 5.1.6 Default MATCH expression | 24 |
| 5.1.7 Display history dialog | 24 |
| 5.1.8 Fast button | 25 |
| 5.1.9 First match group | 25 |
| 5.1.10 Focus-based search | 25 |
| 5.1.11 Global output focus | 25 |
| 5.1.12 Graphical Workspace Editor (GWE) | 25 |
| 5.1.13 Group | 25 |
| 5.1.14 Hot keys | 25 |
| 5.1.15 Infinite | 26 |
| 5.1.16 Keyword | 26 |
| 5.1.17 Lock button | 26 |
| 5.1.18 Manual select group | 26 |
| 5.1.19 Match expression | 26 |

| | | |
|----------|---|-----------|
| 5.1.20 | MAXWINDOWS | 26 |
| 5.1.21 | Module | 26 |
| 5.1.22 | Native window | 26 |
| 5.1.23 | Output focus | 27 |
| 5.1.24 | Output focus button | 27 |
| 5.1.25 | Placeholder | 27 |
| 5.1.26 | Prior display | 27 |
| 5.1.27 | Regular expression | 27 |
| 5.1.28 | Registered application display | 28 |
| 5.1.29 | Round robin group | 28 |
| 5.1.30 | SafeView-aware | 28 |
| 5.1.31 | System button | 28 |
| 5.1.32 | Tabbed Displays | 29 |
| 5.1.33 | Text Editor | 29 |
| 5.1.34 | Themes | 29 |
| 5.1.35 | ViewPort | 30 |
| 5.1.36 | Window | 30 |
| 5.1.37 | Window category | 30 |
| 5.1.38 | Window group | 30 |
| 5.1.39 | Window management | 35 |
| 5.1.40 | Window of output focus | 35 |
| 5.1.41 | Window specification | 35 |
| 5.1.42 | Window title | 35 |
| 5.1.43 | Windows XP | 35 |
| 5.1.44 | Windows 7 | 35 |
| 5.1.45 | Workspace | 35 |
| 5.1.46 | Workspace configuration | 35 |
| 5.1.47 | Workspace Definition Language (WDL) | 36 |
| 5.2 | Workspace organization | 37 |
| 5.2.1 | Workspace groups | 37 |
| 5.2.2 | Workspace - Round robin group | 37 |
| 5.2.3 | Workspace - Manual select group | 38 |
| 5.2.4 | Workspace - First match group | 38 |
| 5.2.5 | Workspace - Match expressions | 39 |
| 5.2.6 | When no match is intended | 39 |
| 5.2.7 | A MATCH expression is a logical expression | 40 |
| 5.2.8 | Workspace - Wildcards | 40 |
| 5.2.9 | Workspace - Group matching | 40 |
| 5.2.10 | Regular expressions make matching easier | 41 |
| 5.2.11 | Using the NOT operator | 41 |
| 5.2.12 | Default MATCH expression | 41 |
| 5.2.13 | Window specification | 41 |
| 5.2.14 | Top-down or focus-based matching | 42 |
| 5.3 | Managed windows | 43 |
| 5.3.1 | Managed - Indicator buttons | 43 |
| 5.3.2 | Managed - Output focus windows | 43 |
| 5.3.3 | Output focus in multiple-group workspaces | 43 |
| 5.3.4 | Managed - Invisible output focus | 44 |
| 5.3.5 | Managed - Global output focus | 44 |
| 5.3.6 | Multiple-window Station and tabbed displays | 45 |
| 5.3.7 | Managed - Locking a window | 47 |
| 5.3.8 | Managed - Fast mode displays | 48 |
| 6 | Designing a workspace | 51 |
| 6.1 | Workspace design | 52 |

| | | |
|----------|---|-----------|
| 6.1.1 | Workspace design overview | 52 |
| 6.1.2 | Workspace configuration files | 52 |
| 6.2 | Configuration file format | 53 |
| 6.2.1 | Configuration file format description | 53 |
| 6.2.2 | Workspace items | 53 |
| 6.2.3 | Window group | 55 |
| 6.3 | Design considerations | 60 |
| 6.3.1 | Design considerations overview | 60 |
| 6.3.2 | Organize the workspace | 60 |
| 6.3.3 | Make sure resized windows are usable | 60 |
| 6.3.4 | Determine area available for a client picture | 60 |
| 6.3.5 | Users who can specify 'Always On Top' | 61 |
| 6.3.6 | Include a default MATCH expression | 61 |
| 6.3.7 | Include a 'catch-all' window group | 61 |
| 6.3.8 | Close managed windows | 62 |
| 6.3.9 | Configure sufficient windows | 62 |
| 6.3.10 | Legacy TPS faceplate limitation for ES-T | 63 |
| 6.3.11 | Configure Microsoft Word applications | 63 |
| 6.3.12 | Configure LCN status applet | 63 |
| 7 | Configuring a workspace | 65 |
| 7.1 | Using the Text Editor or Graphical Workspace Editor | 66 |
| 7.2 | Before you begin | 67 |
| 7.3 | Using the Text Editor to configure a workspace | 68 |
| 7.3.1 | Text Editor Description | 68 |
| 7.3.2 | Starting the Text Editor | 68 |
| 7.3.3 | Building a new workspace configuration file | 68 |
| 7.3.4 | Changing an existing workspace configuration file | 68 |
| 7.3.5 | Creating a new workspace configuration file | 69 |
| 7.3.6 | Opening an existing workspace configuration file | 69 |
| 7.3.7 | Checking the workspace configuration file syntax | 70 |
| 7.4 | Using the Graphical Workspace Editor to configure a workspace | 72 |
| 7.4.1 | Graphical Workspace Editor description | 72 |
| 7.4.2 | Tree view of a workspace configuration | 73 |
| 7.4.3 | Workspace hierarchy | 73 |
| 7.4.4 | Starting the GWE | 73 |
| 7.4.5 | Configuring a new workspace | 73 |
| 7.4.6 | Configuring tabbed displays on multi-window station | 73 |
| 7.4.7 | Changing an existing workspace | 74 |
| 7.4.8 | Creating a new workspace | 75 |
| 7.4.9 | Opening an existing workspace | 75 |
| 7.4.10 | Navigating the workspace hierarchy | 76 |
| 7.4.11 | Saving the active workspace | 77 |
| 7.4.12 | Checking the workspace syntax | 78 |
| 7.4.13 | Starting TextEdit from the GWE | 79 |
| 7.4.14 | Cutting a tree item | 79 |
| 7.4.15 | Copying a tree item | 80 |
| 7.4.16 | Pasting a tree item | 80 |
| 7.4.17 | Displaying the GWE placeholder menu | 80 |
| 7.4.18 | Enabling/disabling the ALWAYS ON TOP property | 80 |
| 7.4.19 | Enlarging/reducing the placeholder display grid | 81 |
| 7.4.20 | Displaying the GWE component menu | 81 |
| 7.4.21 | Showing/hiding placeholders | 82 |
| 7.4.22 | Adding a new window specification | 82 |
| 7.4.23 | Adding a new group | 83 |

| | | |
|----------|---|------------|
| 7.4.24 | Updating window layout data from placeholders | 83 |
| 7.4.25 | Editing techniques | 84 |
| 7.4.26 | Property windows | 84 |
| 7.4.27 | Editing the workspace general properties | 84 |
| 7.4.28 | Editing the workspace startup commands | 85 |
| 7.4.29 | Editing the workspace constants | 87 |
| 7.4.30 | Editing the window general properties using the Window Specification window | 88 |
| 7.4.31 | Editing the group default styles using the Window Group window | 90 |
| 7.4.32 | Editing the window general properties using the Window Specification window | 91 |
| 7.4.33 | Editing the window style using the Window Specification window | 92 |
| 7.4.34 | Editing the window specifications graphically | 94 |
| 8 | Managing a workspace (SafeView run time) | 97 |
| 8.1 | Functions overview | 98 |
| 8.2 | Command line options | 99 |
| 8.3 | Using SafeView to manage a workspace | 100 |
| 8.3.1 | Terminating SafeView management | 100 |
| 8.3.2 | Starting SafeView | 100 |
| 8.3.3 | Loading a workspace | 100 |
| 8.3.4 | Repositioning managed windows | 102 |
| 8.3.5 | Managing existing displays | 102 |
| 8.3.6 | Hiding/displaying the SafeView Control Panel menu | 103 |
| 8.3.7 | Showing the prior display | 103 |
| 8.3.8 | Showing history: listing and invoking prior displays | 104 |
| 8.3.9 | Errors | 104 |
| 8.3.10 | Showing/hiding placeholders | 105 |
| 8.3.11 | Displaying information about SafeView | 106 |
| 8.3.12 | Assigning output focus to a window | 106 |
| 8.3.13 | Assigning global output focus to a window | 107 |
| 8.3.14 | Locking a round robin window | 108 |
| 8.3.15 | Configuring Fast mode displays | 109 |
| 8.3.16 | Using the Task Manager | 110 |
| 8.3.17 | Exiting SafeView | 110 |
| 8.3.18 | System overload | 110 |
| 8.3.19 | Getting help for the Control Panel | 111 |
| 8.3.20 | Changing the SafeView search mode (Normal or Focus) | 112 |
| 9 | Application Program Interface | 113 |
| 9.1 | Application Program Interface Overview | 114 |
| 9.1.1 | Accessing API calls: GUS and Experion Station PKS displays | 114 |
| 9.1.2 | API calls related to focus-based search mode | 114 |
| 9.1.3 | Script examples | 115 |
| 9.1.4 | SafeView-aware applications | 115 |
| 9.1.5 | Workspace configuration | 115 |
| 9.2 | Guidelines for creating SafeView-aware applications | 116 |
| 9.2.1 | Register window calls | 116 |
| 9.2.2 | Determine the window category | 116 |
| 9.2.3 | When to register the window | 116 |
| 9.2.4 | Multiple top-level windows | 117 |
| 9.2.5 | Scale a display window | 117 |
| 9.2.6 | Boolean return values | 117 |
| 9.2.7 | HWND versus long value types | 117 |
| 9.2.8 | Case sensitivity | 117 |
| 9.3 | List of API calls | 118 |
| 9.3.1 | CloseWindow | 118 |

| | | |
|-----------|--|------------|
| 9.3.2 | Configuration Loaded | 118 |
| 9.3.3 | CreateWorkspaceClient | 119 |
| 9.3.4 | DisplayHistoryDialog | 120 |
| 9.3.5 | GetCurrentForwardDisplayIndex | 121 |
| 9.3.6 | GetCurrentPriorDisplayIndex | 121 |
| 9.3.7 | Get FocusBasedSearch | 122 |
| 9.3.8 | GetLastWorkspaceError | 122 |
| 9.3.9 | GetPriorDisplayAtIndex | 123 |
| 9.3.10 | GetWindowGroup | 123 |
| 9.3.11 | GetWindowHandle | 124 |
| 9.3.12 | GetWindowSpecName | 124 |
| 9.3.13 | HidePlaceholders | 124 |
| 9.3.14 | InvokeForwardDisplay | 125 |
| 9.3.15 | InvokePriorDisplay | 125 |
| 9.3.16 | InvokePriorDisplayAtIndex | 126 |
| 9.3.17 | IsManagerActive | 126 |
| 9.3.18 | IsWindowFocus | 127 |
| 9.3.19 | IsWindowGlobalFocus | 127 |
| 9.3.20 | IsWindowLocked | 128 |
| 9.3.21 | IsWindowManaged | 129 |
| 9.3.22 | IsWindowRegistered | 129 |
| 9.3.23 | IsWindowSpecFocus | 130 |
| 9.3.24 | IsWindowSpecGlobalFocus | 131 |
| 9.3.25 | IsWindowSpecLocked | 131 |
| 9.3.26 | IsWindowSpecManaged | 132 |
| 9.3.27 | LoadWorkspaceDialog | 133 |
| 9.3.28 | LoadWorkspaceFile | 133 |
| 9.3.29 | ManageExisting | 134 |
| 9.3.30 | OriginalPositions | 134 |
| 9.3.31 | RegisterWindowCategory | 135 |
| 9.3.32 | RegisterWindowData | 136 |
| 9.3.33 | RegisterWindowDataSpec | 137 |
| 9.3.34 | .SetCurrentDisplayIndex | 139 |
| 9.3.35 | SetFocusBasedSearch | 140 |
| 9.3.36 | SetGlobalOutputFocus | 140 |
| 9.3.37 | SetGlobalOutputFocusOnce | 141 |
| 9.3.38 | SetGlobalOutputFocusWnd | 141 |
| 9.3.39 | SetGlobalOutputFocusWndOnce | 142 |
| 9.3.40 | SetLock | 142 |
| 9.3.41 | SetLockWnd | 143 |
| 9.3.42 | SetOutputFocus | 144 |
| 9.3.43 | SetOutputFocusWnd | 144 |
| 9.3.44 | ShowPlaceholders | 145 |
| 9.3.45 | StartManager | 145 |
| 9.3.46 | StopManager | 146 |
| 9.3.47 | TerminateOnClose | 147 |
| 9.3.48 | ToggleStandardDialog | 148 |
| 9.3.49 | UnmanageWindow | 148 |
| 9.3.50 | WSConfigurationLoaded | 149 |
| 9.3.51 | WSGetWindowSpecification | 150 |
| 9.3.52 | WSTerminateOnClose | 150 |
| 10 | Safeview workspace examples | 153 |
| 11 | Workspace overview | 155 |

| | | |
|-----------|---|------------|
| 11.1 | New SafeView APIs | 156 |
| 11.2 | Display resolution | 157 |
| 11.3 | SafeView Text Editor and Graphical Editor | 158 |
| 11.4 | Faceplates | 159 |
| 12 | Experion single window, single screen | 161 |
| 12.1 | Single window, single screen workspace description | 162 |
| 12.2 | Single window, single screen associated .wdl file names | 163 |
| 12.3 | Single window, single screen human interfaces | 164 |
| 13 | Experion single window, dual horizontal | 167 |
| 13.1 | Single window, dual horizontal Workspace description | 168 |
| 14 | GUS NW catch-all, dual horizontal | 169 |
| 14.1 | GUS NW catch-all, dual horizontal Workspace description | 170 |
| 15 | Experion single window, dual vertical | 171 |
| 15.1 | Single window, dual vertical workspace description | 172 |
| 16 | GUS NW dual vertical, catch-all | 173 |
| 16.1 | GUS NW dual vertical, catch-all Workspace description | 174 |
| 17 | Experion multi-window, single screen | 175 |
| 17.1 | Multi-window, single screen Workspace description | 176 |
| 18 | Experion multi-window, single screen, 2FP, dedicated | 177 |
| 18.1 | Multi-window, single screen, 2FP, dedicated Workspace description | 178 |
| 19 | Experion multi-window, dual horiz, MS, 2T, 4FP | 179 |
| 19.1 | Multi-window, dual horiz, MS, 2T, 4FP Workspace description | 180 |
| 20 | Experion multi-window, dual horiz, MS, hidden taskbar, 2T, 4FP | 181 |
| 20.1 | Multi-window, dual horiz, MS, hidden taskbar, 2T, 4FP Workspace description | 182 |
| 21 | Experion multi-window, dual horiz, MS, 6T, 5FP | 183 |
| 21.1 | Multi-window, dual horiz, MS, 6T, 5FP workspace description | 184 |
| 22 | Experion multi-window, dual horiz, MS, NavWin, 6T, 5FP | 185 |
| 22.1 | Multi-window, dual horiz, MS, NavWin, 6T, 5FP Workspace description | 186 |
| 23 | Experion multi-window, dual horiz, MS, SR, 2T, 4FP | 187 |
| 23.1 | Multi-window, dual horiz, MS, SR, 2T, 4FP Workspace description | 188 |
| 24 | Experion multi-window, dual horiz, FB, SR, 4T, 4FP | 189 |
| 24.1 | Multi-window, dual horiz, FB, SR, 4T, 4FP Workspace description | 190 |
| 25 | Experion multi-window, dual horiz, dual cmnd, FB, SR, 4T, 4FP | 191 |
| 25.1 | Multi-window, dual horiz, dual cmnd, FB, SR, 4T, 4FP Workspace description | 192 |
| 26 | Experion multi-window, dual horiz, dual cmnd, FB, SR, 12T, 6FP | 193 |
| 26.1 | Multi-window, dual horiz, dual cmnd, FB, SR, 12T, 6FP Workspace description | 194 |
| 27 | Experion multi-window, quad horiz, quad cmnd, FB, SR, 4T, 8FP | 195 |
| 27.1 | Multi-window, quad horiz, quad cmnd, FB, SR, 4T, 8FP Workspace description | 196 |
| 28 | Experion multi-window, dual vertical, FB, 2T, 4FP | 197 |
| 28.1 | Multi-window, dual vertical, FB, 2T, 4FP Workspace description | 198 |
| 29 | Experion multi-window, dual vertical, dual cmnd, FB, 4T, 4FP | 199 |
| 29.1 | Multi-window, dual vertical, dual cmnd, FB, 4T, 4FP Workspace description | 200 |
| 30 | Experion multi-window ICON, dual cmnd, FB, SR, 8T, 8FP | 201 |

| | |
|--|------------|
| 30.1 Multi-window ICON, dual cmnd, FB, SR, 8T, 8FP Workspace description | 202 |
| 31 Experion multi-window ICON, dual cmnd, FB, SR, 12T, 8FP | 203 |
| 31.1 Multi-window ICON, dual cmnd, FB, SR, 12T, 8FP Workspace description | 204 |
| 32 Experion multi-window ICON, dual cmnd, FB, SR, 12T, 12FP | 205 |
| 32.1 Multi-window ICON, dual cmnd, FB, SR, 12T, 12FP Workspace description | 206 |
| 33 Experion multi-window ICON, quad cmnd, FB, SR, 8T, 8FP | 207 |
| 33.1 Multi-window ICON, quad cmnd, FB, SR, 8T, 8FP Workspace description | 208 |
| 34 EPKS single screen | 209 |
| 34.1 EPKS single screen Workspace description | 210 |
| 35 EPKS dual screen | 211 |
| 35.1 EPKS dual screen Workspace description | 212 |
| 36 EPKS quad 1x4 screen | 213 |
| 36.1 EPKS quad 1x4 screen Workspace description | 214 |
| 37 EPKS quad 2x2 screen | 215 |
| 37.1 EPKS quad 2x2 screen Workspace description | 216 |
| 38 Appendix A: Workspace Definition Language | 217 |
| 38.1 Workspace Definition Language files | 218 |
| 38.1.1 WDL file format | 218 |
| 38.2 Syntax notation | 219 |
| 38.3 Lexical conventions | 220 |
| 38.3.1 Tokens | 220 |
| 38.4 Workspace syntax and semantics | 221 |
| 38.4.1 Workspace Syntax | 221 |
| 38.4.2 Workspace Semantics | 221 |
| 38.5 Group syntax and semantics | 222 |
| 38.5.1 Group Syntax | 222 |
| 38.5.2 Group Semantics | 222 |
| 38.6 Window syntax and semantics | 224 |
| 38.6.1 Window Syntax | 224 |
| 38.6.2 Window Semantics | 224 |
| 38.6.3 The region concept | 226 |
| 38.6.4 The placeholder concept | 227 |
| 38.6.5 Meaning of coordinate values | 227 |
| 38.6.6 Multiply specified properties | 227 |
| 38.6.7 Relationships between SIZE and POSITION attributes | 227 |
| 38.7 MATCH expressions syntax and semantics | 228 |
| 38.7.1 MATCH expressions Syntax | 228 |
| 38.7.2 MATCH expressions Semantics | 228 |
| 38.7.3 Example MATCH expressions | 229 |
| 38.8 Expressions syntax and semantics | 232 |
| 38.8.1 Expressions Syntax | 232 |
| 38.8.2 Expressions Semantics | 232 |
| 38.9 Regular expression pattern syntax | 234 |
| 38.9.1 Regular expressions in MATCH operators | 234 |
| 38.9.2 Syntax and semantics of regular expressions | 234 |
| 38.10 Predefined constants | 236 |
| 38.11 Miscellaneous | 237 |
| 38.11.1 Miscellaneous Syntax | 237 |
| 38.11.2 Miscellaneous Semantics | 237 |

| | |
|---|------------|
| 39 Appendix B: Workspace client Visual C++ code | 239 |
| 39.1 Visual C++ sample | 240 |
| 40 Appendix C: SafeView error messages | 243 |
| 40.1 Error messages list | 244 |
| 41 Appendix D: SafeView tips and tricks | 249 |
| 41.1 Populating a workspace automatically on startup | 250 |
| 41.2 Normal search mode vs. focus-based search mode | 251 |
| 41.2.1 Normal search mode | 251 |
| 41.2.2 Focus-based search mode | 251 |
| 41.3 Supporting multiple command windows in Station | 253 |
| 41.3.1 Configuring Station for multiple command windows | 254 |
| 41.4 Supporting additional displays in multi-window Station | 256 |
| 41.4.1 Configuring Station to support additional displays | 256 |
| 41.5 Station's configured startup page vs. Station startup commands | 257 |
| 41.6 SafeView's graphical editor vs. text-based editor | 258 |
| 41.7 The focus-based, programmatic, multi-display launch problem | 259 |
| 41.7.1 Workaround | 259 |
| 41.8 SafeView's command line options | 261 |
| 41.9 SafeView's facilitation of Station client tracing | 262 |
| 41.10 Two MATCH-expression options for managing Experion faceplates | 263 |
| 41.11 Inclusive/exclusive 'everything except' MATCH expressions | 264 |
| 41.12 Yoking faceplates to main displays | 265 |
| 41.13 Enhancing consistency and safety with region constraints | 266 |
| 41.14 Closing faceplates automatically on main display close | 267 |
| 41.14.1 Example scenario | 267 |
| 41.14.2 Example script | 267 |
| 41.15 Managing cross-screen, focus-based displays | 269 |
| 41.16 Supporting multiple catch-all windows | 270 |
| 41.17 Using wildcard character-matching | 271 |
| 41.18 Constant-based vs. literal-based coordinates | 272 |
| 42 Notices | 273 |
| 42.1 Documentation feedback | 274 |
| 42.2 How to report a security vulnerability | 275 |
| 42.3 Support | 276 |
| 42.4 Training classes | 277 |

1 About This Document

This document describes the SafeView application used to control the display characteristics of application windows within a Microsoft Windows workspace. The document describes how to design and configure a workspace using SafeView.

Revision history

| Revision | Date | Description |
|----------|---------------|------------------------------|
| A | December 2013 | Initial release of document. |

2 References

The following list identifies all documents that may be sources of reference for material discussed in this publication.

| Document Title | Document ID |
|--|-------------------|
| GUS Display Builder User's Guide | EPDOC-XX44-en-410 |
| TPS System Implementation Guide for Windows 2000 | TP08X |
| Experion PKS Server and Client Configuration Guide | EPDOC-X127-en-410 |
| Experion PKS Display Building Guide | EPDOC-XX30-en-410 |
| Experion PKS HMIWeb Display Building Guide | EPDOC-XX54-en-410 |

3 Intended audience

This documentation is intended for the following audience:

- Engineers to configure the SafeView workspace, create associated applications, and operate the run-time SafeView workspace.
- Operators to operate the run-time SafeView workspace.

4 SafeView overview

Related topics

“Managing windows” on page 18

“Main features” on page 19

“Exclusions and special considerations” on page 20

4.1 Managing windows

4.1.1 Standard windows

In a typical multi-windowed environment such as the Microsoft Windows operating system, applications operate independently from one another. The user has to manage the workspace. Applications can specify such window attributes as origin, size, and style, but this is often done by the operating system. In addition, there is no standard mechanism provided by Windows to coordinate or predict where multiple windows are placed. There is no concept of “application replacement,” where calling up one application display will close another application's display. Either the user or the display itself initiates the closure.

4.1.2 Problems with managing windows manually

Because Microsoft Windows provides no mechanism for managing windows, the user must manage the workspace. This can cause a number of problems. Three of the most serious problems are wasted time, loss of view, and unnecessary distractions.

4.1.3 Wasted time

If you need to have several windows open at the same time, you will typically have to spend time positioning, sizing, minimizing, and maximizing windows to make sure that you can view all the applications.

4.1.4 Loss of view

If you need to have several windows open at the same time, it is easy to have a window hidden behind other windows.

4.1.5 Unnecessary distractions

Managing windows manually requires focused hand-eye coordination to perform tasks such as window movement and sizing. These tasks have nothing to do with primary process control tasks, such as maintaining control and responding to plant alarms. Performing these tasks can only be a distraction.

4.1.6 Solving window management problems

SafeView enables you to create well-organized groupings of window specifications. This organizational capability, coupled with the ability to map application display windows to desired specifications, enables you to coordinate window placement across applications.

4.1.7 Crashed or terminated displays

SafeView checks each managed window every 30 seconds to make sure they still exist. If a nonexistent window is found (one that has crashed or been terminated by the user), SafeView automatically restores the ViewPort.

4.1.8 Terminating application processes

If a SafeView-managed display window is closed, but the process associated with the window does not terminate, SafeView can be employed, if desired, to terminate the process after a specified amount of time.

4.2 Main features

4.2.1 ViewPorts

Within the Microsoft Windows environment, the windows through which applications are viewed can be thought of as “ViewPorts.” While standard Windows exist only until the application appearing in the window is closed, ViewPorts are long lasting and can hold many hundreds of applications. New application displays replace old displays within the same ViewPort, similar to the way a “sticky window” functions, retaining the same window attributes such as origin, size, and style. ViewPorts possess a number of properties that can be set up to streamline and unify the user's experience of working in a multi-display environment.

You can think of a ViewPort as the visual representation of a window specification. A window specification is simply a set of many properties that is assigned to an application display at run time. There are a number of Windows properties that you are already familiar with, such as SIZE and POSITION. Other properties, such as REGION and DRAGGABLE, provide additional ways of controlling application windows. Window specifications will be defined in detail later on.

4.2.2 Functional summary

The SafeView package allows you to perform the following functions:

- Configure various window parameters that define the initial location and size of application windows, and limit subsequent changes to the position or size of the windows. You can define a region, outside of which certain applications cannot grow or move. In such a case, even if the window is maximized, it will not cover the entire screen. This will allow you to specify “protected area” in which critical displays can be positioned so they cannot be overlaid by other less critical displays.
- Define a set of applications to be automatically invoked when a workspace is loaded.
- Manage various application windows based on any combination of the application file name, window title, and window “category.” All kinds of applications including Honeywell, customer, and third-party applications can be managed.
- Configure ViewPorts (window specifications) to constrain application windows in ways that are not available in the Windows environment. For example, you can set up a ViewPort with the ALWAYS ON TOP attribute to provide this functionality to an application display that does not itself have this feature. You can also configure a ViewPort to be “non-draggable” and “non-sizeable,” so that displays cannot be moved from their current location.
- Organize application windows into logical groups and subgroups, based on configurable window selection strategies. This organizational capability enables you to locate application displays to specific locations on the screen.
- Define a set of applications that will be automatically turned on when the workspace is loaded. Note that Display History only displays “registered” applications. All Honeywell third party applications can be registered, if desired.
- List and then invoke any of the last 50 application displays that appeared in managed windows and then were closed. This is accomplished using the Display History function.
- Identify and be able to designate the ViewPort in which the next application to be initiated will appear. This is accomplished by the automatic inclusion of one or more ViewPort indicator buttons in the title bar of a managed display.

4.3 Exclusions and special considerations

4.3.1 Excluded applications and displays

SafeView, by design, excludes some applications from management. Here is a list of the displays and applications that SafeView does not manage:

- Windows Explorer (*EXPLORER.exe*)
- Microsoft Office Manager
- MS DOS/Console applications (*NTVDM.exe*)
- Visual C++ Development (*MSDEV.exe*)
- Spy++(*SPYXX.exe*)
- SafeView Graphical Workspace Editor (*SVEDIT.exe*)
- Problem Step Recorder (*PSR.exe*)
- Honeywell LCNP Status Applet (*EMSTATUS.exe*)
- MS Calculator (*CALC.exe*)
- Honeywell Dbtrace Configuration Utility (*TRACEUI.exe*)
- Honeywell Configuration Utility (*CONFIG.exe*)
- SafeView Configuration Applet (*SVCONFIG.exe*)
- Honeywell Message-Transfer Applet (*MSGTRANSFER.exe*)

These displays and applications will appear in their standard Windows formats.

4.3.2 Test your applications

On rare occasions, SafeView may not be able to detect and manage a given application. You should attempt to verify that an application can be managed before deploying. Trying to use SafeView to manage such an application may even cause it to crash.

4.3.3 Task Manager

Do not use the Task Manager to end a task associated with a placeholder or the *SAFEVIEW.exe* file. If you do, SafeView will terminate.

4.3.4 Using application titles

If an application adds a document name to its title after it is launched, SafeView may not be able to manage the application successfully if you use the document name to identify the application. This will occur if the title check is not completed *before* the document name is added. However, this kind of application can be launched using the SafeView “CreateWorkspaceClient” on page 119” API call.

4.3.5 For GUS scripts only: Using the Viewport. Open script

When you use the **viewport.open** command in a GUS script, it results in the viewport application being opened behind the calling display. To allow Viewport to appear on top, you must explicitly activate it from scripting. An example of a correct script would be:

```
viewport.open "myviewport", 100, 100, 400, 200
appActivate "myviewport"
```

4.3.6 Applications with pseudo-dialogs may close main display

One of SafeView's main benefits is its ability to close old main displays to accommodate new main displays. Occasionally, a third-party application may use a pop-up dialog box that *appears* to the operating system (and thus SafeView) as a main display. While dialog boxes always have a parent window, SafeView never affects them. However, since some of these third-party dialog boxes *are* actually main display applications, SafeView may accommodate them by closing the *true* main display. Thus, if using third-party software in your main displays, you can prepare for this scenario by explicitly excluding such pop-ups from the main display's MATCH expression (such as through title-based exclusion).

5 SafeView concepts

5.1 Terminology

5.1.1 Application

An executing program. A process that, for the purpose of this document, has one or more user-visible displays or application windows.

5.1.2 Application window

A window, in the sense of a Microsoft Windows display window, in which an application is made visible to the user. In this document, an application window is one that is not a component of another (parent) window, but rather is an independently displayed window that has no user-interaction kind of relationship (that is, not a modal “file open” dialog box) with a parent window.

5.1.3 Category

See ““Window category” on page 30.”

5.1.4 Constant expression

An expression that evaluates to a number or string constant. These exist to simplify the use of common data, such as the size of a particular display region. Workspace configurations can be created manually by using a UNICODE text editor, such as the SafeView Text Editor.

5.1.5 Control panel

The standard human interface dialog box that is available with the SafeView product. This control panel, which can be shown or hidden by pressing the **Ctrl+Alt+W** keys simultaneously at any time, provides the operator or engineer the means to control SafeView.

5.1.6 Default MATCH expression

The `DEFAULT = true/false` language construct can be associated with, at most, one MATCH expression in a workspace configuration. Only registered, workspace client windows use this mechanism. This *default* option applies in the case where a workspace client fails to match any of the MATCH expressions in the entire configuration. This can occur, for example, when new displays with new categories are utilized within an existing workspace configuration. When a match of a registered window fails, the default MATCH expression, if provided, is forced *true* and the window is managed accordingly.

Note, however, that you can *exempt* a workspace client from SafeView's default category matching function by setting its category value to either the empty string (NULL or “”), or a single blank character (“ ”).

For the DEFAULT expression to be effective when a MATCH expression includes a wildcard, such as `title=(“?*”)`, the wildcard expression should be: `match = title(“?*”) and not category(“?*”)`.

5.1.7 Display history dialog

A dialog box containing a list of up to fifty of the most recently displayed (and removed) applications. This dialog can be used to invoke a formerly displayed application.

5.1.8 Fast button

The Fast button in the title area of the GUS, Native window, and Station displays window allows the operator to update the display data at a pre-configured fast rate. Only one display is fast updated at a given point in time.

5.1.9 First match group

This kind of window group searches for a matching subgroup and/or window specification according to the physical ordering of this information in the workspace configuration file.

5.1.10 Focus-based search

The focus-based search option allows the user to specify whether SafeView will, for a given workspace, use its default “top down” search mode or the “focus-based” search mode, which searches the group that owns the currently active window first.

5.1.11 Global output focus

This is a workspace-wide version of output focus. Only one window in the entire workspace can have this status at any one time and it overrides any group-level *output focus* windows in multiple-group workspaces.

5.1.12 Graphical Workspace Editor (GWE)

This is a SafeView application that provides a graphical human interface for creating and editing workspace configuration files. The Graphical Workspace Editor provides displays of the structure of a given workspace configuration, in terms of the workspace and its component window groups and window specifications. Each workspace component can be individually viewed and edited using the “tabbed property page” dialogs. There is no API programmatic interface provided by the GWE. Although this application duplicates the majority of the functionality provided by the Text Editor, the Text Editor has been retained.

5.1.13 Group

See “Window group items” .

5.1.14 Hot keys

A hot key provides direct keystroke access to certain SafeView functions regardless of which application has user input focus. Hot keys are provided to

- Enable/disable the focus-based search option (**Ctrl+Alt+F**).
- Invoke the Display History dialog box (**Ctrl+Alt+H**),
- Reposition all application displays to their original positions (**Ctrl+Alt+O**),
- Invoke the prior *invoked (not closed)* display, which is the most recent entry in the Display History dialog box (**Ctrl+Alt+P**), and
- Show/hide the standard workspace control panel (**Ctrl+Alt+W**).

5.1.15 Infinite

A value that can be configured for the MAXWINDOWS window specification property. If MAXWINDOW = *infinite* in a window specification, this specification can be simultaneously used to manage an unlimited number of actual displays.

5.1.16 Keyword

A reserved word in the Workspace Definition Language (WDL). Keywords describe specific kinds of information involved in configuration workspaces (.wdl files).

5.1.17 Lock button

An indicator button in the title area of round robin grouped windows that allows the operator to exempt a window from the round robin automatic display replacement algorithm. The operator cannot lock all of the windows in a round robin group; if only one window remains unlocked, its lock button is unavailable (dimmed).

5.1.18 Manual select group

A kind of window group in which the window of output focus is determined only by specifically selecting the window's option focus button; this window represents the output focus (where the next application display will be presented) until the operator manually selects a different window. It is important to note that a workspace can consist of multiple window groups, each of which is typically intended to contain a related set of application displays. Each window group can have its own window of output focus. Thus, a workspace with multiple workspace window groups will have multiple windows of output focus. See also ““Global output focus” on page 25”

5.1.19 Match expression

An expression that is evaluated by the SafeView at run time to determine if an associated workspace group/window specification should be used to manage an application display. A MATCH expression can express any combination of the following kinds of values: *module* (file path), *window title*, and *window category*.

5.1.20 MAXWINDOWS

A property available to window specifications only for first match window groups. This property determines whether the window specification can be simultaneously applied to an unlimited set of application windows or just one application window. This property is normally 1, but can be set to *infinite*. Such a setting is useful when, for example, a single window specification is configured to “catch all” miscellaneous applications that are not specifically managed elsewhere in a workspace configuration.

5.1.21 Module

The file name and path of an executable file name. An application's module is one of the three means by which an application's display window can be “matched” and thus managed by SafeView. The other two are *window title* and *window category*.

5.1.22 Native window

The Native Window is an application that runs in a Windows environment and provides a display window for a user interface to the LCN. The Native Window provides all of the functionality of the Universal Station,

including the operating and engineering functions. The Native Window is the only means by which standard LCN displays, such as the classic “group display,” can be viewed and managed by SafeView.

5.1.23 Output focus

A concept involving the automatic replacement (closure) of an existing application display by a new application display. When an existing display “has output focus” this means that it will be replaced by the next, newly invoked application display. In multiple-group workspaces, each round robin and manual select window group has its own *output focus* that determines which window in these groups is replaced when new displays are managed in these groups. See also ““Global output focus” on page 25.”

5.1.24 Output focus button

A button on the title bar of all managed application windows and placeholders (appears in the form of an “>”) that indicates whether or not a window currently has *output focus* status. You click this button to give output focus status to the window. If one window has output focus and you select another window in the window's group for output focus, the previous window's output focus button is automatically deselected.

5.1.25 Placeholder

A blank window associated with a particular window specification, created by SafeView at run time. It represents the actual application display window had one been managed according to that window specification. A placeholder represents a window specification or ViewPort when there is an actual application on display. Each ViewPort can be configured to either have a placeholder, or not have one.

When no application is being managed by using a given window specification and the placeholder option is specified, the placeholder defined in the window's specifications is displayed. It is removed from display when a real application display is associated with the window specification. The placeholder can be used to “manually select” where the next application window associated with a manual select group will be displayed. See also ““The placeholder concept” on page 227.”

5.1.26 Prior display

The display name just below the currently highlighted display name as listed in the **Display History** list. Typically, since the most recently invoked (*not* closed) display name is highlighted at the top of this list, then the “prior” display name is defined as the one just below it. However, if you click the **Invoke** button on the **Display History** dialog box or press the **Ctrl+Alt+P** hot keys, then the highlight automatically moves down one position, re-invoking that display. Thus, a sequence of these “prior display” commands will invoke previous displays in reverse order from that in which they were invoked. For example, if you invoke displays A, B, C, D and E, then, consecutive prior commands will invoke displays D, C, B, and A in that order. Invoking any display by any other means will return the highlight to the top of the history list. Only registered application displays can be invoked. Note that the ““List of API calls” on page 118” include “prior” and “next” calls that can control and leverage the selected (currently highlighted) display in the list.

5.1.27 Regular expression

A technique for describing lexical string patterns; a string-expression that can contain “wildcard” characters, so that it can match a range of actual strings. Example: the regular expression “Unit1?*” would match the string “Unit1 Schemes” as well as “Unit1 Overviews.” Regular expressions allow MATCH expressions to specify a range of values without enumerating each value.

5.1.28 Registered application display

See ““SafeView-aware” on page 28.”

5.1.29 Round robin group

A kind of window group in which window specifications are applied (that is, new application displays are presented) in a “least recently used” fashion, for application windows that match the group's MATCH expression. This typically results in new application windows replacing old ones, in a least-recent manner. This group can be thought of as a “round robin” style window group, if one positions the windows in a clockwise fashion.

5.1.30 SafeView-aware

SafeView-aware displays are those that register a category with SafeView, thus allowing enhanced SafeView management, such as the ability to differentiate between different displays presented by the same application. Without this category information, SafeView can only differentiate displays by window title and/or process executable name.

Optionally, registration may also include sufficient information to allow re-invocation of displays by SafeView. These displays are then included in the SafeView History List, and are available for re-invocation through any of the following means:

- The SafeView Prior Display option (**Ctrl+Alt+P**),
- The SafeView History List (**Ctrl+Alt+H**), or
- The Honeywell PRIOR DISP key on an IKB or OEP keyboard.

The Honeywell Native Window is SafeView-aware *and* SafeView-historized by default.

Honeywell displays that register a category with SafeView include the following:

- Native Window (category is NativeWindow)
- GUS .pct displays
 - User-configurable category through GUS Display Builder
 - Default category is “testcategory”
- Experion PKS dsp and .hmi displays
 - User-configurable category through Display Builder and HMIWeb Display Builder
 - Available categories include “HW_System_Trend,” “HW_System_Detail,” “HW_System_Group,” “HW_System_Alarm_Summary,” “HW_System_Faceplate,” and “HW_System_Display.” For default categories, refer to the *Experion PKS Server and Client Configuration Guide*.

Note: If the category is empty, the display does not register a category and is not managed by SafeView. However, if the category is “NOPRIOR,” the display is managed by SafeView without appearing in the history or being invoked by the Prior Display.

To create SafeView-aware applications, refer to ““Application Program Interface” on page 113.”

5.1.31 System button

A term used to denote any of the standard buttons provided by the operating system in the upper-right corner of top-level displays. These include Minimize, Maximize/Restore, and Close buttons.

5.1.32 Tabbed Displays

With Experion R410, Tabbed Display is an optional Server Wide Settings that allows you to have multiple displays available as a tabs within a Station window. You can enable tabbed displays in a station in server wide settings. In addition, you can configure one or more windows in Safeview to be a tabbed window. With tabbed displays, you can choose to open a new display either in an existing tab or in a new tab. When more than one display tab is available, click the tab you want to view, to navigate between displays.

Server Wide Settings

General | **OPC Options** | Security | Summary Displays | IKB/OEP Settings

Startup page
Default system start up page: ⓘ

Tabbed displays
☒ Enable tabbed displays
Reconnect station after enable/disable tabbed displays

Timeouts
Select timeout (sec): ⓘ ☒ Apply select timeout to faceplates
Idle timeout (sec):

Operator actions
Slow raise/lower step (%):
Fast raise/lower step (%):

Point value error indication
☐ Display as question marks
☒ Display using inverse video
☒ Show message for invalid references in displays

Faceplate options
☐ Automatically select most appropriate parameter
☐ Show values of intermediate tick marks
☐ Enable preferred SP

Callouts
☒ Enable callouts

Display scripting
☒ Perform device read after a write

Control confirmation
☐ Default control confirmation response is NO
☐ Enable mode change confirmation via keyboard

Load and performance measurement
☐ Enable display call up events ⓘ
☐ Enable load measurement parameters
Controller measures filter: ⓘ

Last reset time: 12:00:00 AM

History assignment
☒ Allow DSA points in history assignment

Note that the tabbed display option is applicable only for HMIWeb and DSP displays. If a tabbed display is enabled for windows other than SafeView such as GUS display, Native Window, Word, Excel and so on, the first time one of the displays is invoked, the window containing the tabbed displays will be closed to make room for the non-Station display.

5.1.33 Text Editor

A simple, MDI Unicode text-based editor program (TextEdit) used to create and edit workspace configuration files. It includes an option to verify correct workspace syntax. Although most of the functionality provided by the Text Editor has been duplicated by the new SafeView Graphical Workspace Editor (GWE) application, the Text Editor has been retained to provide the functionality that is not provided by the GWE.

5.1.34 Themes

When running on Windows XP, SafeView responds to themes. A theme is a set of user interface characteristics that the operating system applies to windows throughout the system. Microsoft provides Windows XP with default standard themes including “Classic Windows” and “Windows XP”.

5.1.35 ViewPort

A user-visible rendering of a window specification. A paradigm essential to understanding and communicating several of the features and concepts existing in SafeView. A ViewPort is a “sticky window” in which multiple applications can be consecutively displayed. It lends a sense of continuity, different than standard Windows, to display windows. In SafeView, the terms “managed window” and “ViewPort” are used interchangeably.

For the operator, a ViewPort is a static window that can exist for a very long time. Over a period of time, it can hold a number of different displays (one at a time). If configured appropriately, a ViewPort can be repositioned and resized; subsequent applications presented in this ViewPort would then be presented in this new position and the new size.

5.1.36 Window

A rectangular area on a computer display in which application displays can be presented. In SafeView, the terms managed window and ViewPort are used interchangeably.

5.1.37 Window category

A user-definable attribute that a workspace client can assign to its application window(s). This attribute provides a high degree of flexibility in managing such windows, because these windows can be differentiated by category and managed accordingly. Used by SafeView at run time to associate a configured window specification with a real application window, at which time the application window is managed accordingly. Also see ““SafeView-aware” on page 28.”



Attention

When using Station in multi-window mode, individual Station displays should be managed by SafeView according to each display's configured SafeView category. Station displays cannot be managed “by title” even though SafeView generally supports management by title. The reason for this limitation is that Station displays are presented and managed before all of the details regarding the display (including the title) are fully loaded.

Standard Station displays each have a built-in and predefined SafeView category, such as *HW_System_Detail*, *HW_System_Trend*, *HW_System_Display*, and so on. A SafeView category should be configured for each custom Station display as well. Any string value can be used as a SafeView category, though users are encouraged to devise a set of category values that adequately meets their needs for display classification. This classification “by category” forms the basis of a well-constructed SafeView workspace configuration, enabling displays to be properly differentiated and positioned as they are invoked by operators.

To configure a display category for a custom Experion Station display, use the HMIWeb display builder. The properties dialog for the page includes a “Details” tab with an editable combobox that can be used for selecting an existing built-in category, or to type any desired category value.

5.1.38 Window group

A workspace configuration must contain one or more window groups. There are three kinds of groups or “group behaviors:” first match, round robin, and manual select.

A workspace configuration that differentiates windows into more than one group should be composed of a top-level and one or more subgroups. Because only first match groups can have subgroups, if you want your workspace to have subgroups, the top-level group *must be* configured as a first match group. Subgroups can be defined as either round robin or manual select subgroups. A workspace that does not differentiate windows into separate groups should contain only a top-level group, typically round robin or manual select. The following provides a description of the group items.

| | |
|----------------------------|--|
| <i>window_group:</i> | GROUP <i>group_name</i> [<i>group_description</i>] '('group_behavior ')' { <i>window_group_item</i> } END GROUP |
| <i>group_name:</i> | <i>Identifier</i> |
| <i>group_description</i> | description '=' <i>string</i> ',' |
| <i>group_behavior:</i> | <i>String</i> DEFAULT '=' <i>boolean_expression</i> ; MATCH '=' <i>match_expression</i> ; REGION '=' <i>windowbox</i> ',' |
| <i>window_group_item :</i> | <i>window_specification</i> <i>window_group</i> |

group_name

The name of this group.

group_description

The description field can be used to hold helpful information about this group. The text here does not affect SafeView management of displays. One Description entry is allowed for each Workspace, Group, and Window object. The Description content must be a single long text string (not multiple lines and no embedded quotes).

Description example:

```
description="This is my description.";
```

group_behavior

This property identifies the group as first match, round robin, or manual select.

DEFAULT = true/false

This property (set *true*) can appear only once in the entire configuration. If a registered window has a category that fails to match any category in the workspace configuration, the window, when displayed, is managed as if it matched the group or window specification containing this default.

MATCH = match expression

This expression (not a property) defines the logic SafeView uses to determine if a given window should be managed in this window group (see “Workspace - Match expressions” on page 39” for details).

REGION = x, y, width, height

Managed windows cannot be dragged, sized, or maximized manually or automatically, such that they extend beyond this REGION property. When specified at the group level, this property is inherited by all window specifications in this group, though specific window specifications can override this property.

Window group items

The window specification holds the actual information for managing (or constraining) the display of a managed window. Essentially, these are the SafeView properties that extend standard Windows functionality. A wide range of control is available for configuring and controlling displays for a particular end user. If care is not taken, *you might improperly design and configure a workspace such that the end-user's displays are so constrained as to be unusable.* The following provides a description of the window items.

| | |
|-------------------------------|--|
| <i>window_specification :</i> | WINDOW <i>window_spec_name</i> [<i>description</i>] { <i>window_property</i> } END WINDOW |
| <i>window_spec_name:</i> | <i>identifier</i> |
| <i>window_description</i> | description '=' <i>string</i> ',' |
| <i>window_property :</i> | ALWAYS ON TOP '=' <i>boolean_expression</i> ',' CLOSABLE '=' <i>boolean_expression</i> ',' DEFAULT '=' <i>boolean_expression</i> ',' DRAGGABLE '=' <i>boolean_expression</i> ',' GLOBALFOCUS '=' <i>boolean_expression</i> ',' MATCH '=' <i>match_expression</i> ',' MAXPOS '=' <i>point</i> ',' MAXSIZE '=' <i>size</i> ',' MAXWINDOWS '=' <i>expression</i> ',' MINIMIZABLE '=' <i>boolean_expression</i> ',' MINSIZE '=' <i>size</i> ',' PLACEHOLDER '=' <i>boolean_expression</i> ',' POSITION '=' <i>point</i> ',' REGION '=' <i>windowbox</i> ',' SIZE '=' <i>size</i> ',' SIZEABLE '=' <i>boolean_expression</i> ',' |

window_spec_name

This is the name presented in the placeholder if no application is being managed by this window specification. This name can be used in API calls to reference a particular window specification directly. This name must be unique within a window group. If window spec names are going to be used in API calls, it is a good idea to keep their names unique across the entire workspace.

window_description

The description field can be used to hold helpful information about this window. The text here does not affect SafeView management of displays. One Description entry is allowed for each Workspace, Group, and Window object. The Description content must be a single long text string (not multiple lines, and no embedded quotes).

Description example:

```
description="This is my description.";
```

window_property

ALWAYS ON TOP = true/false

This property (set to *true*) assigns the Windows “top-most” style to a window so that a window that does not have the “topmost” style can never overlay it. It can, however, be overlaid by another “top-most” window. To construct a workspace that you do not want to be overlaid, design it so that all application displays are managed and constrain non-critical displays to a region other than the critical area.

**Tip**

SafeView does not prevent applications from resetting the ALWAYS ON TOP property.

CLOSABLE = true/false

This property (set to *false*) protects a window from being closed directly. Setting CLOSABLE to *false* disables the system “Close command” for the application. The LCN Native Window is intelligent enough to disable the

File > Exit command when managed in a window configured with CLOSABLE set to *false*. This protects against direct closure of the Native Window (see GLOBALFOCUS below). Since some applications do not have a menu, **File > Close** is not a problem. However, it should be noted that other applications will only disable the system Close command and *not* their **File > Exit** (and others) disabled. Be careful using this option.

You should use it primarily to protect critical displays from being closed directly (default = standard windows behavior, that is, close is allowed).

DEFAULT = true/false

This property (set to *true*) allows you to designate the MATCH expression for a window as the default MATCH expression for all registered windows (workspace client displays associated with a given category using the “RegisterWindowData” or “CreateWorkspaceClient” API call) that do not match their workspace configuration. DEFAULT = *true* can appear only once in a workspace configuration file.

DRAGGABLE = true/false

This property (set to *false*) enables you to disable the repositioning of a window by not allowing a user to drag the window. A nondraggable window can be “sized” and thus moved by stretching or shrinking. Consider making nondraggable windows also non-sizeable by setting SIZEABLE to *false*. Another way to constrain window positioning is by using the REGION property.

GLOBALFOCUS = true/false

This property (set to *false*) configures a window such that the global focus button is not available for that window. This protects an operator from replacing a critical application with another managed application. Using this option in conjunction with the CLOSABLE = *false* option effectively disables the operator's ability to close a critical application. Using these in conjunction with the MINIMIZE = *false* option assures that important operational displays are not closed, *and* not iconized. In addition, using the ALWAYS ON TOP property, and constraining other applications from being always on top and limiting their regions, provides a high degree of protection for critical applications.

Workspace windows in a single-group round robin or manual select workspace have, by default, redundant output focus and global focus buttons. By setting GLOBALFOCUS = *false* for these windows, the redundant focus buttons are removed.

MATCH = match expression

This is an expression, rather than a property that defines the logic SafeView uses to determine if a window should be managed in this window group. This property is only valid for window specifications directly owned by a first match group. For more details, see ““Workspace - Match expressions” on page 39.” An example when this might be used is to “catch all” displays not explicitly matched elsewhere.

MAXPOS = x, y

This property specifies the maximized position (upper left *x* and *y* coordinates) of a maximized window.

MAXSIZE = width, height

This property specifies the maximum width and height allowed for a window. This property stops “growing” a window if you attempt to size the window beyond either the *x* or *y* parameter. Also, if the user maximizes the window, this property constrains the window to the MAXSIZE dimensions.

MAXWINDOWS = 1/infinite

This property specifies whether a window specification has a 1:1 (the default) relationship with managed windows (as is the case for round robin and manual select groups) or if it has a one-to-many relationship with managed windows. This property is applicable *only* for window specifications in first match window groups.

It is generally preferable to leave MAXWINDOWS set to 1 when the window specification includes a number of properties such as POSITION or SIZE. This will ensure a one-to-one correspondence between each display and its associated window specification.

If MAXWINDOWS is set to “infinite” in a window specification, any number of displays can be managed by this single window specification. A single window specification can be used to manage many windows simultaneously. This is helpful when you wish to “mildly constrain” an indeterminate number of windows that share one or more properties, such as a similar constraining region. Application displays managed with

MAXWINDOWS = *Infinite* are never closed by SafeView unless an application window has global output focus, in which case it is replaced by the next SafeView display invoked. They can, however, be closed in normal Windows fashion.

MINIMIZABLE = *true/false*

This property (set to *false*) disables the ability to minimize (iconize) a window. When it is set equal to *true*, the Minimize function has no effect. An example of where this property can be used is for critical alarm displays. These displays could be configured so that they are “always on top,” located in a region that excludes less important displays, set up with an adequate minimum size and with MINIMIZABLE set to *false* so they cannot be minimized. This is normally how this function is used. Also note that while a maximizable property is not currently available, including “MAXIMIZABLE = *true/false*” will not produce a syntax error.

MINSIZE = *width, height*

This property specifies the minimum size for a window. It prevents the user from sizing the window below the given dimensions. Note that iconization is not affected by this property.

PLACEHOLDER = *true/false*

This property indicates whether a “placeholder” should be displayed, according to this window specification's properties, when no application is currently displayed using this window specification. This provides a means to display the workspace “structure” when a full complement of application displays remains to be managed. The SafeView Control Panel contains a button (and an API) that allows placeholders to be either hidden or shown. Only the placeholders that are set *true* in their window specifications are ever shown.

POSITION = *x, y*

This property specifies the initial position, (x and y upper left window coordinates) of a window. It is constrained by the REGION property. If POSITION is not configured, the actual position is determined by the Windows operating system. If POSITION is configured and SIZE is not configured, the system will sacrifice the size before sacrificing the position in conforming to a specified REGION property. If both POSITION and SIZE are configured, the syntax checker verifies that the window fits into a specified region. If the “return to original position” command (Reposition) is executed, SafeView will move the window back to this position if that was where the window was initially displayed.

REGION = *x, y, width, height*

This property specifies the area within which a window can be displayed. This property can be inherited from round robin and manual select group, if it is configured at these group levels. If it is configured here, at the window specification level, it will override the window group's region specification. Managed windows cannot be dragged, sized, or maximized manually or automatically, such that they extend beyond this region. This property nicely constrains a maximize-window, for example, to a given area that does not necessarily cover the entire display region. It also provides some functionality that at first may seem surprising; namely, that a window can be dragged only so far and then it stops, for no apparent reason. A good design principle is to configure each window group (excluding first match) to have a prescribed region; that is, each window specification in the group should have a similar REGION attribute.

SIZE = *width, height*

This property specifies the initial width and height of a window. It is constrained by the REGION property. If SIZE is not configured, the Windows operating system determines it. If SIZE is configured and POSITION is not configured, the system will sacrifice position before sacrificing size in conforming to a specified REGION property. If the Reposition (“return to original position”) command is executed, SafeView will restore the window to its original size.

SIZEABLE = *true/false*

This property indicates whether a user can size a window. When SIZEABLE = *false*, the window style is modified at initial display time such that the window has a “thin-frame.” This provides the visual cue that the window cannot be sized. The normal Window's Minimize and Maximize functions are still available. It should be noted that even though some applications redisplay their original (thick) frame and appear to be sizable, they are not.

5.1.39 Window management

Window Management is the control of the display characteristics (for example, POSITION and STYLE) of application windows. This is, essentially, workspace management.

5.1.40 Window of output focus

See “ “Output focus” on page 27.”

5.1.41 Window specification

The window specification is a set of detailed information that comprises the set of rules by which an application window is managed, or constrained, by the SafeView software. A workspace configuration typically contains many different window specifications, so that many different windows can be simultaneously managed, each according to its own unique criteria. Each window specification can contain information relating to the management of display windows. This information is considered in terms of a window property *keyword* and a description of the value associated with the property described by the keyword. The specific syntax of window specifications and all other workspace constructs is defined “Appendix A: Workspace Definition Language.” Generally, the syntax is

```
WINDOW window-specification-name
window property keyword = value;
window property keyword = value;
.
.
END WINDOW
```

5.1.42 Window title

For purposes of understanding SafeView, a window title is the text that appears in the “title area” of a top-level, application display window.

5.1.43 Windows XP

The version of Windows XP that SafeView is qualified on is Windows XP Professional. For purposes of this document, XP always refers to the Professional version.

5.1.44 Windows 7

With Experion R400, SafeView is qualified on Windows 7 and Windows 2008 Server.

5.1.45 Workspace

An operator's working user-interface environment. Microsoft Windows provides a basic workspace. The SafeView software extends the capabilities of that workspace.

5.1.46 Workspace configuration

Refer to the “Window specification” on page 35 .

5.1.47 Workspace Definition Language (WDL)

The Workspace Definition Language describes the workspace window management as defined in “Appendix A: Workspace Definition Language” on page 217.”

5.2 Workspace organization

5.2.1 Workspace groups

Workspace configurations organize window specifications into logical *window groups*. Each workspace configuration consists of a single top-level group consisting of one or more window specifications and/or subgroups. Thus, a hierarchy of window groupings is possible.

Each window group must be configured with one of three possible strategies for searching for matching window specifications: **Round Robin Group**, **First Match Group**, and **Manual Select Group**

5.2.2 Workspace - Round robin group

This kind of group uses a “round robin” approach to selecting which of its window specifications is applied to a given application window. This group supports a cyclical window replacement style, in which the newest application windows replace the oldest ones. All window specifications in this type of group share a common MATCH expression (described in “Workspace - Match expressions” on page 39”) so that a similar set of applications can equally match to each window in the group. The MATCH expression concept is applied at the *group level* for round robin groups.

In the following figure, after the display in Window1 is replaced, the display in Window2 will be designated as the next one to be replaced by a new application display, and so on.

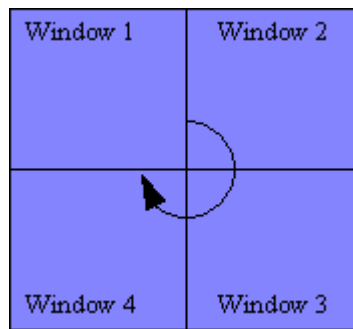


Figure 1: Example: Round Robin Group Arrangement

Notes:

- When a match is made with a round robin window group, a “least recent” type of window specification selection is provided such that the least recently used window specification is applied. Thus, the “oldest” pictures are replaced with new displays.
- Managed application displays can be “locked” either by an operator or by a program through selection of the “lock” button in the title area of the display. Locked displays are excluded from the round robin replacement mechanism. Not all displays can be locked, or new displays couldn't be shown.
- An “output focus” indicator button allows the operator to see which display will be replaced by a newly invoked display. The operator can reset this indicator to any window in this group, by selecting this indicator button in the title area of any display in this group.
- The *match* language construct is required with the definition of a round robin group. Optionally, the *default* language construct can be associated with the MATCH expression, indicating that if a registered application display fails to match any window group, the MATCH expression is considered to apply and this group will be used to manage the display.
- This group requires a 1:1 correspondence between an application and a window specification to support unique window positions for each window in a round robin group; therefore, the MAXWINDOWS property is *not* valid for round robin groups.

- Newly invoked applications replace exiting displays in the current positions of the exiting displays. Thus, if the current output focus window has been moved or sized, when it is replaced, the new display appears in the latest position of the replaced window. Also, if the output focus window (ViewPort) is maximized at the time it is replaced, the new display is likewise maximized.
- By design, this group does not support subgroups.

5.2.3 Workspace - Manual select group

The manual select group is similar to the round robin group in that all applications displayed in this group share a common MATCH expression. This group, however, does not automatically determine which window specification to use to manage the next application display that matches the group's MATCH expression. Rather, the user manually selects where the next application display will appear. This is done by selecting an existing application display window (or a *placeholder*) as the *output focus* window, using the indicator button provided by SafeView in the window's title area. This window's specification is then used to manage the next display associated with this window group. Because the manual select group emphasizes this 1:1 correspondence between an application and a window specification, the MAXWINDOWS property is not valid for window specifications for manual select groups.

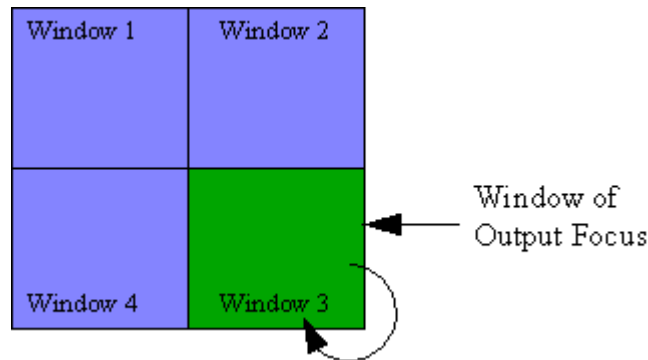


Figure 2: Manual Select Group Arrangement

Notes:

- New applications continue to replace the current window with output focus until the operator selects another window to be the output focus window.
- As in round robin groups, if the current output focus window has been moved or sized, when it is replaced, the new display appears in the latest position of the replaced window. Also, if the output focus window is maximized at the time it is replaced, the new display is likewise maximized.
- This group requires a one-to-one correspondence between an application and a window specification to support the manual select window-replacement strategy provided by this type of window group; therefore, the MAXWINDOWS property is *not* valid for round robin groups.
- By design, this group does not support subgroups.

5.2.4 Workspace - First match group

The first match group searches its list of subgroups and window specifications in physical order (as entered during the creation of the workspace configuration) to find a match with a given application window, as that window is about to be displayed. The first-found MATCH expression that matches the window's criteria (window title, module/filename, or category) is used to manage the window.

It makes no sense to include two items with similar MATCH expressions in a first match group, because the first match found will always “win.” Each item in this group, therefore, including individual window specifications, has its own MATCH expression.

In the figure below, a first match group allows the workspace to search for the “correct subgroup” for a given application; then, the application is placed into this subgroup in a round robin fashion.

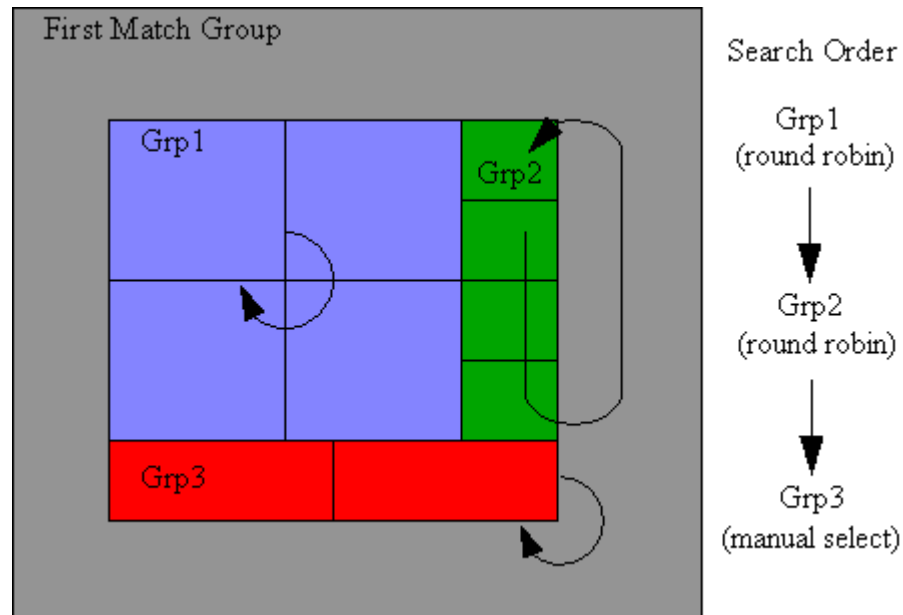


Figure 3: First Match Group Arrangement

Notes:

- This group's window specifications must each have its own MATCH expressions.
- It is appropriate for each MATCH expression in this group not to intersect in any way with other MATCH expressions, because only the first satisfied expression is always used.
- This kind of group is the only one that can have subgroups; thus, any workspace configuration that arranges windows into more than one group *must* have a first match group as the top-level group.
- This type of group can be best used to organize other subgroups.
- This group can appropriately contain a “catch all” window specification (MAXWINDOWS set to *infinite*), set up so that all miscellaneous applications can at least be minimally managed, if desired. This would be appropriate in cases where not all applications available to operators are known at the time a given workspace configuration is being built.

5.2.5 Workspace - Match expressions

Intelligent workspace display management is accomplished by intelligently organizing different kinds of application displays into different logical window groups. These window groups, in turn, contain a number of window specifications that share some common features, such as display geography. Thus, a workspace should typically be configured such that similar kinds of displays are presented in, and even limited to, specific regions on the display surface. The result is that the system intelligently positions all of the myriad application displays invoked throughout the shift, based upon display content. The means by which different kinds of displays are differentiated by SafeView is the MATCH expression. SafeView manages application windows in real time *only* when these windows are successfully matched with a MATCH expression in the workspace configuration.

5.2.6 When no match is intended

Although perhaps unlikely, you can configure a window or window group without a MATCH expression. The SafeView Text Editor will not flag the absence of a MATCH expression as a syntax error. SafeView would not

select a window or group configured without a MATCH expression for displaying applications except in the following two cases:

- When the DEFAULT keyword is defined for the window or group and is set to *true*, in which case SafeView would select this window or group as the match for category-registered applications that have categories unknown to the currently loaded SafeView workspace configuration file, and
- When the global output focus button for the window ViewPort is pressed only if that application's display had a match somewhere in the workspace. For registered displays, a match would occur if there was a DEFAULT set to *true* somewhere in the workspace. For any application, a match would occur if there were a wildcard somewhere in the workspace.

5.2.7 A MATCH expression is a logical expression

A MATCH expression is essentially a logical expression that ultimately evaluates to *true* or *false*. This evaluation is made by SafeView each time an application window is about to be displayed. The logical expression is any combination of three kinds of values, associated at run time with the application window that is about to be displayed:

- **Title:** The text in the title area of the application's display window
- **Module:** The complete pathname of the executable associated with the application
- **Category:** A user-defined string value that is registered by a workspace client. A category value is associated with a display through the display's "category" property in the display builder application. The workspace client instructs SafeView to associate a given window category string with a given application display handle, as described in "Application Program Interface."

A MATCH expression typically contains a logical combination of titles, modules, and/or categories, as the following examples demonstrate:

`match = title("Unit100") or title("Unit200") or module("?*winword.exe?*") or category("Schematics")`

`match = category("Overhead Displays") or category("Misc. Displays")`

5.2.8 Workspace - Wildcards

In SafeView, a wildcard is defined as two consecutive characters, a question mark followed by an asterisk (?*). When you want to use a wildcard in a module expression, it is recommended that you use a wildcard at both the beginning and end. For example, use ("?*winword.exe?*"), not ("?*winword.exe"). For more information, see "Using wildcard character-matching" on page 271."

5.2.9 Workspace - Group matching

Match expressions exist at the group level for round robin and manual select groups because these two groups are designed to contain similar kinds of application displays. Application displays are deemed "similar" if they match the same MATCH expression. Conversely, the first match group is intended to contain different kinds of groups and/or window specifications. The first match group, therefore, does not itself own a MATCH expression, but any window specification directly owned by a first match group will have a MATCH expression.

Round robin group matching

If a display matches a MATCH expression that is part of a round robin group, the next (output focus) round robin window specification in that group is associated with the next application display invoked.

Manual select group matching

If a display matches a MATCH expression that is part of a manual select group, the manually selected window in that group is replaced by the next application display invoked.

First match group matching

If a display matches a MATCH expression that is directly associated with a window specification in a first match group, the window specification is used to manage that display.

When no match is found

If no match is found, two things can occur:

- If the workspace is configured with a default MATCH expression and the application display has been registered (that is, the application is a *workspace client*), the display will be managed as if it matched the default MATCH expression.
- Otherwise, SafeView does not manage the application's display. Such a display is presented according to the display parameters given by the Windows operation system as if SafeView was not present.

5.2.10 Regular expressions make matching easier

Match expressions essentially pattern-match on strings; the module, window title, and window category are all strings. So that the engineer needn't explicitly enumerate every possible matching string, SafeView supports *regular expression* string-pattern matching. Regular expressions are a means of utilizing “wildcard values, so that a single regular expression string can match on a range of strings, without the need to explicitly enumerate each one. Example:

```
match = module(“?*myApp.exe”) or title(“Micro?*)” or title(“File Manager”) or category(“picture”)
```

This expression would match any application window whose file path ends in “myApp.exe,” whose title starts with “Micro,” whose title is “File Manager,” or whose window category is “picture.”

5.2.11 Using the NOT operator

The power of MATCH expressions can be greatly enhanced by using the NOT operator. For example, `match = title(“?*”) and not title(“Program Manager”)` would catch every application except the Program Manager.

5.2.12 Default MATCH expression

The `DEFAULT = true/false` language construct can be associated with, at most, one MATCH expression in a workspace configuration. Only registered, workspace client windows use this mechanism. This *default* option applies in the case where a workspace client fails to match any of the MATCH expressions in the entire configuration. This can occur, for example, when new displays with new categories are utilized within an existing workspace configuration. When a match of a registered window fails, the default MATCH expression, if provided, is forced *true* and the window is managed accordingly.

Note, however, that you can *exempt* a workspace client from SafeView's default category matching function by setting its category value to either the empty string (NULL or “”), or a single blank character (“ ”).

For the DEFAULT expression to be effective when a MATCH expression includes a wildcard, such as `title(“?*”)`, the wildcard expression should be: `match = title(“?*”) and not category(“?*”)`.

5.2.13 Window specification

The window specification is a set of detailed information that comprises the set of rules by which an application window is managed, or constrained, by the SafeView software. A workspace configuration typically contains many different window specifications, so that many different windows can be simultaneously managed, each according to its own unique criteria. Each window specification can contain information relating to the management of display windows. This information is considered in terms of a window property *keyword* and a description of the value associated with the property described by the keyword. The specific syntax of window specifications and all other workspace constructs is defined “Appendix A: Workspace Definition Language.” Generally, the syntax is

```
WINDOW window-specification-name
```

```

window property keyword = value;
window property keyword = value;
.
.
END WINDOW

```

5.2.14 Top-down or focus-based matching

The focus-based search option allows you to specify whether SafeView will, for a given workspace, use its default “top down” matching or the “focus-based” matching algorithm.

With the focus-based search option enabled, SafeView does not search top-down through its workspace main group as it normally would, but rather it searches the group that owns the *currently active window* first. If this group is nested, then SafeView begins with the most deeply nested owning-group, then its owning group, and so on, moving outward in the workspace tree structure toward the outermost group. The “most deeply nested owning group” means the innermost group that contains (or “owns”) the managed window in focus at the time of the new display invocation.

For examples of focus-based matching, see ““Safeview workspace examples” on page 153.”

For *build-time* activation of the focus-based search mode, see ““focusbasedsearch”.”

For *run-time* activation of the focus-based search mode, see ““Changing the SafeView search mode (Normal or Focus)” on page 112.”

5.3 Managed windows

5.3.1 Managed - Indicator buttons

SafeView adds up to four indicator buttons (fast, lock, output focus, global output focus) to the title bar of the managed windows and placeholders when they appear on an operator station. If the window does not have a title bar, it can still be managed, but will appear without the indicator buttons.

5.3.2 Managed - Output focus windows

When invoking a display, it is helpful to know where it will appear. SafeView indicates the next window to be replaced by providing an output focus indicator button in the title area of a managed window, as shown in the following figure.

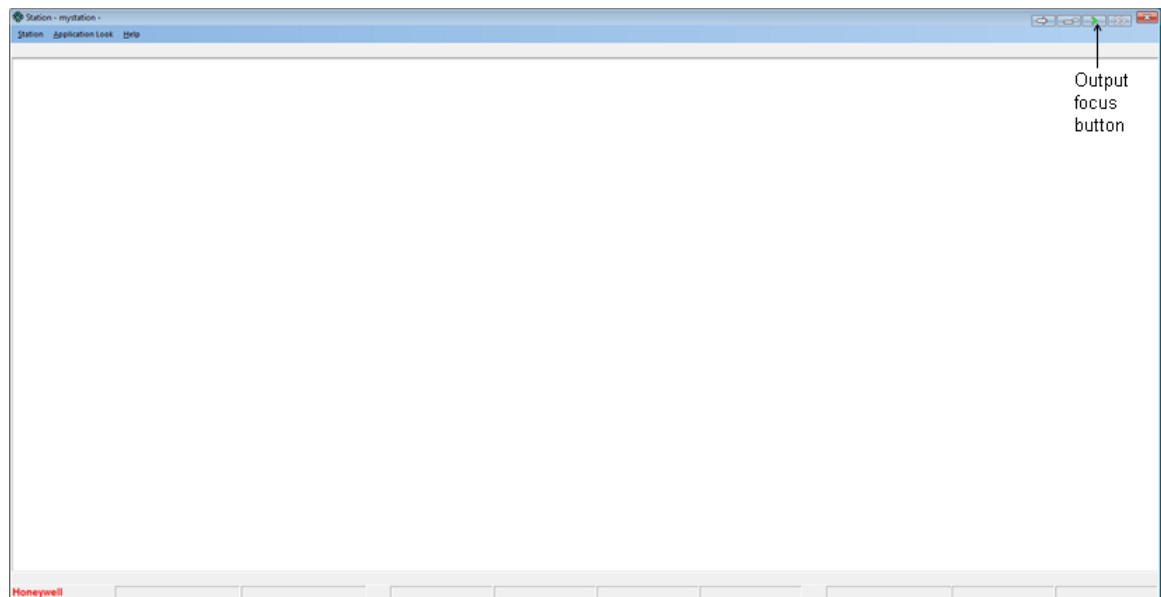


Figure 4: Output Focus Button

This depressed green > button indicates that this is the next window in this group that will be replaced by the next application display to be managed in this group. Such a window is referred to the *output focus* window. The new application is displayed in the same place as the window it is replacing, even if this window has been moved or sized such that it deviates from the position and size configured in the window specification. Thus, the new application appears to the operator to “go into” the output focus window.

5.3.3 Output focus in multiple-group workspaces

In a multiple-group workspace, each round robin and manual select window group maintains its own “local” output focus window. In fact, these groups are named for the manner in which output focus is determined. Multiple-group workspaces, therefore, have multiple output focus windows available at one time, one for each round robin and manual select window group. When a display is invoked, the workspace determines, through the MATCH expression, in which group the window should be displayed. The application is then displayed in this group's current output focus window. The following figure shows a three-group workspace, with three active output focus windows, identified by green output focus buttons.

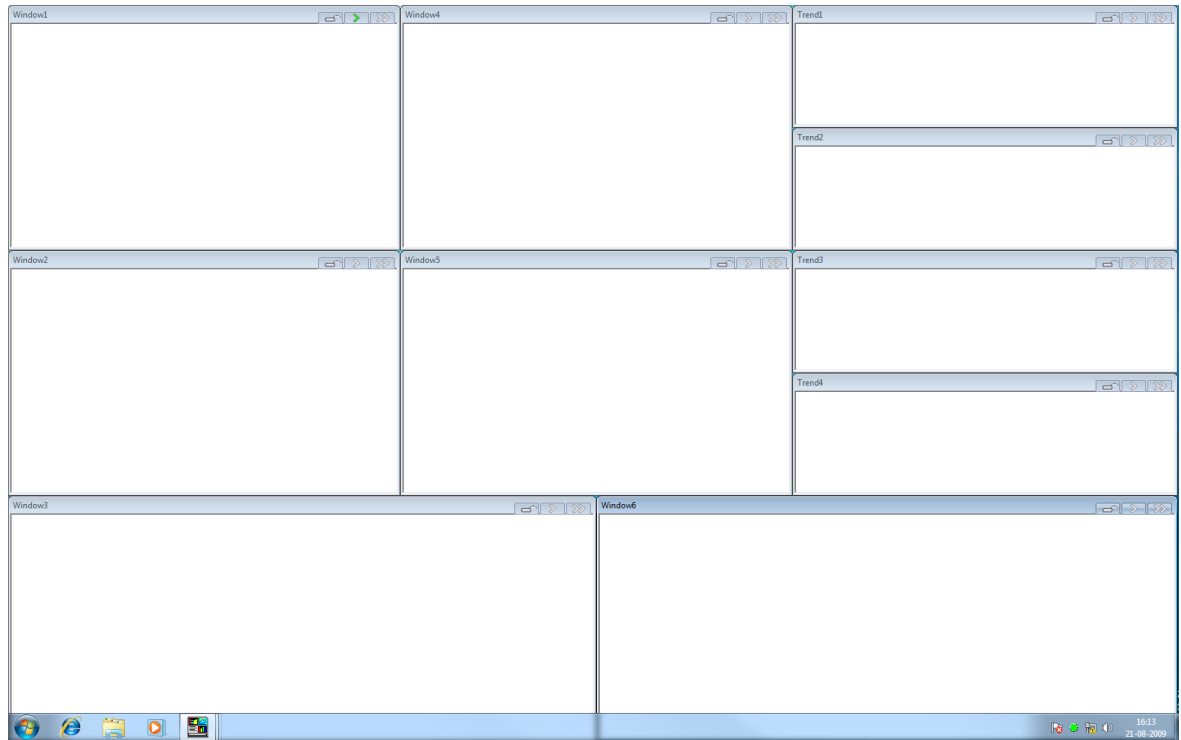


Figure 5: Workspace with one Active Output Focus Window

5.3.4 Managed - Invisible output focus

A depressed indicator button that is provided by SafeView in the existing window's title bar indicates output focus windows.

If this window's specification is configured with `PLACEHOLDER = true`, a placeholder is presented as a “stand-in” until another window is invoked and managed using this window specification. If no placeholder is configured, and if this window is the output focus window, the operator will have an “invisible output focus” window until a display is invoked that utilizes the window specification.

Another possibility for an invisible output focus window is if the “hide placeholders” option has been selected; then no placeholders are displayed, even if configured to be there. This case can be addressed by selecting the “show placeholders” option from the workspace control panel.

In manual select groups, if no placeholders are configured, there will be no way for the operator to select or use windows (other than the first one) as output focus windows. It is therefore a good rule of thumb, especially for manual select groups and round robin groups, to configure `PLACEHOLDER = true` for each window specification. The engineer can also choose not to expose the “hide placeholders” option, if the workspace control panel features are exposed by a custom picture or VB application, rather than the built-in SafeView control panel.



Tip

It is possible for the operator to manually close an application display.

5.3.5 Managed - Global output focus

In workspaces that do not contain multiple window groups, the concept of “output focus” is sufficient to describe and control the replacement of existing displays by new displays. In multiple group workspaces, however, it is desirable to have a workspace-wide focus, or “global output focus.” An operator may want to override a workspace's normal application-association logic and simply want to “put the picture there.” So, in

multiple-group workspaces, SafeView provides a global output focus button in the title area of a managed window, as shown in the following figure.

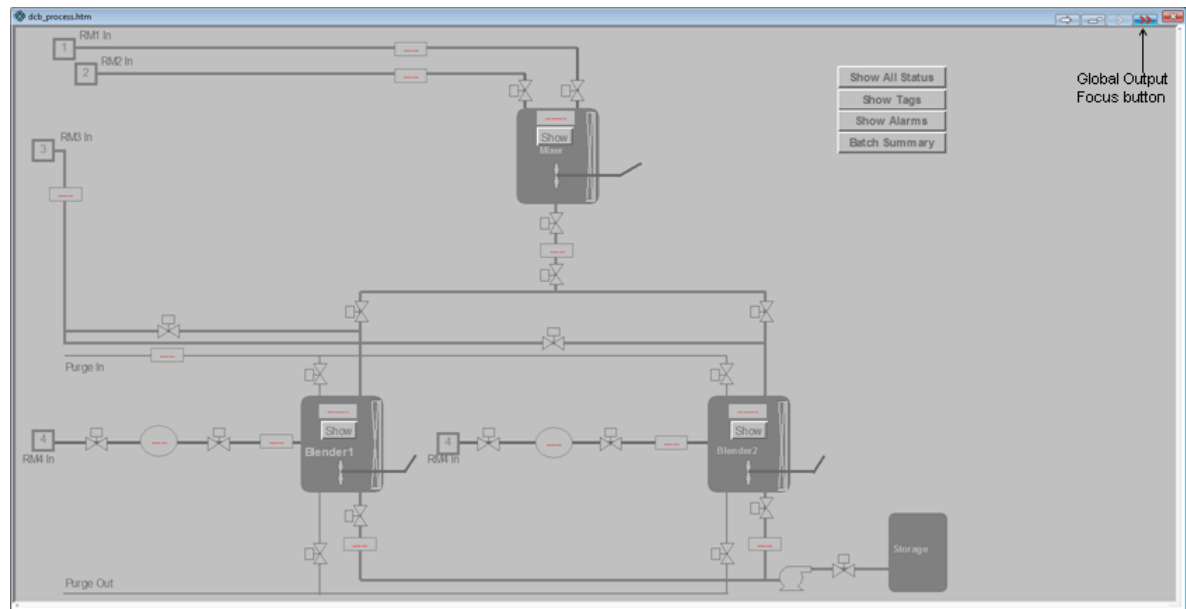


Figure 6: Global Output Focus Button

In the figure, the depressed red button indicates that all “normal” output focus buttons are temporarily unavailable. This button is available for all the managed windows in a multiple group workspace. In the entire workspace, only one window at a time can be the “global output focus” window.

When a window's global output focus button is selected, SafeView will replace this window with the next matching invoked display, regardless of whether or not that display would normally be managed in the global output focus window's group. Using this feature, an operator can choose to display, for example, a trend picture in a window normally used for larger schematics, perhaps to get a larger rendering of the trend than is normally used.



Tip

The window specification property GLOBALFOCUS can be set to *false* to disable (hide) the global output focus button.

5.3.6 Multiple-window Station and tabbed displays

With Experion R410, you can enable tabbed displays on a Flex or Console Station that uses SafeView to manage several windows. With tabbed displays, you can invoke displays as individual tabs within a Station window to enable easier navigation to other tabbed displays.

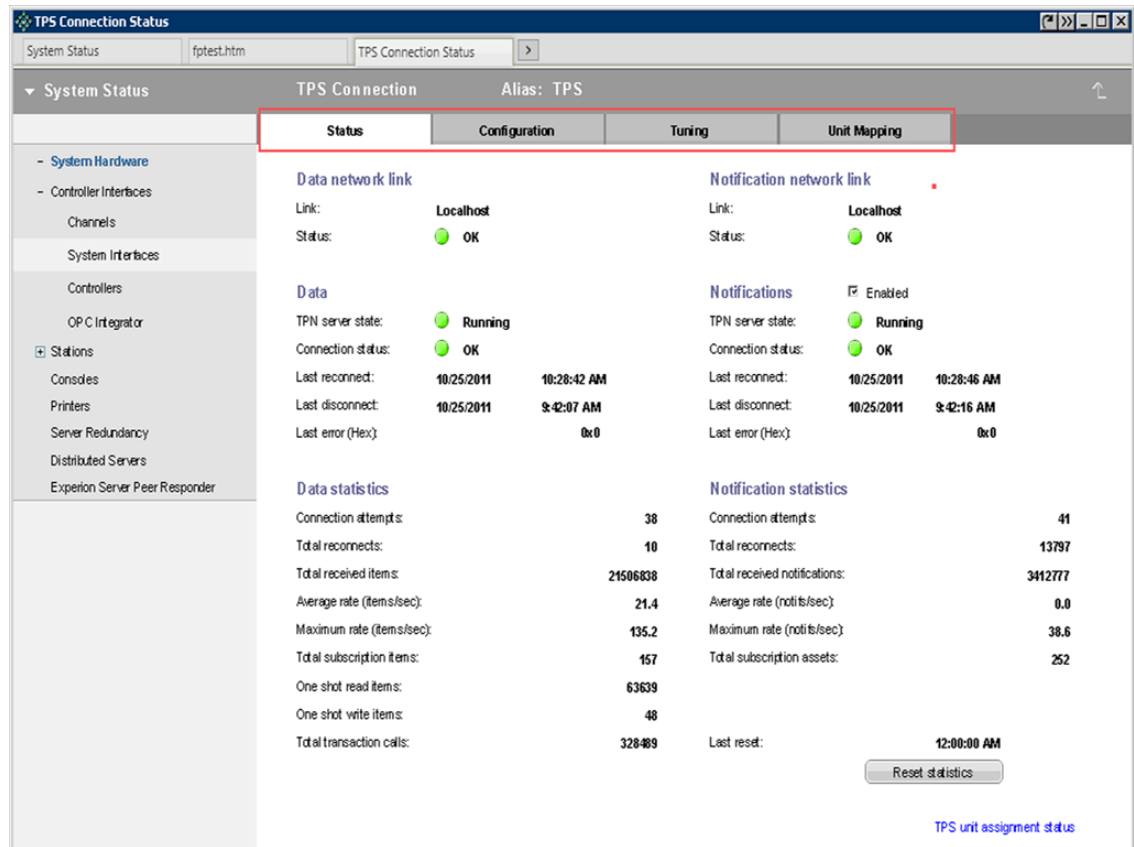
The screenshot displays the Honeywell Safeview alarm management interface. On the left is a 'Location Pane' with a tree view showing the hierarchy: Network > Computers > pkcd > Devices > System Components (1008) > SERVER_232 [local server] (1). Below this are links for 'System Management' and 'FTE Status'. The main area is a table of alarms with columns: Date & Time, Location Tag, Source, Condition, Priority, Description, and Trip V... The table lists 20 alarms, mostly 'Testing license' events from 'SERVER_232' and two 'COMMS U 00' events from 'TPS'. At the bottom left, a summary shows: Unacknowledged alarms: 1008, Acknowledged alarms: 0, Shelved alarms: 0 of 0. At the bottom right are buttons for 'Shelve Alarm', 'Unshelve Alarm', 'Pause', 'Resume', and 'Acknowledge Page'.

| Date & Time | Location Tag | Source | Condition | Priority | Description | Trip V... |
|---------------------|--------------|-----------------|--------------|----------|--|-----------|
| 10/25/2011 14:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 13:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 12:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 11:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 10:26:17 | TPS | TPS | COMMS U 00 | | CStr02: Notifications from TP... UNAV... | |
| 10/25/2011 10:11:09 | TPS | TPS | COMMS U 00 | | CStr02: Data connection to T... UNAV... | |
| 10/25/2011 10:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 9:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 8:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 7:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 6:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 5:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 4:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 3:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 2:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 1:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/25/2011 0:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 23:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 22:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 21:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 20:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 19:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 18:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 17:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |
| 10/24/2011 16:00:00 | SERVER_232 | Testing license | LICENSE H 00 | | License for internal testing - 2... | |

Unacknowledged alarms: 1008
 Acknowledged alarms: 0
 Shelved alarms: 0 of 0

Shelve Alarm ... Unshelve Alarm
 Pause Resume Acknowledge Page

A tabbed display can have its own sub-tabs.



5.3.7 Managed - Locking a window

Recall that in round robin groups, SafeView automatically applies a first-in, first-out approach to replacing old displays with new ones. However, you may want to keep a display and exempt it temporarily from the normal round robin replacement approach. For this purpose, SafeView provides a “lock button” located in the title area of all windows in all round robin window groups, as shown in the following figure.

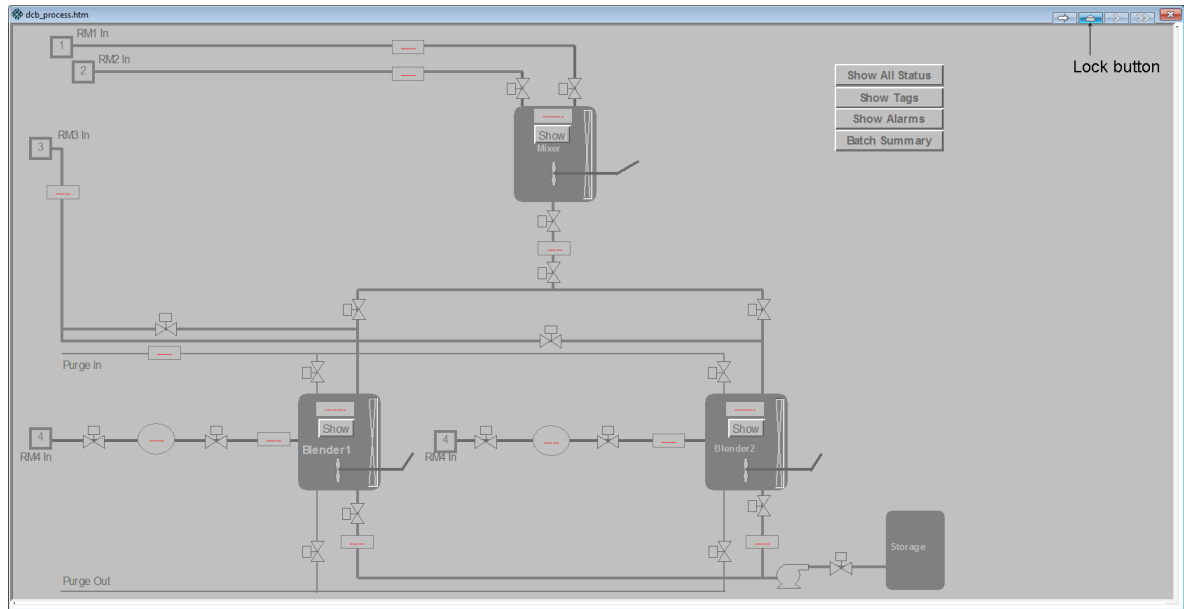


Figure 7: Lock Button

In the figure, the depressed green button indicates that this display is locked. You can lock one or more windows in a round robin window group, thus preventing SafeView from automatically replacing these windows with new displays. For example, in a four-window round robin window group, you may decide to lock the window in the upper left corner. If the window to be locked has the output focus, SafeView will move the output focus to the next available window. Then you can select the window to be locked and from then on, until you explicitly unlock it, the window will be exempted from the normal round robin selection for replacement by new displays.

Not all windows in a group can be locked, or new displays would be prevented from appearing. To enforce this, when all but one window is locked, the remaining unlocked window's lock button is unavailable. Once a window has been unlocked, it becomes eligible for the window's group replacement strategy.

**Tip**

Note that the lock button appears only in round robin windows.

5.3.8 Managed - Fast mode displays

A window and/or a display can be set to fast mode using the SafeView Fast button. This directs the window/display to update the data at a preconfigured fast rate. The Fast mode button is applicable for the following displays:

- GUS displays
- Native Window
- Station displays

The SafeView Fast button is not applicable to other displays like Notepad, third-party displays, and Honeywell provided displays. Therefore, the button does not appear for these kinds of displays.

The animated double arrow on the title bar of the display indicates that the display is in fast mode. The single arrow indicates that the display is in normal mode. If a GUS display or a Station custom display does not have any fast mode parameters, the Fast button is disabled.

The IKB/OEP Fast key or any Station-configured key can be used for requesting the display to be in fast mode. The Fast key LED on the IKB/OEP indicates if the active display is in fast mode. The same can be inferred from the status bar of the Station window.

At a given point in time, only one display can be in fast mode. For instance, assume Window A is in fast mode. If you click the Fast button on Window B, Window B switches to fast mode and Window A switches to normal mode.

6 Designing a workspace

Related topics

“Workspace design” on page 52

“Configuration file format” on page 53

“Design considerations” on page 60

6.1 Workspace design

6.1.1 Workspace design overview

Workspaces are designed using configuration files. Workspace configuration files are built with the SafeView Text Editor, using a text-based language called the Workspace Definition Language (WDL). In the event that different users or classes of users are required to perform different functions (which will probably be the norm), configuration files will have to be built for a number of different workspaces.

Refer to “ “Workspace Definition Language files” on page 218” for a complete listing and a detailed description of each component.

6.1.2 Workspace configuration files

A workspace configuration file contains a variety of information that is required by the SafeView workspace:

- Attributes of the workspace,
- Window groups that determine how the windows they contain are managed, and
- Window specifications that contain management properties for real windows.

6.2 Configuration file format

6.2.1 Configuration file format description

The general format of a WDL workspace configuration file is as follows:

```

WORKSPACE <workspace name> [description=<workspace
description>] [focusbasedsearch=<bool>] [<application
startup list>] [<number and string constants>] GROUP <top-level
group name> [description=<group description>] <sub groups>
WINDOW <window name> [description=<window description>]
<window specification properties> END WINDOW END GROUP END WORKSPACE

```

6.2.2 Workspace items

A workspace must be composed of one, and only one, top-level window group. To be of any real value, this group must contain window specifications, directly or indirectly, through subgroups. The ordering of these items (subgroups or window specifications) within groups is very important, because this ordering determines the order in which SafeView searches for a *match* at run time. Here is a description of the workspace items.

| | |
|-------------------------------|---|
| <i>workspace:</i> | WORKSPACE <i>workspace_name</i> { <i>workspace_defn_item</i> } END WORKSPACE |
| <i>workspace_name:</i> | <i>Identifier</i> |
| <i>workspace_defn_item:</i> | [<i>description</i>] [<i>focusbasedsearch</i>] [<i>startup_specification</i>] [<i>constant_declaration</i>] <i>window_group</i> |
| <i>workspace_description:</i> | description '=' <i>string</i> ',' |
| <i>focusbasedsearch:</i> | focusbasedsearch '=' <i>bool</i> ',' |
| <i>startup_specification:</i> | STARTUP [<i>string</i> {',' <i>string</i> }] END STARTUP |
| <i>constant_declaration:</i> | NUMBER <i>identifier</i> '=' <i>constant_expression</i> ',' STRING <i>identifier</i> '=' <i>string_expression</i> ',' |
| <i>window_group:</i> | GROUP <i>group_name</i> '(' (<i>group_behavior</i>) ' { <i>window_group_item</i> } END GROUP |

workspace_name

The name of the workspace.

workspace_defn_item

There are five workspace definition items:

- Workspace description,
- Setting for the focus-based search option,
- Startup specification,
- Constant declaration, and

- Window group.

workspace_description

The description field can be used to hold helpful information about this workspace. The text here does not affect SafeView management of displays. One Description entry is allowed for each Workspace, Group, and Window object. The Description content must be a single long text string (not multiple lines and no embedded quotes).

Description example:

```
description="This is my description.";
```



CAUTION

Users should document SafeView workspace entries through the use of workspace item descriptions (**DESCRIPTION** = "**comments**"), rather than with `/* */` delimited comments. The workspace item **DESCRIPTION** = "**comments**" is recognized by both SafeView editors, while any `/* */` delimited comments are discarded when the `.wdl` file is read into the standard graphical SafeView Editor.

focusbasedsearch

The focus-based search option allows the user to specify whether SafeView will, for a given workspace, use its default "top down" matching or the "focus-based" matching algorithm. For example workspaces that use the focus-based search option, see "Safeview workspace examples" on page 153."

Focus-based search command example:

The following command enables or disables the focus-based search option:

```
focusbasedsearch=yes;
```

Where: Yes = Enable focus-based search mode

startup_specification

This specification enables you to launch one or more applications at workspace startup time. Each startup string is a command line for an application.



Tip

Because the single backslash (\) character is typically the "escape" character, each is replaced with a double backslash (\\).

Experion Station PKS Startup command example:

The following example starts the Station application and, through the `/windows` switch, starts two `.htm` displays and one `.dsp` display.

```
STARTUP
"C:\Program Files\Honeywell\Experion PKS\Client\Station\Station.exe" /windows display1.htm
sysAlarmSummary.htm sysmnu.html;
END STARTUP
```

For additional information on Experion Station PKS startup commands and requirements for implementing SafeView on Experion Station PKS nodes, refer to the *Experion PKS Server and Client Configuration Guide*.

GUS Startup command examples:

Example 1: This example starts three GUS displays.

```
STARTUP
"RUNPIC c:\L1\cp1.pct";
"RUNPIC c:\L3\faceplate.pct";
"RUNPIC c:\L3\tb1.pct";
END STARTUP
```

Example 2: This example starts Native Window and a GUS display.

```
STARTUP
"lcnwindow.exe";
```

```
"RUNPIC c:\\L1\\panelset.pct ";
END STARTUP
```

constant_declaration

While both string and number constants are supported, only numeric constants are of real practical value. Numeric constants are probably most useful as a way to simplify the editing of numeric window specification properties such as POSITION, MAXSIZE, MINSIZE, REGION, and perhaps even MAXWINDOWS. For example, you can create numeric constants representing the dimensions and layout of the workspace and window groups. You can reuse these constants for each window specification that you subsequently configure. Using constants also enables you to “exactly line up” window properties. Constants also simplify the migration of SafeView configurations across monitors with different screen resolutions or sizes.

6.2.3 Window group

A workspace configuration must contain one or more window groups. There are three kinds of groups or “group behaviors:” first match, round robin, and manual select.

A workspace configuration that differentiates windows into more than one group should be composed of a top-level and one or more subgroups. Because only first match groups can have subgroups, if you want your workspace to have subgroups, the top-level group *must be* configured as a first match group. Subgroups can be defined as either round robin or manual select subgroups. A workspace that does not differentiate windows into separate groups should contain only a top-level group, typically round robin or manual select. The following provides a description of the group items.

| | |
|----------------------------|--|
| <i>window_group:</i> | GROUP <i>group_name</i> [<i>group_description</i>] '('group_behavior')' { <i>window_group_item</i> } END GROUP |
| <i>group_name:</i> | <i>Identifier</i> |
| <i>group_description</i> | description '=' <i>string</i> ',' |
| <i>group_behavior:</i> | <i>String</i> DEFAULT '=' <i>boolean_expression</i> ; MATCH '=' <i>match_expression</i> ; REGION '=' <i>windowbox</i> ',' |
| <i>window_group_item :</i> | <i>window_specification</i> <i>window_group</i> |

group_name

The name of this group.

group_description

The description field can be used to hold helpful information about this group. The text here does not affect SafeView management of displays. One Description entry is allowed for each Workspace, Group, and Window object. The Description content must be a single long text string (not multiple lines and no embedded quotes).

Description example:

```
description="This is my description.";
```

group_behavior

This property identifies the group as first match, round robin, or manual select.

DEFAULT = true/false

This property (set *true*) can appear only once in the entire configuration. If a registered window has a category that fails to match any category in the workspace configuration, the window, when displayed, is managed as if it matched the group or window specification containing this default.

MATCH = match expression

This expression (not a property) defines the logic SafeView uses to determine if a given window should be managed in this window group (see “Workspace - Match expressions” on page 39” for details).

REGION = x, y, width, height

Managed windows cannot be dragged, sized, or maximized manually or automatically, such that they extend beyond this REGION property. When specified at the group level, this property is inherited by all window specifications in this group, though specific window specifications can override this property.

Window group items

The window specification holds the actual information for managing (or constraining) the display of a managed window. Essentially, these are the SafeView properties that extend standard Windows functionality. A wide range of control is available for configuring and controlling displays for a particular end user. If care is not taken, *you might improperly design and configure a workspace such that the end-user's displays are so constrained as to be unusable*. The following provides a description of the window items.

| | |
|-------------------------------|--|
| <i>window_specification :</i> | WINDOW <i>window_spec_name</i> [<i>description</i>] { <i>window_property</i> } END WINDOW |
| <i>window_spec_name:</i> | <i>identifier</i> |
| <i>window_description</i> | description '=' <i>string</i> ',' |
| <i>window_property :</i> | ALWAYS ON TOP '=' <i>boolean_expression</i> ',' CLOSABLE '=' <i>boolean_expression</i> ',' DEFAULT '=' <i>boolean_expression</i> ',' DRAGGABLE '=' <i>boolean_expression</i> ',' GLOBALFOCUS '=' <i>boolean_expression</i> ',' MATCH '=' <i>match_expression</i> ',' MAXPOS '=' <i>point</i> ',' MAXSIZE '=' <i>size</i> ',' MAXWINDOWS '=' <i>expression</i> ',' MINIMIZABLE '=' <i>boolean_expression</i> ',' MINSIZE '=' <i>size</i> ',' PLACEHOLDER '=' <i>boolean_expression</i> ',' POSITION '=' <i>point</i> ',' REGION '=' <i>windowbox</i> ',' SIZE '=' <i>size</i> ',' SIZEABLE '=' <i>boolean_expression</i> ',' |

window_spec_name

This is the name presented in the placeholder if no application is being managed by this window specification. This name can be used in API calls to reference a particular window specification directly. This name must be unique within a window group. If window spec names are going to be used in API calls, it is a good idea to keep their names unique across the entire workspace.

window_description

The description field can be used to hold helpful information about this window. The text here does not affect SafeView management of displays. One Description entry is allowed for each Workspace, Group, and Window object. The Description content must be a single long text string (not multiple lines, and no embedded quotes).

Description example:

```
description="This is my description.";
```

window_property

ALWAYS ON TOP = true/false

This property (set to *true*) assigns the Windows “top-most” style to a window so that a window that does not have the “topmost” style can never overlay it. It can, however, be overlaid by another “top-most” window. To construct a workspace that you do not want to be overlaid, design it so that all application displays are managed and constrain non-critical displays to a region other than the critical area.

**Tip**

SafeView does not prevent applications from resetting the ALWAYS ON TOP property.

CLOSABLE = true/false

This property (set to *false*) protects a window from being closed directly. Setting CLOSABLE to *false* disables the system “Close command” for the application. The LCN Native Window is intelligent enough to disable the **File > Exit** command when managed in a window configured with CLOSABLE set to *false*. This protects against direct closure of the Native Window (see GLOBALFOCUS below). Since some applications do not have a menu, **File > Close** is not a problem. However, it should be noted that other applications will only disable the system Close command and *not* their **File > Exit** (and others) disabled. Be careful using this option.

You should use it primarily to protect critical displays from being closed directly (default = standard windows behavior, that is, close is allowed).

DEFAULT = true/false

This property (set to *true*) allows you to designate the MATCH expression for a window as the default MATCH expression for all registered windows (workspace client displays associated with a given category using the “RegisterWindowData” or “CreateWorkspaceClient” API call) that do not match their workspace configuration. DEFAULT = *true* can appear only once in a workspace configuration file.

DRAGGABLE = true/false

This property (set to *false*) enables you to disable the repositioning of a window by not allowing a user to drag the window. A nondraggable window can be “sized” and thus moved by stretching or shrinking. Consider making nondraggable windows also non-sizeable by setting SIZEABLE to *false*. Another way to constrain window positioning is by using the REGION property.

GLOBALFOCUS = true/false

This property (set to *false*) configures a window such that the global focus button is not available for that window. This protects an operator from replacing a critical application with another managed application. Using this option in conjunction with the CLOSABLE = *false* option effectively disables the operator's ability to close a critical application. Using these in conjunction with the MINIMIZE = *false* option assures that important operational displays are not closed, *and* not iconized. In addition, using the ALWAYS ON TOP property, and constraining other applications from being always on top and limiting their regions, provides a high degree of protection for critical applications.

Workspace windows in a single-group round robin or manual select workspace have, by default, redundant output focus and global focus buttons. By setting GLOBALFOCUS = *false* for these windows, the redundant focus buttons are removed.

MATCH = match expression

This is an expression, rather than a property that defines the logic SafeView uses to determine if a window should be managed in this window group. This property is only valid for window specifications directly owned

by a first match group. For more details, see ““Workspace - Match expressions” on page 39.” An example when this might be used is to “catch all” displays not explicitly matched elsewhere.

MAXPOS = x, y

This property specifies the maximized position (upper left *x* and *y* coordinates) of a maximized window.

MAXSIZE = width, height

This property specifies the maximum width and height allowed for a window. This property stops “growing” a window if you attempt to size the window beyond either the *x* or *y* parameter. Also, if the user maximizes the window, this property constrains the window to the *MAXSIZE* dimensions.

MAXWINDOWS = 1/infinite

This property specifies whether a window specification has a 1:1 (the default) relationship with managed windows (as is the case for round robin and manual select groups) or if it has a one-to-many relationship with managed windows. This property is applicable *only* for window specifications in first match window groups.

It is generally preferable to leave *MAXWINDOWS* set to 1 when the window specification includes a number of properties such as *POSITION* or *SIZE*. This will ensure a one-to-one correspondence between each display and its associated window specification.

If *MAXWINDOWS* is set to “infinite” in a window specification, any number of displays can be managed by this single window specification. A single window specification can be used to manage many windows simultaneously. This is helpful when you wish to “mildly constrain” an indeterminate number of windows that share one or more properties, such as a similar constraining region. Application displays managed with *MAXWINDOWS = Infinite* are never closed by SafeView unless an application window has global output focus, in which case it is replaced by the next SafeView display invoked. They can, however, be closed in normal Windows fashion.

MINIMIZABLE = true/false

This property (set to *false*) disables the ability to minimize (iconize) a window. When it is set equal to *true*, the Minimize function has no effect. An example of where this property can be used is for critical alarm displays. These displays could be configured so that they are “always on top,” located in a region that excludes less important displays, set up with an adequate minimum size and with *MINIMIZABLE* set to *false* so they cannot be minimized. This is normally how this function is used. Also note that while a maximizable property is not currently available, including “*MAXIMIZABLE = true/false*” will not produce a syntax error.

MINSIZE = width, height

This property specifies the minimum size for a window. It prevents the user from sizing the window below the given dimensions. Note that iconization is not affected by this property.

PLACEHOLDER = true/false

This property indicates whether a “placeholder” should be displayed, according to this window specification's properties, when no application is currently displayed using this window specification. This provides a means to display the workspace “structure” when a full complement of application displays remains to be managed. The SafeView Control Panel contains a button (and an API) that allows placeholders to be either hidden or shown. Only the placeholders that are set *true* in their window specifications are ever shown.

POSITION = x, y

This property specifies the initial position, (*x* and *y* upper left window coordinates) of a window. It is constrained by the *REGION* property. If *POSITION* is not configured, the actual position is determined by the Windows operating system. If *POSITION* is configured and *SIZE* is not configured, the system will sacrifice the size before sacrificing the position in conforming to a specified *REGION* property. If both *POSITION* and *SIZE* are configured, the syntax checker verifies that the window fits into a specified region. If the “return to original position” command (Reposition) is executed, SafeView will move the window back to this position if that was where the window was initially displayed.

REGION = x, y, width, height

This property specifies the area within which a window can be displayed. This property can be inherited from round robin and manual select group, if it is configured at these group levels. If it is configured here, at the

window specification level, it will override the window group's region specification. Managed windows cannot be dragged, sized, or maximized manually or automatically, such that they extend beyond this region. This property nicely constrains a maximize-window, for example, to a given area that does not necessarily cover the entire display region. It also provides some functionality that at first may seem surprising; namely, that a window can be dragged only so far and then it stops, for no apparent reason. A good design principle is to configure each window group (excluding first match) to have a prescribed region; that is, each window specification in the group should have a similar REGION attribute.

SIZE = width, height

This property specifies the initial width and height of a window. It is constrained by the REGION property. If SIZE is not configured, the Windows operating system determines it. If SIZE is configured and POSITION is not configured, the system will sacrifice position before sacrificing size in conforming to a specified REGION property. If the Reposition (“return to original position”) command is executed, SafeView will restore the window to its original size.

SIZEABLE = true/false

This property indicates whether a user can size a window. When *SIZEABLE = false*, the window style is modified at initial display time such that the window has a “thin-frame.” This provides the visual cue that the window cannot be sized. The normal Window's Minimize and Maximize functions are still available. It should be noted that even though some applications redisplay their original (thick) frame and appear to be sizable, they are not.

6.3 Design considerations

6.3.1 Design considerations overview

SafeView provides a high degree of flexibility and functionality for altering several characteristics (such as POSITION, SIZE, and DRAGGABLE) of top-level application displays. With this flexibility comes the responsibility of the workspace developer to take care that workspace configurations do not so severely constrain applications as to render them unusable. Here are some tips intended to help avoid potential problems.

6.3.2 Organize the workspace

Organize the workspace so that you can easily predict which application displays are managed by a particular window group. This can be accomplished by being sufficiently selective in each MATCH expression.

6.3.3 Make sure resized windows are usable

Make sure that a window group is not so severely constrained as to be unusable. For example, if an application must have a particular minimum size (width, height), do not allow it to be managed by a smaller-sized window specification. This is especially important when managing applications that have non-sizeable displays, such as dialog-based application windows.

An exception would be that if the window is sizeable, it could also be acceptable to present the resized application window.

Dialog-based applications can become unusable if they are resized. If you designate that an application is sizeable (SIZEABLE set to *true* in the window specification), resize it to an inappropriate size, and then exit or reload SafeView, it can become unusable. It will remain at its current size and assume its non-sizeable window style.

6.3.4 Determine area available for a client picture

The actual workspace size is:

workspace width x (workspace height - operating system task bar size)

The task bar size for the operating system is 28 pixels high.

If the workspace is configured to be 1024x768 pixels and the user wants the task bar to always display, then configure the window to be 1024x740 pixels (768-28=740).

A side by side dual screen width and height would be 2048x768, with screen resolution of 1024x768.

A stacked dual screen with 1024x768 resolution would have a height of 1536 and width of 1024.

To determine the client area for a given workspace, subtract the window title bar, status bar, and borders from the area available:

- Subtract 6 pixels for each border (top/bottom/left/right).
- Subtract *n* pixels from the height for the Windows title bar and status bar.

Where *n* =

36 If using Windows Classic or Windows Standard scheme

(16 pixels each for title and status bar)

40 If using a Large scheme

44 If using an Extra Large scheme

**Tip**

Users select the scheme through **Control Panel > Display > Display Properties > Appearance**.

The size of the Native Window is 652x524. This size includes its title bar and status bar.

Example: Experion Station

In this example, the user sized an Experion Station picture, ensuring there would be no empty space or scroll bars when the picture was displayed in SafeView. The screen resolution in this example is 1024x768.

In SafeView, the picture placeholder window is sized 1024x588. No allowance is made for borders or the title bar because that is done when the picture is built. The SafeView window height of 588 allocated 53 pixels for Experion Station's status bar, 99 pixels for Experion Station's menu bar, and 28 pixels for the Windows task bar. To achieve a full screen picture without scrollbars, the user created a picture that was 1012x540. Here is how that size was determined:

- The width of the graphic is 1024 less 12 for the borders (6 left and 6 right) = 1012
- The height of the graphic is as follows:
 - 768 less 12 for the borders (6 top and 6 bottom) = 756
 - 756 less 53 for Station's Status Bar = 703
 - 703 less 99 for Station's Menu Bar = 604
 - 604 less 28 for the Windows task bar = 576
 - 576 less 36 for the Windows Standard title bar = 540

6.3.5 Users who can specify 'Always On Top'

Applications that allow users to specify at any time that a window be “always on top” should be region-constrained so that they cannot overlay other critical (presumably configured as *always on top*) displays.

6.3.6 Include a default MATCH expression

Recall that workspace client applications (those that are registered by the Workspace API) are registered with a given window category (string) value. By including a default MATCH expression, you can manage even those client applications that register with non-matching or unknown window categories. If a window is registered with an unrecognized window category (and if it does not match according to its module or title), the window is managed as if it matched the default MATCH expression.

**Tip**

Category must be explicitly excluded from “catch-all” MATCH expressions, as discussed in the next topic.

6.3.7 Include a 'catch-all' window group

The desired approach to workspace organization is to have a very good idea of the applications that are to be managed and then organize the workspace accordingly. In a Windows environment, however, it can be impossible to predict all of the applications that can be invoked, even at an operator's station. For this reason, it can be appropriate to configure a “catch all” window group to assure that all applications, even the unpredictable ones, are managed, but care must be taken.

It is important to differentiate between (a) catching a specific number of applications and (b) catching an unlimited number of applications, as described below:

- For a limited number of applications (for example, four), a round robin window group set up with the specifications for four round robin windows, constrained only by region, is one suggested approach.
- For an unlimited number of applications, a first match group with a single window specification with MAXWINDOWS set to *infinite*, constrained only by region can be used.

Here is a valid scenario for having a “catch all”, for example, `title(“?*)` or `module(“?*)`. The engineer cannot predict all of the third-party applications that can execute during operations, but has the requirement that these un-configured applications be restricted to a particular “out of the way” area, perhaps on an upper-tier physical window. For this example, unconfigured third-party applications are those that do not match a specific criterion for title or module other than the wide-open “?*) default. To satisfy this case, consider the following approach:

- Include in the workspace a special window group for miscellaneous third-party applications. Configure this as a round robin group with a single window specification (or as a single window specification at the end of a first match group.)
- Do not set its POSITION property, but limit its REGION to the appropriate out-of-the-way location.
- Do not set POSITION or SIZE for a first match group when MAXWINDOWS is set to *infinite*.
- Exclude all category entries from the catchall MATCH expression. This is because the workspace language supports a specific means for default handling of workspace client windows that happen to register a window with an unknown window category name. It is the `DEFAULT =yes` statement associated with a MATCH expression. Thus, an example of a catch all MATCH expression that also allows for separate default handling of unknown window categories is:

`match title(“?*) and not category(“?*)`;

The result is that all applications that do not match the more specific criteria of other groups will “fall through” to this group and be only minimally managed; namely, they will be presented and limited to the relegated display area. Alternatively, you could set the POSITION attribute, but not the REGION attribute. This approach would allow the third party applications to be moved anywhere after being initially placed in a specific location. The downside to this approach is that multiple third-party applications would exactly cover previous ones, if the previous ones were not dragged elsewhere or closed.

6.3.8 Close managed windows

When a new display is invoked and managed according to a particular window specification, SafeView closes the existing display by sending it a standard WM_CLOSE message. This is analogous to selecting **File > Close** from the Window's system menu, or double-clicking the Control menu box in the upper-left corner of the title bar. Most applications handle this message by shutting down, which is the desired behavior. Some applications however, such as document editors, give the operator a chance to *save changes before exit* or even *cancel* the close request. If SafeView sends an application a WM_CLOSE message and it fails to close for whatever reason, the application becomes unmanaged. The operator should close it.

If an application fails to respond to a WM_CLOSE message from SafeView, it becomes unmanaged. It is necessary to keep the save changes and cancel dialog always visible to the user. Here are some suggestions on how to ensure that WM_CLOSE messages will be handled correctly:

- Avoid setting up applications such as document editors that give the operator a chance to *save changes before exit*, or even *cancel* the **File > Close** request with draggable windows, if at all possible. You can also configure these types of applications with MAXWINDOWS set to *infinite*.
- When SafeView closes an application that has produced some edited work, use *save and exit*, not *cancel and return*.
- Do not leave applications that are ready to be replaced by SafeView (windows of output focus) in a state where they cannot handle a Close request (which generates a WM_CLOSE message) correctly. For example, an **About** box is displayed.

If SafeView is in the process of shutting down, with either a Visual Basic application running or modal dialogs currently on display for an application, a message will appear asking you to close down all of these applications.

6.3.9 Configure sufficient windows

Some applications can create multiple top-level windows. Be sure that workspace window groups configured to match such applications have sufficient windows to support all of such applications' windows.

Consider, for example, a Visual Basic application that consists of a main application window and that has an option to create four additional top-level windows. This application is now invoked and matches a four-window,

round robin window group. The main application window is displayed; then, the user selects the option that invokes the other four displays associated with the main display (all of which match the same group). Because the fourth additional display would replace the original main display (sending it a WM_CLOSE message), the whole application (perhaps including all of the four existing displays, if they are owned) would disappear. This would be quite unexpected.

Alternatives to this case include the following:

- Configure sufficient windows to handle the number of simultaneous windows you expect to be generated automatically by such an application. Manage such applications in a “catch-all” group as described above.
- “Lock” the master window, so that it cannot be automatically replaced.

6.3.10 Legacy TPS faceplate limitation for ES-T

HMIWeb TPS faceplates are provided for Experion systems that connect to TPS (LCN) systems. Alternatively, Experion also provides the option for using the GUS legacy faceplates instead. In this case, be advised that on an Experion Station for TPS (ES-T), only four such faceplates are supported at any one time. You should therefore avoid SafeView scenarios where more than three TPS Faceplate ActiveX controls will be present, because a fifth one would be required in order for SafeView to close the fourth one to make room for it.

If four TPS Faceplate ActiveX controls are currently displayed in a SafeView workspace on an ES-T, and the operator invokes a new *hmiweb* display containing an embedded TPS Faceplate ActiveX control, then the newly invoked display may not get the faceplate channel. This is because the Station application attempts to create all objects before “asking” SafeView to close any existing displays (and therefore to release a faceplate channel).

If the TPS Faceplate ActiveX control cannot connect to the TPS/LCN, then it draws a large, red X in its window to indicate that it is not collecting TPS/LCN data, and displays the following error message: “Unable to connect. If no other faceplates are connected, verify the YGOCX Load Module version.”

6.3.11 Configure Microsoft Word applications

Because of a problem in Microsoft Word, there is an inconsistency in the way Microsoft Word applications function when configured with the ALWAYS ON TOP property. The drop-down menus that are provided with this application will not assume the ALWAYS ON TOP functionality. Some Microsoft applications do not exhibit this particular behavior. Do not configure Microsoft Word applications as “always on top” if you intend to use the drop-down menus. It is also possible that “always on top” applications do not have modal dialogs. This could also be a problem.

6.3.12 Configure LCN status applet

To allow an LCN Status applet to be draggable within the window, you must set both the SIZEABLE and DRAGGABLE properties in the window specification to *true*.

7 Configuring a workspace

Related topics

“Using the Text Editor or Graphical Workspace Editor” on page 66

“Before you begin” on page 67

“Using the Text Editor to configure a workspace” on page 68

“Using the Graphical Workspace Editor to configure a workspace” on page 72

7.1 Using the Text Editor or Graphical Workspace Editor

SafeView's Text Editor (TextEdit) and Graphical Workspace Editor (GWE) are used for creating and editing workspace configuration files. The Text Editor is a simple text-editing program. The GWE replaces the majority of the functionality in the Text Editor. The Text Editor can be invoked directly from the GWE, to provide the following benefits:

- A means to view a workspace configuration in the form of a text file (just as it would be saved on disk),
- A means to view and edit workspace configuration files that are syntactically incorrect or otherwise corrupted, such that they may not be successfully parsed and rendered graphically by the GWE,
- Additional syntax checking, and
- The means to print a workspace configuration (text) file.

7.2 Before you begin

Read the following sections before you attempt to build a workspace configuration file:

- ““Designing a workspace” on page 51” describes the workspace configuration file format.
- ““Safeview workspace examples” on page 153” provides examples of configuration files. Because different users or classes of users can be required to perform different functions, you will probably have to create configuration files for a number of different workspaces.
- “Appendix A: Workspace Definition Language” explains the Workspace Definition Language (WDL) in detail.



Attention

Only station displays are supported as tabbed windows. Hence, it is recommended that only station windows be managed in SafeView windows that are configured as tabbed windows.

7.3 Using the Text Editor to configure a workspace

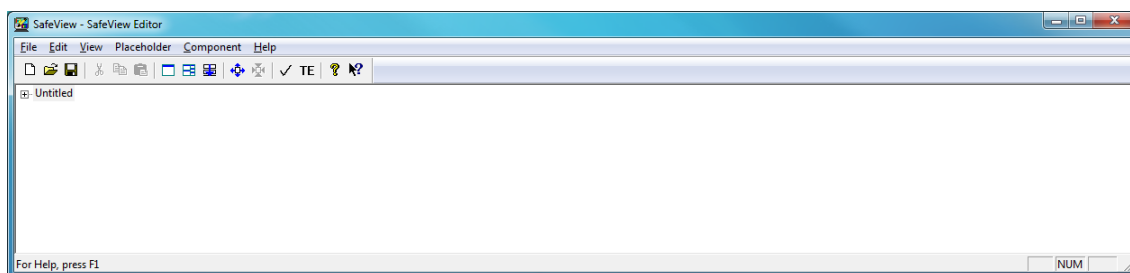
7.3.1 Text Editor Description

The Text Editor uses a text-based language called the Workspace Definition Language (WDL) to build workspace configuration files. It is a Multi-Document Utility (MDI) application. How SafeView manages an active workspace is completely determined by the contents of the configuration files that have been built for it. Only one workspace configuration can be active at a time.

7.3.2 Starting the Text Editor

The following procedure describes how to start the text editor from a Windows system with the SafeView package installed in accordance with Honeywell recommendations.

- Choose **Start > All Programs > Honeywell Experion PKS > SafeView > SafeView Text Editor**.
The **SafeView TextEdit** window appears.



7.3.3 Building a new workspace configuration file

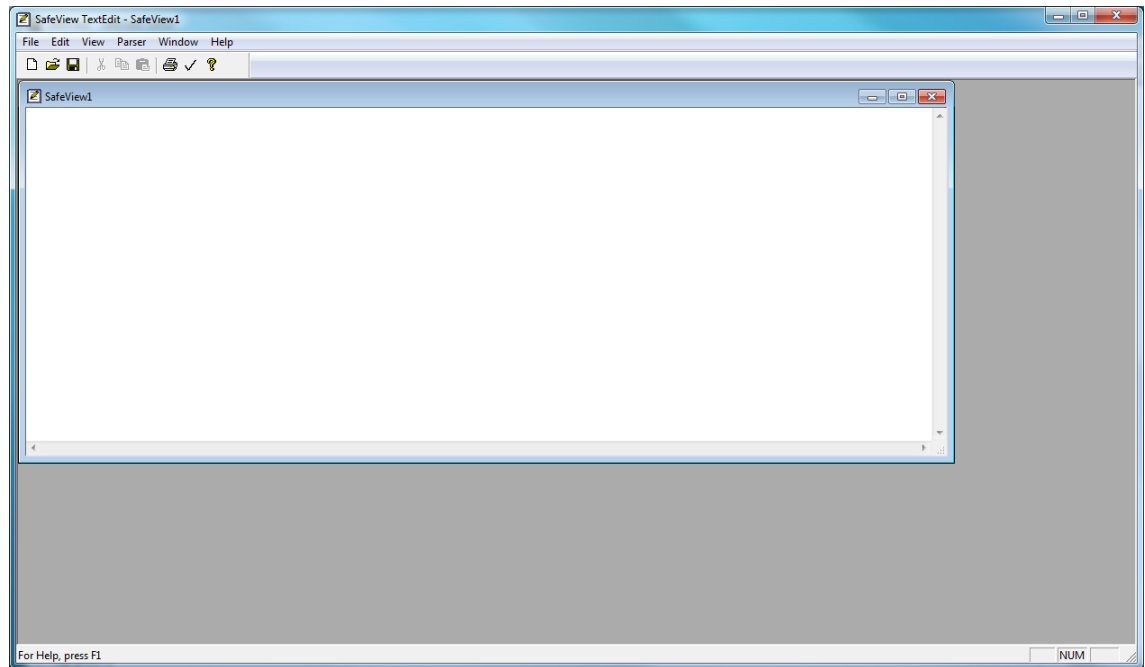
- 1 Open **SafeView TextEdit**.
- 2 Create a new workspace configuration file.
- 3 Use WDL to build the workspace (see “Appendix A: Workspace Definition Language”).
- 4 Check the file syntax (Parser).
- 5 If an error is detected, correct it and repeat step 4. When there are no errors, go to Step 6.
- 6 Save the new file.
- 7 Close **SafeView TextEdit**.

7.3.4 Changing an existing workspace configuration file

- 1 Open **SafeView TextEdit**.
- 2 Open the workspace configuration file you want to change.
- 3 Edit the new file (see Appendix A: Workspace Definition Language’).
- 4 Check the file syntax (Parser).
- 5 If an error is detected, correct it and repeat Step 4. When there are no errors, go to Step 6.
- 6 Save the changed file.
- 7 Close **SafeView TextEdit**.

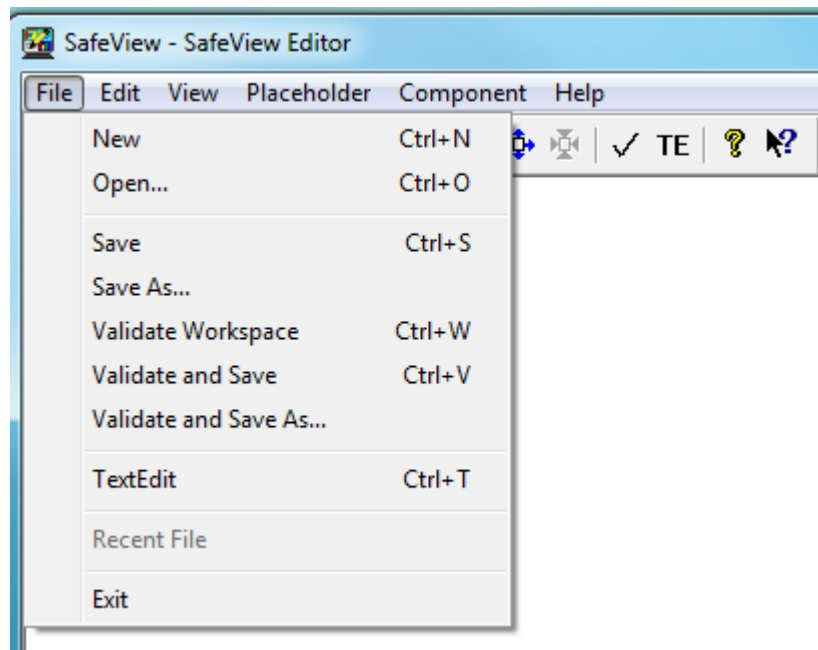
7.3.5 Creating a new workspace configuration file

- 1 On the **SafeView TextEdit** menu bar, click *File* to open its drop-down menu.
- 2 On the **File** drop-down menu, click *New* (or press **Ctrl+N** simultaneously).
A new, blank workspace configuration file is created and appears in the **SafeView TextEdit** window. In this example, the new workspace configuration file is titled “SafeView1,” indicating that this is the first file configured during the current editing session.



7.3.6 Opening an existing workspace configuration file

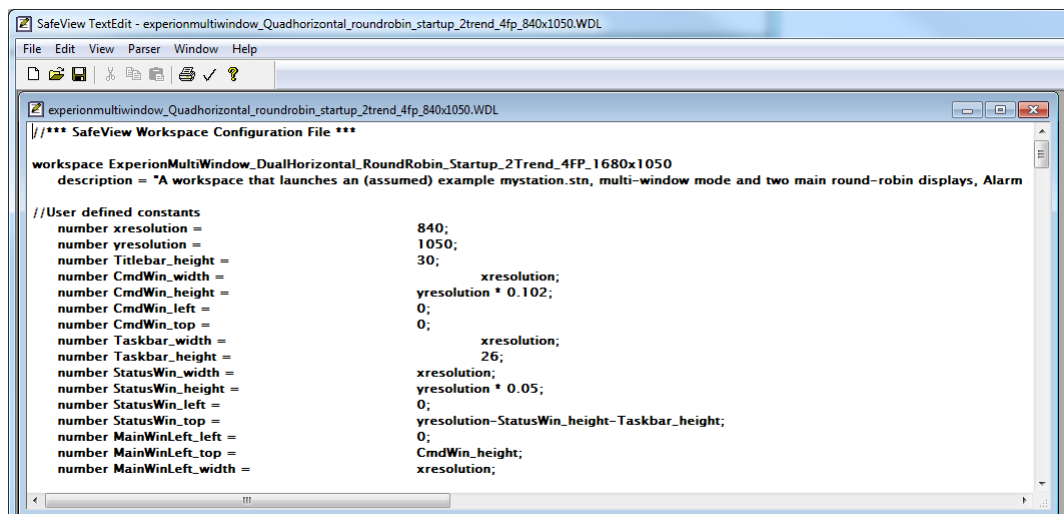
- 1 On the **SafeView TextEdit** menu bar, click *File* to open its drop-down menu.
The *File* drop-down menu appears.



- 2 If the configuration file you want to open is in the “recently opened” list, enter the associated number (for example, 1, 2, 3, or 4). Go to step 4.
- 3 If the configuration file you want to open is not in the “recently opened” list, do the following steps:
 - a Click **Open** (or press **Ctrl+O** simultaneously).
The standard Windows **Open** dialog box appears.
 - b Type the correct file type (.wdl), drive, directory path, and file name of the configuration file you want to open and then click **Open**. Go to step 4.

The selected .wdl configuration file appears in the **SafeView TextEdit** window.

The menu bar is automatically expanded to include three new items: **Edit**, **Parser**, and **Window**. In addition, four new items are added to the **File** drop-down menu: **Close**, **Save**, **Save As**, and **Print**.



7.3.7 Checking the workspace configuration file syntax

- 1 Click the  button on the Toolbar.
Or,

On the **SafeView TextEdit** menu bar, click *Parser* and then click **Check Syntax**.

Syntax checking begins, starting at the beginning of the file.

If an error is detected, an error message appears.

- 2 Correct the error and then repeat this procedure until the “No errors detected” message appears.

7.4 Using the Graphical Workspace Editor to configure a workspace

7.4.1 Graphical Workspace Editor description

The primary view provided by the Graphical Workspace Editor (GWE) main menu is a “tree view” that displays the structure of a given workspace configuration, in terms of the workspace and its component window groups and specifications. The user can view and edit each workspace component individually using tabbed property pages.

The GWE provides no API programmatic interface. The GWE protects you from configuring a workspace with syntax errors or creating syntax errors when you are changing a workspace by displaying a warning message. The following table shows the relationship between a configuration file and a GWE main menu.

| Workspace Configuration File | GWE Main Menu |
|--|------------------------|
| WORKSPACE <workspace name> <workspace description> <focusbasedsearch setting> <application startup list> <number and string constants> | SafeViewWorkspaceThree |
| GROUP <top-level group name> <group description> | PrimaryWorkspaceGroup |
| GROUP <sub group name> <subgroup description> <group behavior items> | TheRoundRobinGroup |
| WINDOW <window name> <window description> <window specification properties> END WINDOW | GoodWindowName |
| WINDOW <window name> <window description> <window specification properties> END WINDOW END GROUP | SecondWindowName |
| WINDOW <window name> <window description> <window specification properties> END WINDOW | TopLevelWindow_One |
| WINDOW <window name> <window specification properties> <window description> END WINDOW END GROUP END WORKSPACE | SecondFirstMatchWindow |

7.4.2 Tree view of a workspace configuration

One of the two principal views of a SafeView workspace configuration is the “tree view” which displays the workspace hierarchy. It is initiated through the initial SafeView GWE tree view display. The other principal view (described later in this section) is the display of an actual workspace placeholder. Both views are developed from the same source, the *.wdl* UNICODE text file that is created for each workspace.

7.4.3 Workspace hierarchy

The top item in any GWE hierarchy represents the workspace name. It is always followed by the mandatory top-level group item, which appears one level lower in the hierarchy. If any other subgroups and windows exist, they can be viewed at lower levels by expanding the workspace hierarchy. You can expand or collapse your view of the workspace in stages, depending on the number of workspace items defined. There is no single action that expands all of the workspace.

In the following figure, the initial tree view has a workspace named “Untitled” with a single top-level group named “MainGroup.”



7.4.4 Starting the GWE

- Choose **Start > All Programs > Honeywell Experion PKS > SafeView > SafeView Graphical Editor**. The **SafeView Editor** window appears, displaying the initial tree view workspace named “Untitled.”

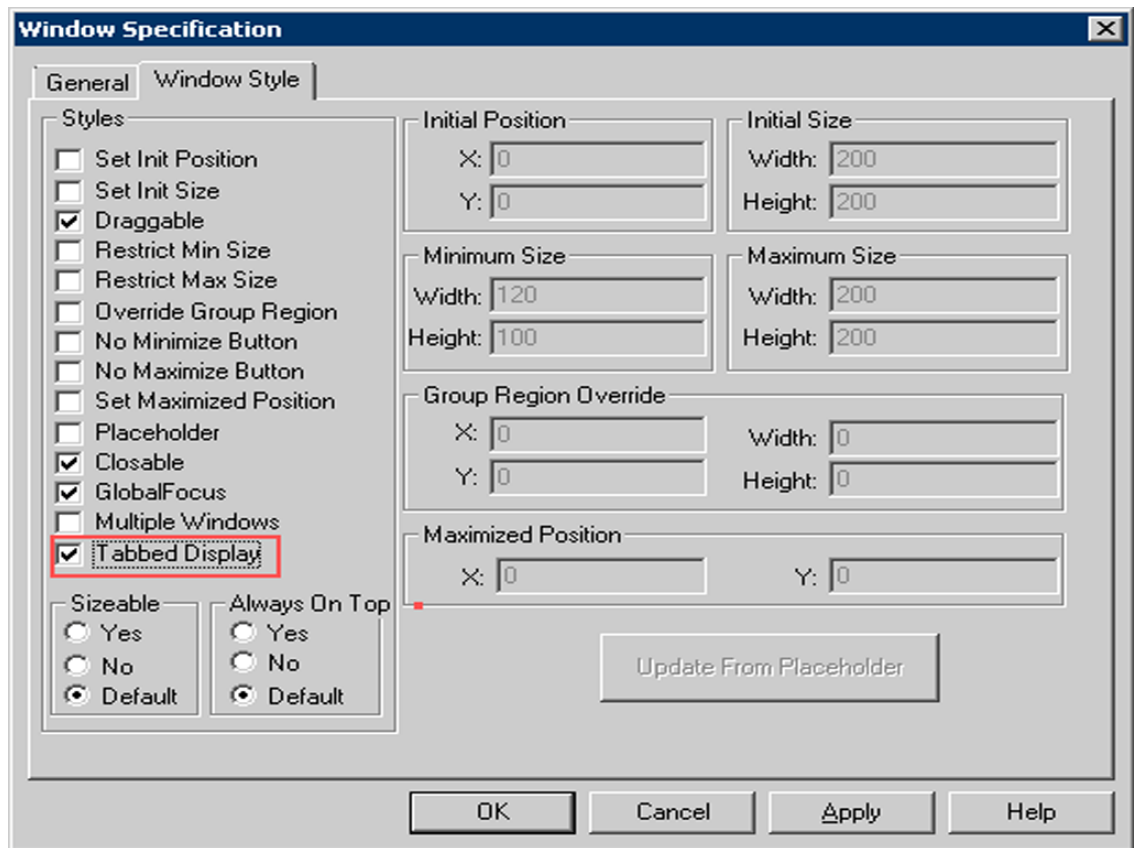
7.4.5 Configuring a new workspace

- 1 Start the GWE.
- 2 Click **File** to open its drop-down menu.
- 3 Create a new workspace.
- 4 Edit the workspace general properties, startup commands, and constants.
- 5 Edit the group general properties, and default styles.
- 6 Edit the window general properties and specifications.
- 7 Check the workspace syntax.
- 8 If errors are detected, correct them and then repeat steps 4 thru 7 until no errors are detected.
- 9 Save the new workspace.
- 10 Close the GWE.

7.4.6 Configuring tabbed displays on multi-window station

Use the SafeView Graphical Workspace Editor or the SafeView Text Editor to enable the Tabbed Display feature for that window.

- 1 In the graphical editor, to enable the tabbed display, click **Windows Style** tab of the Window Specification dialog box.
- 2 In the **Styles** area, select **Tabbed Display**.



- 3 Click to **OK**.

You can enable the tabbed display using the text editor. For example,

```

window StationTabwindow1
position = 0, 0;
size = 800, 600;
tabwindow = yes;
globalfocus = no;
sizeable = no;
placeholder = yes;
maximizable = no;
end window
  
```



Attention

Only station displays are supported as tabbed windows. Hence, it is recommended that only station windows be managed in SafeView windows that are configured as tabbed windows.

7.4.7 Changing an existing workspace

- 1 Start the GWE.
- 2 Open the existing workspace and navigate the workspace hierarchy as required.
- 3 Edit the workspace general properties, startup commands, and constants.
- 4 Edit the group general properties, and default styles.
- 5 Edit the window general properties and specifications.
- 6 Check the workspace syntax.
- 7 If errors are detected, correct them and then repeat steps 4 thru 7 until no errors are detected.
- 8 Repeat steps 3 thru 8 until all required workspace changes have been made.
- 9 Save the changed workspace.

- 10 Close the GWE.

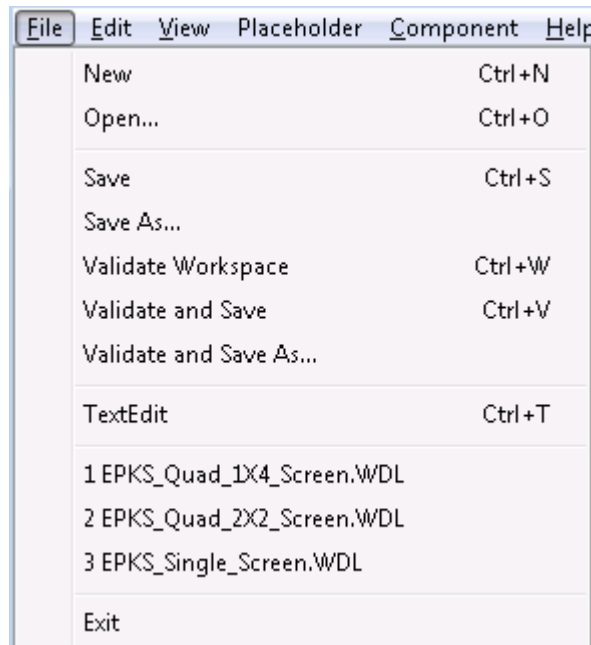
7.4.8 Creating a new workspace

- 1 Click the  icon on the GWE toolbar. Continue to step 2.

Or,


On the **SafeView Editor** menu bar, click **File**.

The **File** drop-down menu appears. The following example shows that three workspaces have already been created.



- 2 On the **File** drop-down menu, click **New** (or press **Ctrl+N** simultaneously).
A tree view of the initial, “Untitled” workspace appears in the **SafeView Editor** window.

7.4.9 Opening an existing workspace

- 1 Click the  icon on the GWE toolbar to open the **Open** dialog box. Continue to step 3.
Or,
On the **SafeView Editor** menu bar, click **File** to open its drop-down menu. Continue to step 2.
- 2 If the file name of the workspace you want to open is listed in the following **File** drop down menu, click that workspace or type its related number (1, 2, 3, 4).

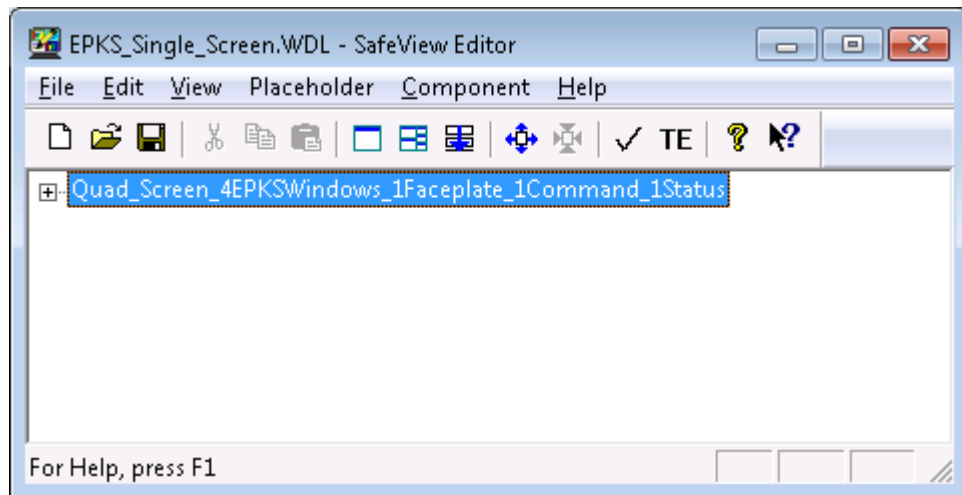
| |
|----------------------------|
| 1 EPKS_Quad_1X4_Screen.WDL |
| 2 EPKS_Quad_2X2_Screen.WDL |
| 3 EPKS_Single_Screen.WDL |

If the file name of the workspace you want to open is not listed, click **Open** on the **File** drop-down menu (or press **Ctrl+O** simultaneously).

Note: When using the GWE to open workspaces, the file names of the last four opened workspaces are listed. In this example, “EPKS_Quad_1X4_Screen.WDL” was the most recent one opened, with all others opened before it in descending order.

- 3 Type the correct file type (*.wdl*), drive, directory path, and name of the workspace you want to open and then click **Open**.

The selected workspace is retrieved and checked for correct syntax. If no syntax errors are found, a tree view of that workspace appears.

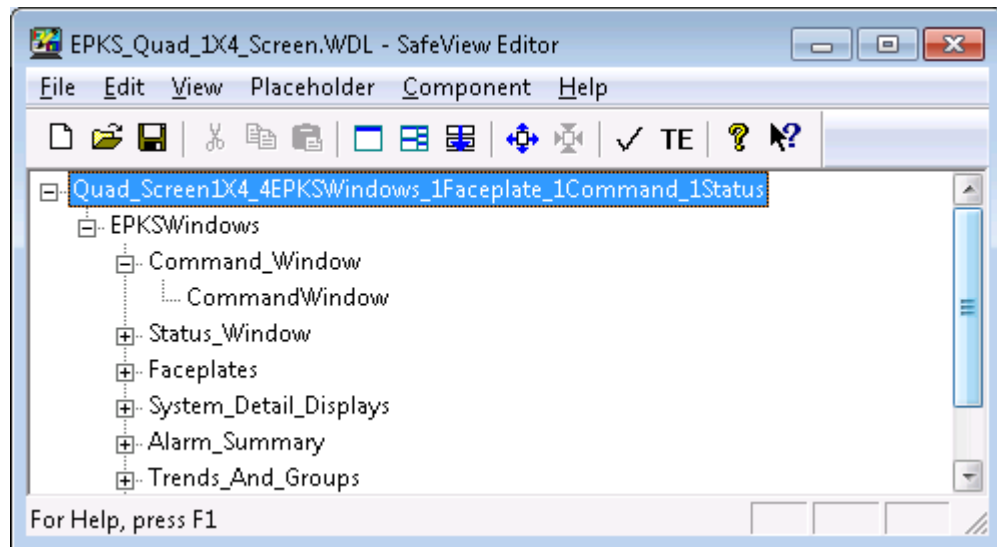


- 4 If the **Placeholder Show All** flag is selected (the default value), placeholders representing the selected workspace will also be shown in the background.
If a syntax error was found, a dialog box similar to the following will appear. At this point, you can click **Yes** to open the **Text Editor**, allowing you to view the corrupted text file and fix it.



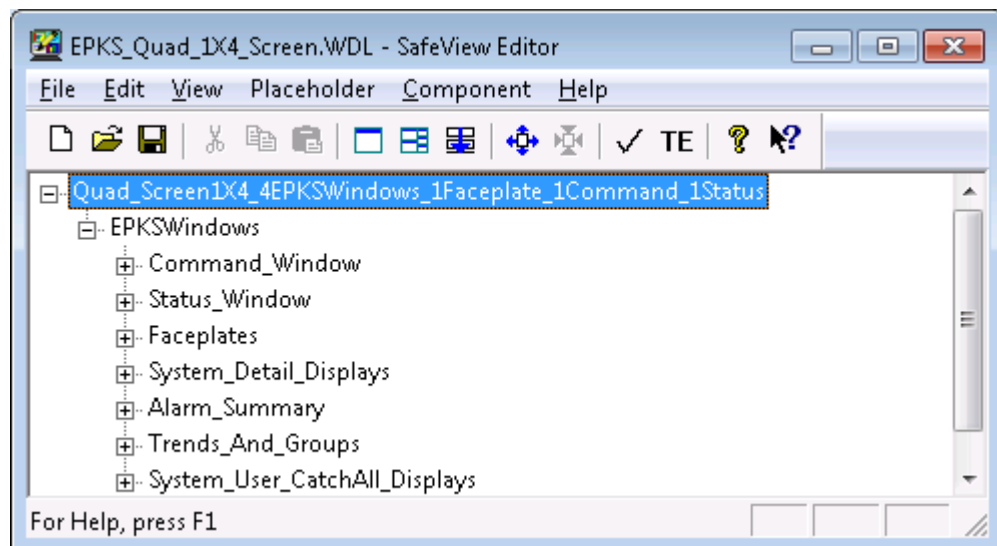
7.4.10 Navigating the workspace hierarchy

- 1 To expand your view of the workspace, click the +box to the left of the tree item you want to expand (or press **ENTER**).
The next lower tree item and its component items (if any) appear.




- 2 To collapse your view of the workspace, click the “-” box to the left of each tree item you want to collapse (or press **ENTER**).

The component items of each item disappear and the “-” box changes to a + box.



7.4.11 Saving the active workspace


Using “Save”

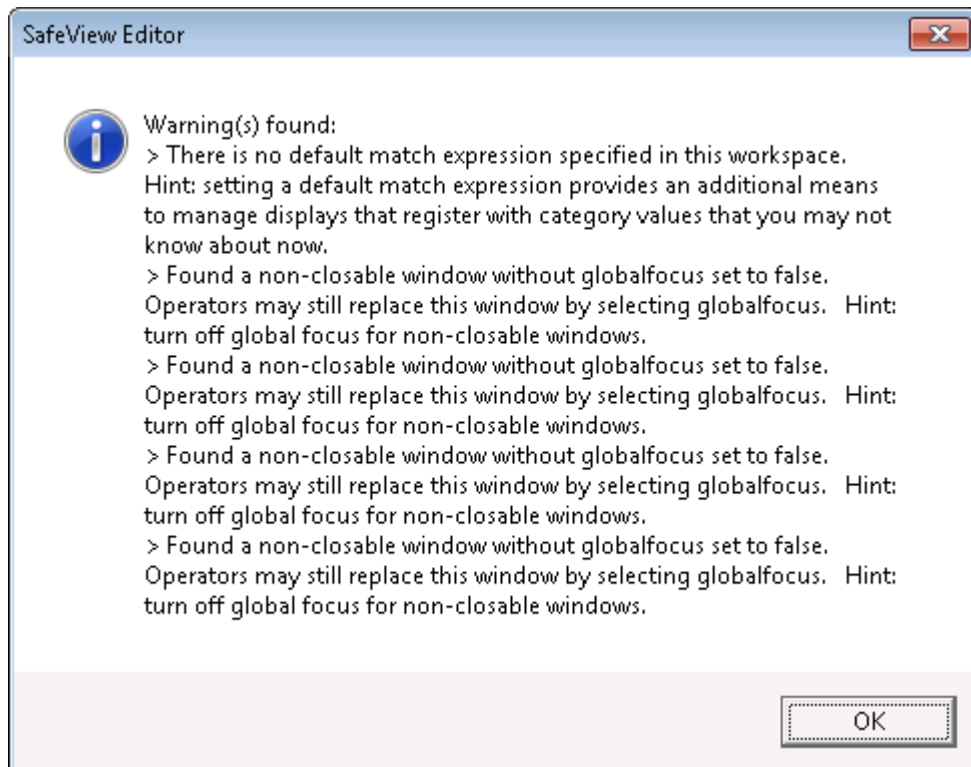
- 1 Use the following procedure to save the active workspace to its original location.
- 2 Click the  icon on the GWE toolbar. Continue to step 3.
Or,
On the **SafeView Editor** menu bar, click **File**.
The **File** drop-down menu appears. Continue to step 2.
- 3 On the **File** drop-down menu, click **Save** (or press **Ctrl+S** simultaneously).
The active workspace is saved.

Using “Save As”

- 1 Use the following procedure to save the active workspace under a specified name to a specified location.
- 2 On the **File** drop-down menu, click **Save As** (or press **Ctrl+A** simultaneously).
The standard Word **Save As** dialog box appears.
- 3 As needed, connect to a new network, type a new drive, a new directory, and a new workspace name, and then click **OK**.
The current workspace is saved under the new name and/or new location, if specified. The new workspace replaces the previous workspace in the **SafeView Editor** window.

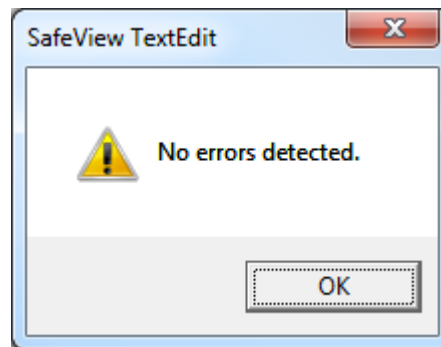
7.4.12 Checking the workspace syntax

- 1 Click the  icon on the GWE toolbar. Continue to step 3.
Or,
On the **SafeView Editor** menu bar, click **File** to open the drop-down menu. Continue to step 2.
- 2 On the **File** drop-down menu, click one of the following options:
 - **Validate Workspace** (or press **Ctrl+W** simultaneously) to check the syntax of the active workspace.
 - **Validate and Save** (or press **Ctrl+V** simultaneously) to check the syntax of the active workspace and then save it to its original location.
 - **Validate and Save As** to check the syntax of the active workspace and then save it under a specified name to a specified location.
- 3 Syntax checking begins, starting at the beginning of the workspace. If any non-critical problems are detected, a SafeView Editor “Warning(s) found” message will be displayed.



- 4 Record the description of the problem(s) detected and then click **OK**.
The “Warning(s) found” message disappears.
- 5 If desired, fix the problem(s) and repeat the previous steps until no warning message appears.

- 6 If a syntax error is detected, correct it and repeat the previous steps until no errors are detected. If no syntax errors are detected, the “No errors found” message appears.



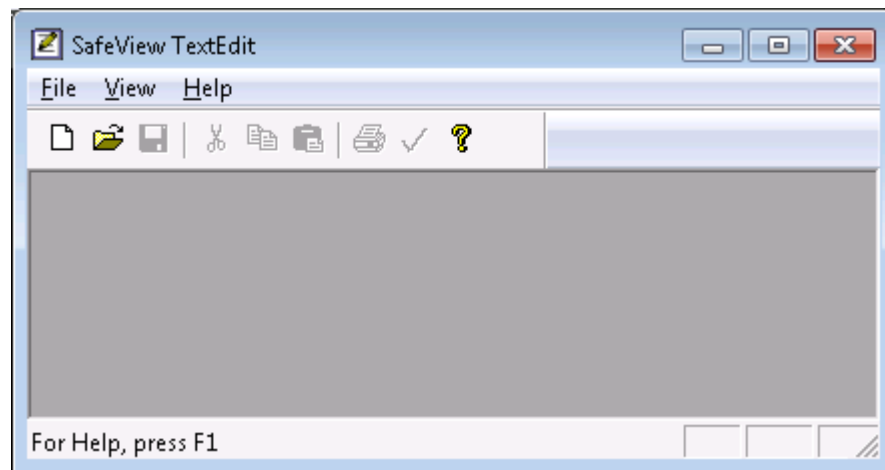
- 7 Click **OK**.
The “No errors found” message disappears.

**Tip**

When you are creating a workspace, the GWE helps to avoid syntax errors by automatically generating the configuration file. However, a *foolproof* way to detect syntax errors is to start the Text Editor and then check the workspace configuration file syntax there.

7.4.13 Starting TextEdit from the GWE


- 1 Click the **TE** icon on the **GWE** toolbar. Continue to step 3.
Or,
On the **SafeView Editor** menu bar, click **File** to open its drop-down menu. Continue to step 2.
- 2 On the **File** drop-down menu, click **TextEdit** (or press **Ctrl+T** simultaneously).
- 3 The **SafeView TextEdit** window appears, overlaying the **SafeView Editor** window.




7.4.14 Cutting a tree item

This function can only be used to do simple cuts. You cannot do complex cuts, such as a group with one or more windows.

7.4.15 Copying a tree item

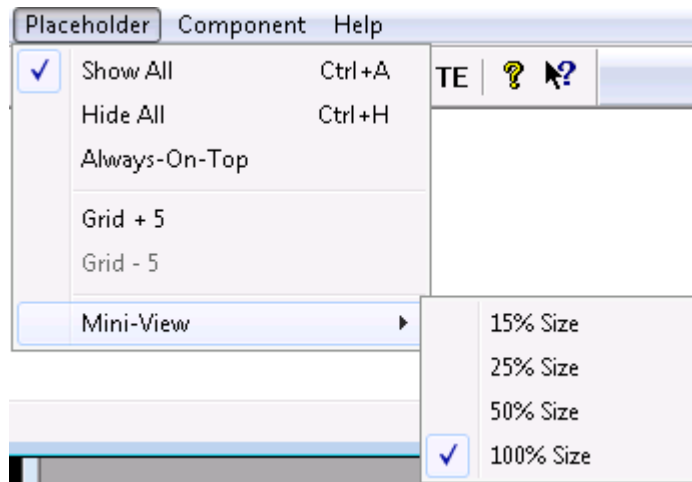
- 1 On the active workspace, select the tree item you want to copy.
- 2 Click the  icon on the GWE toolbar.
Or,
On the **SafeView Editor** menu bar, click *Edit*.
The **Edit** drop-down menu appears.
- 3 On the **Edit** drop-down menu, click **Copy** (or press **Ctrl+C** simultaneously).
The selected tree item is saved so that it will be available for pasting during the current GWE session.

7.4.16 Pasting a tree item

- 1 On the active workspace, select the tree item you want to paste.
- 2 Save the tree item you want to paste using either the **Cut** or **Copy** function.
- 3 Select a tree item to indicate where you want the saved item to be placed.
The item will be highlighted.
- 4 Click the  icon on the GWE toolbar.
Or,
On the **SafeView Editor** menu bar, click *Edit*.
The **Edit** drop-down menu appears.
- 5 On the **Edit** drop-down menu, click **Paste** (or press **Ctrl+V** simultaneously).
The last cut or copied item will appear on the active workspace immediately *below* the highlighted item.

7.4.17 Displaying the GWE placeholder menu

- On the **SafeView Editor** menu bar, click **Placeholder** to open its drop-down menu. The **Mini-View** option allows convenient visualization of a multi-screen workspace on a single screen workstation or laptop.



7.4.18 Enabling/disabling the ALWAYS ON TOP property

You can configure placeholders to be (or not be) “always on top” by using the Always-On-Top option buttons on the Window Style page (from the Window Specification window). On the other hand, you can enable the always on top behavior at edit time by using the Always-On-Top command in the Placeholder menu. This

command does not change the specifications that define the window, the SafeView run time, or the associated .wdl file. It only changes the behavior exhibited by placeholders at edit time.

By default, the ALWAYS ON TOP property is disabled (not checked) to avoid covering a display with an Always-On-Top placeholder window.

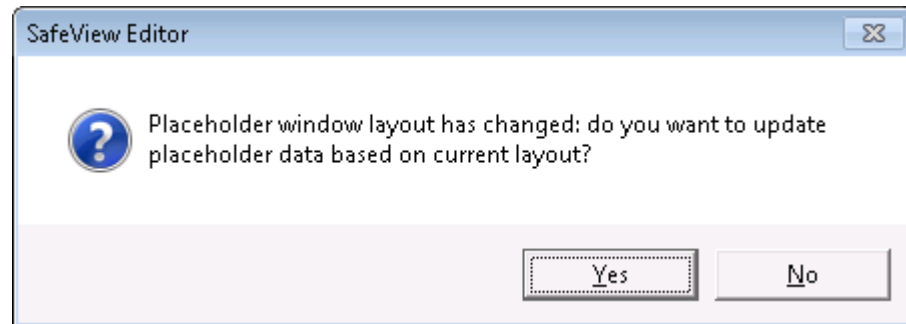
To enable/disable the ALWAYS ON TOP property, do the following procedure.

- 1 On the **SafeView Editor** menu bar, click **Placeholder** to open its drop-down menu.
- 2 Look at the **Always-On-Top** item. If it has a checkmark, the property is currently enabled. If it does not, it is disabled.
Click the **Always-On-Top** item as appropriate to enable it (add checkmark) or disable it (remove checkmark).




Tip

If you manually resize or reposition the placeholder for a window and then select the Always-On-Top command, the following dialog box will appear:



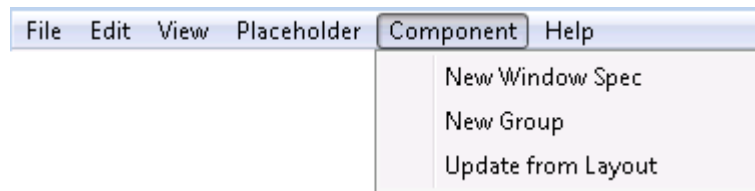
7.4.19 Enlarging/reducing the placeholder display grid

A placeholder is an invisible, expandable display grid used to help you align and position windows. When the GWE is initially launched, the placeholder grid size is set to one pixel, the default grid size. Any changes you make to the default grid size remain in effect only while the GWE is on. To enlarge/reduce the placeholder display grid, do the following procedure:

- 1 Click the  icon on the GWE toolbar.
Or,
On the **SafeView Editor** menu bar, click **Placeholder**.
The **Placeholder** drop-down menu appears.
- 2 On the **Placeholder** drop-down menu, click **Grid + 5**.
The minimum increment for placeholder movement and resizing is increased by five pixels each time the icon is selected until a maximum size of fifty pixels is reached.
- 3 On the **Placeholder** drop-down menu, click **Grid - 5**.
The minimum increment for placeholder movement and resizing is decreased by five pixels each time the icon is selected until a minimum size of one pixel (the default grid size) is reached.

7.4.20 Displaying the GWE component menu

- On the **SafeView Editor** menu bar, click **Component** to open its drop-down menu.



7.4.21 Showing/hiding placeholders

When a workspace is configured, specifying a window placeholder is optional. The show/hide placeholder functions are operational only when one or more of the windows in the workspace have placeholders configured (**PLACEHOLDER = yes**). To show/hide placeholders, do the following procedure:

To show/hide placeholders:

- 1 On the **SafeView Editor** menu bar, click **Placeholder** to open its drop-down menu.
- 2 On the **Placeholder** menu, click **Hide All**
The window placeholders disappear from the workspace currently having placeholders configured.
- 3 On the **Placeholder** menu, click **Show All**.
The window placeholders appear over the active workspace currently having placeholders configured.




Attention

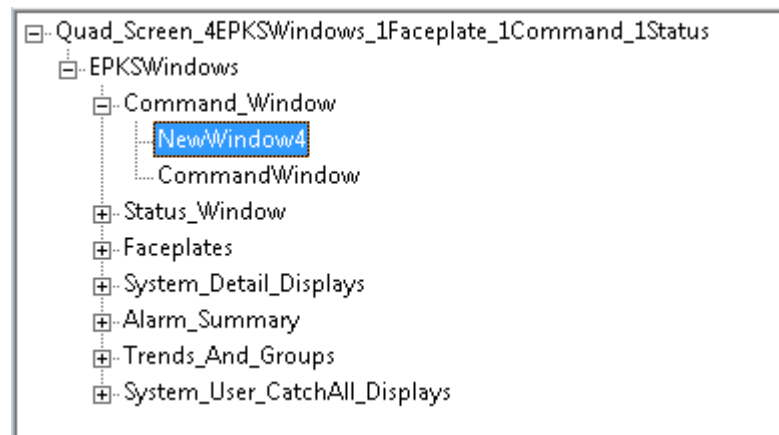
The show/hide placeholder option only affects the visibility of configured placeholders. If only a subset of windows has **PLACEHOLDER = yes**, then only those placeholders will be affected. If none of the windows have configured placeholders, the hide/show placeholder functions have no apparent effect until you subsequently select **PLACEHOLDER = yes**.

Results


<replace with description of results>

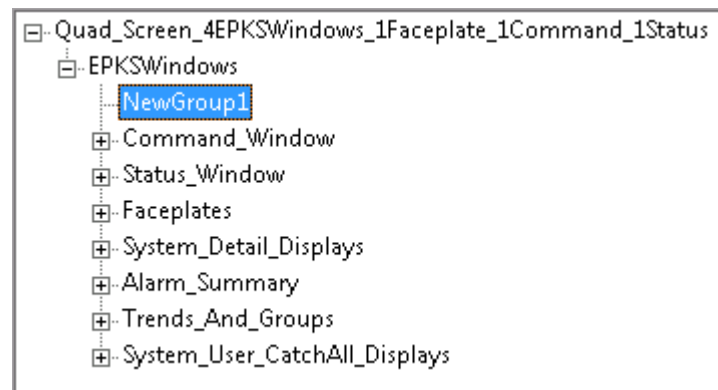
7.4.22 Adding a new window specification

- 1 Select a window tree item to indicate where you want the new window specification to be placed.
The item will be highlighted.
- 2 Click the  icon on the GWE toolbar.
Or,
On the **SafeView Editor** menu bar, click **Component**.
The **Component** drop-down menu appears.
- 3 On the **Component** drop-down menu, click **New Window Spec** (or press **Ctrl+N** simultaneously).
A new window specification item titled "NewWindow4" is added to the active workspace, immediately *below* the selected item.




7.4.23 Adding a new group

- 1 Select a first match group item to indicate where you want the new group to be placed.
The item will be highlighted.
- 2 Click the  icon on the GWE toolbar.
Or,
On the **SafeView Editor** menu bar, click **Component**.
The **Component** drop-down menu appears.
- 3 On the **Component** drop-down menu, click **New Group** (or press **Ctrl+G** simultaneously).
A new group item titled “NewGroup1” is added to the active workspace, as the first item in the selected first match group.



7.4.24 Updating window layout data from placeholders

- 1 Click the  icon on the GWE toolbar.
Or,
On the **SafeView Editor** menu bar, click **Component**.
The **Component** drop-down menu appears.
- 2 On the **Component** drop-down menu, click **Update from Layout**.
A **SafeView Editor** dialog box appears.
- 3 Click **OK** to update the layout.
The window layout data is copied from all the placeholders currently specified to the workspace.

7.4.25 Editing techniques

“In-place” name editing

The names of the tree items may be edited “in place.” Click the tree item to highlight it and then perform the normal Windows editing techniques. Each tree item is named, or labeled, with the name of the associated window group or specification. Thus, in-place tree item name editing provides a means to rename window groups and specifications.

“Drag and drop” editing

Drag and drop editing provides the means for the operator to drag a workspace component from one position in the hierarchy, and drop it (move or copy) into another place in the hierarchy.

7.4.26 Property windows

To edit the workspaces, SafeView provides the **Property**, **Window Group**, and **Window Specification** windows.

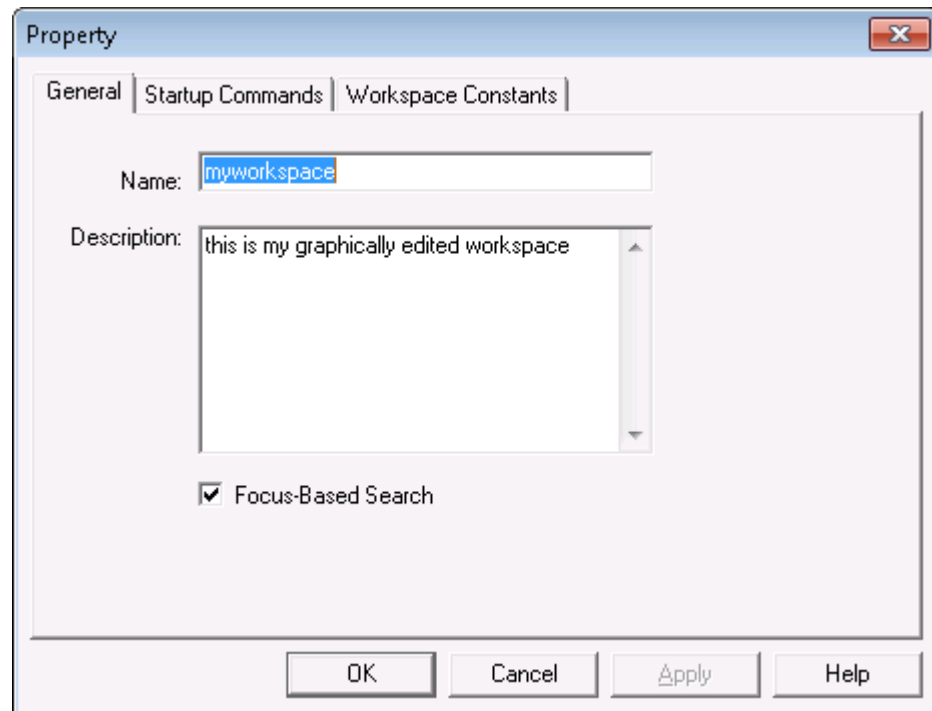
- The **Property** window has the **General**, **Startup Commands**, and **Workspace Constants** pages.
- The **Window Group** window has the **General** and **Group Default Styles** pages.
- The **Window Specification** window has the **General** and **Window Style** pages.

Apply button

The **Apply** button (on the bottom of each **Property** window) is used to apply the current property modifications to the workspace. This function is *not* implemented in this release; thus, the button remains unavailable (dimmed) on all **Property** windows.

7.4.27 Editing the workspace general properties

- 1 Open the desired workspace.
A tree view of the workspace appears in **SafeView Editor** window.
- 2 Double-click the top tree view item, the workspace name.
The **Property** window appears.
- 3 Click the **General** tab to open the associated page.



- 4 Edit the workspace properties as needed. The operations you can do include:
 - Change workspace name by typing a new name in the **Name** edit window.
 - Change the workspace description by typing a new description in the **Description** edit window.
 - Enable/disable the **Focus-Based Search** option by selecting or clearing the check box, depending on the search mode wanted at run time.
- 5 If you want to see how the workspace looks when the new general properties are applied without leaving the **General** page, click **Apply**.
NOTE: Clicking **Cancel** does not undo applied changes.
- 6 Click **OK**.
The **Property** window disappears. If you changed the workspace name, the new name item will be displayed in the tree view of the workspace.
- 7 Check the workspace syntax and save the edited workspace.

Associated .wdl file text

The .wdl file for the sample workspace represented by the preceding **Property** page would contain the following lines of text:

```
//***Safeview workspace Configuration File***
workspace myworkspace
description = "this is my graphically edited workspace";
focusbasedsearch=yes;
```

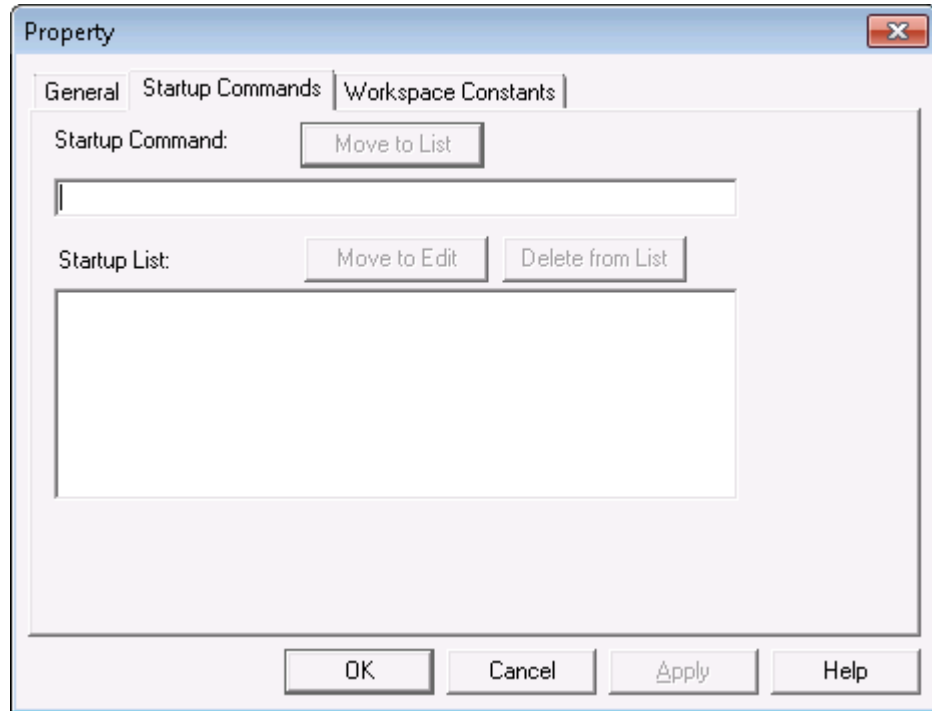
7.4.28 Editing the workspace startup commands

While SafeView issues requests to start programs based on the order in which you enter the startup commands, do not assume that the programs will appear in that same order. It is the operating system, and not SafeView, that determines the order in which the programs actually appear.

For startup command examples, see ““startup_specification”.”

To edit the workspace start-up commands, do the following procedure:

- 1 Open the desired workspace.
A tree view of the workspace appears in the **SafeView Editor** window.
- 2 Double-click the top tree view item, the workspace name.
The **Property** window appears.
- 3 Click the **Startup Commands** tab to open the associated page.



- 4 Edit the workspace properties as needed. The operations you can do include:
 - Create a new startup command by typing it in the **Startup Command** edit window.
 - Move a startup command from the list into the edit window by selecting it and then clicking **Move to Edit**.
 - Move a startup command from the edit window into the list by selecting it and then clicking **Move to List**.
 - Delete a startup command from the list by selecting it and then clicking **Delete from List**.
- 5 Click **OK**.
The **Property** window disappears.
- 6 Check the workspace syntax and save the edited workspace.
The `.wdl` file would contain the following lines of text after successfully adding the new startup command.



Tip

Each single backslash (\) is changed to two backslashes (\\) to disable the “escape” meaning normally assigned to a single backslash.

Results

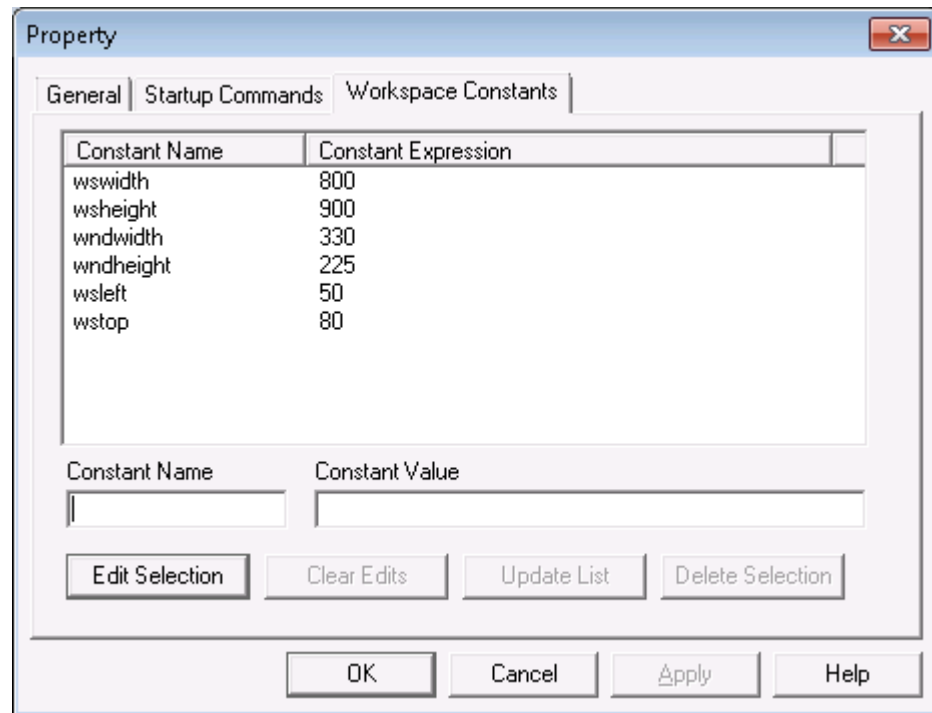
```
//User defined application invocations
startup
//The startup commands would appear here
end startup
```

7.4.29 Editing the workspace constants

Although string constants are supported in the TextEdit environment, they are not supported (nor considered necessary) in the graphical environment. Workspaces with string constants are correctly loaded into the graphical environment, but the constants are permanently replaced with their literal values.

To edit string constants

- 1 Open the desired workspace.
A tree view of the workspace appears in the **SafeView Editor** window.
- 2 Double-click the top tree view item, the workspace name.
The **Property** window appears.
- 3 Click the **Workspace Constants** tab to open the associated page.



- 4 To create a new workspace constant, do the following steps.
 - a Type the name of the new workspace constant in the **Constant Name** edit window.
 - b Type the value of the new workspace constant in the **Constant Value** edit window.
 - c If you want to clear both the **Constant Name** and **Constant Value** edit windows, click **Clear Edits**.
 - d Click **Update List**. The name and value of the new workspace constant appear in the **Workspace Constants** list.

You can also create a new workspace constant by selecting an existing constant and then changing its name. This will *not* rename the existing constant, but will in fact create a new one. To do this, go to step 5.

- 5 To change the value of a workspace constant, do the following steps:
 - a Select the name of the constant from the **Constant Name** list, and then click **Edit Selection**. The name and value of the selected constant appear in the **Constant Name** and **Constant Value** edit windows.
 - b Type the appropriate value in the **Constant Value** edit window.
 - c If you want to clear *both* the **Constant Name** and **Constant Value** edit windows, click **Clear Edits**.
 - d Click **Update List**. The new value appears in the **Workspace Constants** list.

- 6 To delete a workspace constant from the list, select its name from the **Workspace Constants** list, and then click **Delete Selection**.
- 7 If you want to see how the workspace looks when the new constants are applied without leaving the **Workspace Constants** tab, click **Apply**.
NOTE: Clicking **Cancel** does not undo applied changes.
- 8 Click **OK**.
- 9 Click **Yes** to apply your changes to the workspace.
All current edits including the values in the edit fields will be saved and the **Update** dialog box and the **Property** window will disappear. But, If a problem is detected with any of the edited values, such as an illegal constant name or value, an error detection message will appear. The **Update** dialog box and the **Property** window will remain open.
- 10 Check the workspace syntax and save the edited workspace.

Associated .wdl file text

The .wdl file for the sample workspace represented by the preceding **Property** page contains the following lines of text:

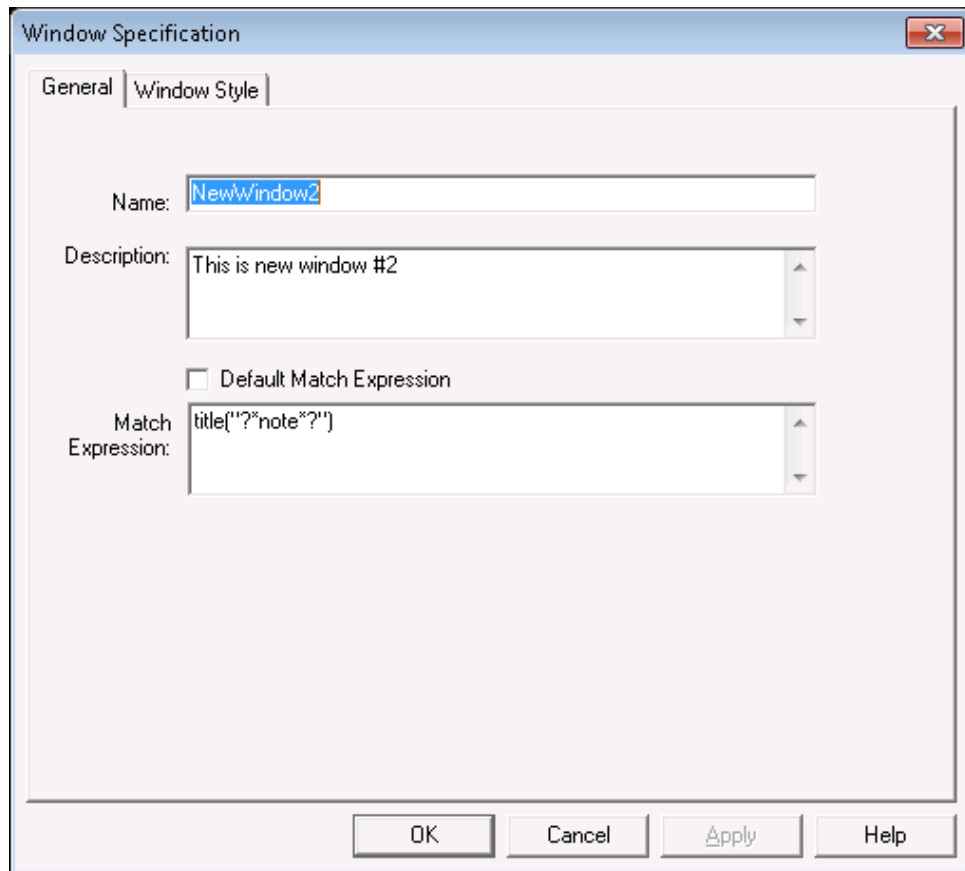
```
//User defined constants
number wswidth = 800;
number wsheight = 900;
number wndwidth = 330;
number wndheight = 225;
number wsleft = 50;
number wstop = 80;
```

7.4.30 Editing the window general properties using the Window Specification window

The .wdl file for the sample workspace represented by the preceding **Window Specification** page contains the following lines of text:

```
window NewWindow2
description = "This is new window #2";
match = title("?*note?*");
...
end window
```

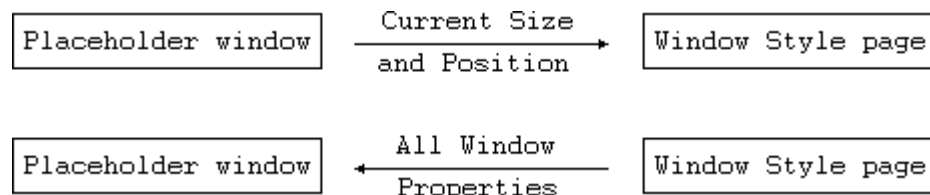
- 1 Open the desired workspace. A tree view of the workspace appears in the **SafeView Editor** window.
- 2 Navigate the workspace hierarchy as required until the tree view item for desired window is displayed.
- 3 Double-click the tree view item representing the name of the desired window. The **Window Specification** window is displayed.
- 4 Click the **General** tab to open the associated page.



- 5 Edit the workspace properties as needed. The operations you can do include:
 - Change the window name by typing a new name in the **Name** edit window.
 - Change the window description by typing a new description in the **Description** edit window.
 - Change the **Match Expression** for this window if it's in a first match group.
 - Designate the current MATCH expression as the default match by selecting the **Default Match Expression** check box. You can only do this function if no default MATCH expression is currently specified for the workspace.
- 6 Click **OK**. The **Window Specification** window disappears. If you changed the window name, the new name appears in the tree view of the workspace.
- 7 Check the workspace syntax and save the edited workspace.

When the **PLACEHOLDER** property for a window is enabled, you can edit the window graphically using placeholders. A placeholder is displayed on the screen in accordance with its **POSITION** and **SIZE** properties. A window's style and properties are listed on its **Window Style** page, which is shown in “Editing the window style using the Window Specification window” on page 92.”

You can transfer window property data between a placeholder and its **Window Style** page in *both* directions, as shown in the following diagram:



The **Apply** button on the **Window Style** page is used to move the current position and size from the placeholder to the **Window Style** page.

The **Apply** button on the **Window Style** page is used to move *all* the window properties from the **Window Style** page to the placeholder, not just size and position. For example, if you configure a display to make it draggable, the placeholder will also be draggable.

7.4.31 Editing the group default styles using the Window Group window

Group default values are the values that will be used by SafeView for all window properties whose values have not been specifically defined in the specifications for the window. Group default values can only be represented by numerics or user-defined constants. Numeric expressions are not supported.

- 1 Open the desired workspace. A tree view of the workspace appears in the **SafeView Editor** window.
- 2 Navigate the workspace hierarchy as required until the tree view item for desired group is displayed.
- 3 Double-click the tree view item representing the name of the desired group. The **Window Group** window appears.
- 4 Click the **Group Default Styles** tab to open the associated page.

- 5 The **Restrict Windows to Region** style is the *only* one available for the current version of the software. Edit this group property as needed. The operations you can do are:
 - Change the definition of a region by typing new values in the **X**, **Y**, **Width**, and **Height** boxes in the **Region** section.
 - Set up a new region by selecting the **Restrict Windows to Region** check box and then typing new values in the **X**, **Y**, **Width**, and **Height** boxes in the **Region** section.
- 6 Click **OK**. The **Window Group** window disappears.
- 7 Check the workspace syntax and save the edited workspace.

Associated .wdl file text

The .wdl file for the sample workspace represented by the preceding **Window Group** page contains the following lines of text:

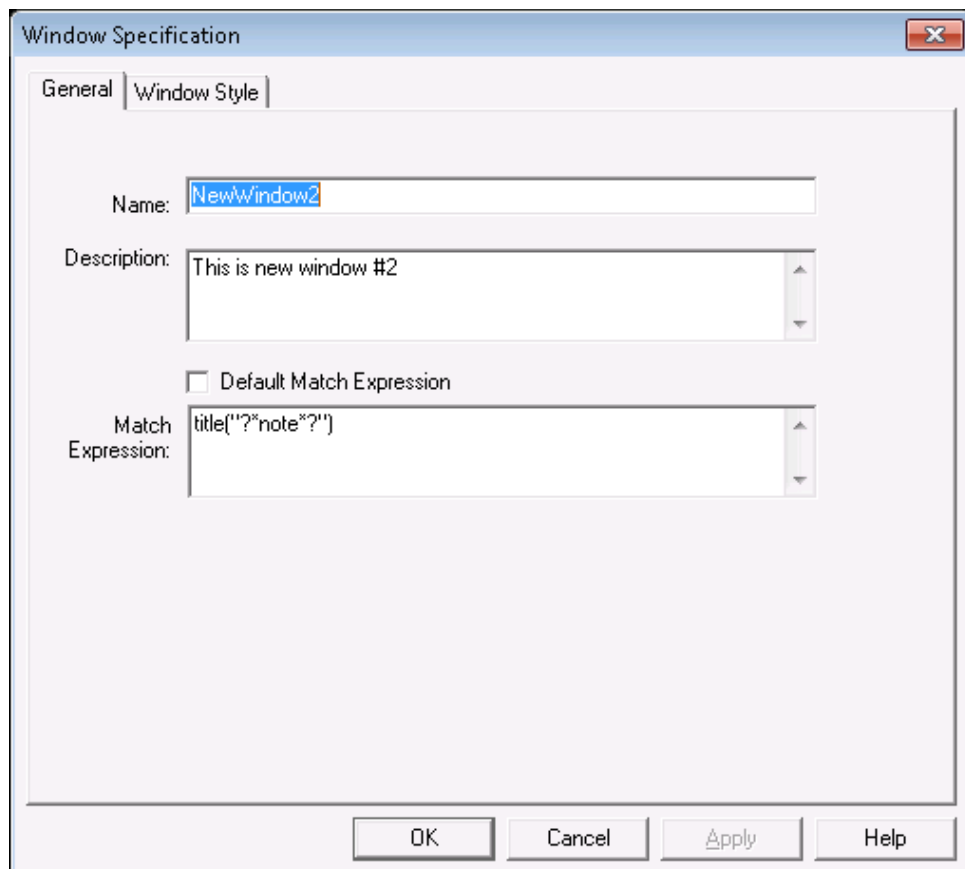
```
group RoundRobin1 ("round robin")
...
region = wsleft, wstop, wswidth, wsheight;
...
end group
```

7.4.32 Editing the window general properties using the Window Specification window

The .wdl file for the sample workspace represented by the preceding **Window Specification** page contains the following lines of text:

```
window NewWindow2
description = "This is new window #2";
match = title("?*note*");
...
end window
```

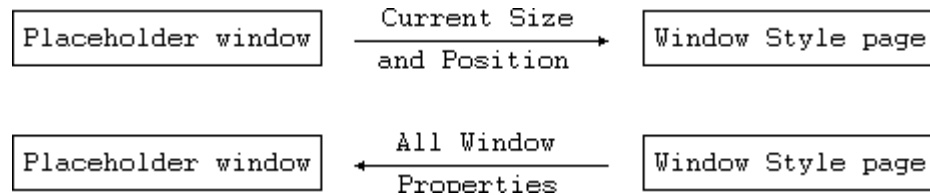
- 1 Open the desired workspace. A tree view of the workspace appears in the **SafeView Editor** window.
- 2 Navigate the workspace hierarchy as required until the tree view item for desired window is displayed.
- 3 Double-click the tree view item representing the name of the desired window. The **Window Specification** window is displayed.
- 4 Click the **General** tab to open the associated page.



- 5 Edit the workspace properties as needed. The operations you can do include:
 - Change the window name by typing a new name in the **Name** edit window.
 - Change the window description by typing a new description in the **Description** edit window.
 - Change the **Match Expression** for this window if it's in a first match group.
 - Designate the current MATCH expression as the default match by selecting the **Default Match Expression** check box. You can only do this function if no default MATCH expression is currently specified for the workspace.
- 6 Click **OK**. The **Window Specification** window disappears. If you changed the window name, the new name appears in the tree view of the workspace.
- 7 Check the workspace syntax and save the edited workspace.

When the **PLACEHOLDER** property for a window is enabled, you can edit the window graphically using placeholders. A placeholder is displayed on the screen in accordance with its **POSITION** and **SIZE** properties. A window's style and properties are listed on its **Window Style** page, which is shown in “Editing the window style using the Window Specification window” on page 92.”

You can transfer window property data between a placeholder and its **Window Style** page in *both* directions, as shown in the following diagram:



The **Apply** button on the **Window Style** page is used to move the current position and size from the placeholder to the **Window Style** page.

The **Apply** button on the **Window Style** page is used to move *all* the window properties from the **Window Style** page to the placeholder, not just size and position. For example, if you configure a display to make it draggable, the placeholder will also be draggable.

7.4.33 Editing the window style using the Window Specification window

The following procedure describes how to manipulate a window *exclusively* through the **Window Style** page of the **Window Specification** window. (To manipulate a window *manually* using its placeholder, see “Editing the window specifications graphically” on page 94.”)

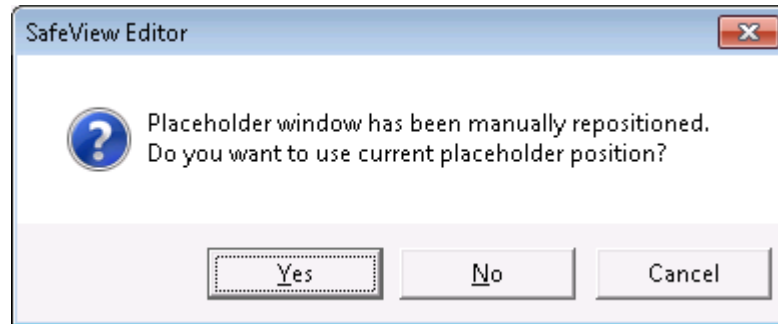
- 1 Open the desired workspace. A tree view of the workspace appears in the **SafeView Editor** window.
- 2 Navigate the workspace hierarchy until a tree view of the item you want to edit appears.
- 3 Double-click the tree view item representing the name of the desired window. The **Window Specification** window appears.
- 4 Click the **Window Style** tab to open the associated page.

- 5 Edit the style properties as needed. The operations you can do include:
 - Designate any of the window properties listed in the **Styles** section as configured (or not configured) by selecting the appropriate check boxes. A checked box means that the property is configured, and an unchecked box means it is not.
 - Define the initial position of the window by selecting the **Set Init Position** check box, if required, and then typing new **X** and **Y** values in the **Initial Position** section.
 - Define the initial size of the window by selecting the **Set Init Size** check box, if required, and then typing new **Width** and **Height** values in the **Initial Size** section.
 - Define the minimum size of the window by selecting the **Restrict Min Size** check box, if required, and then typing new **Width** and **Height** values in the **Minimum Size** section.
 - Define the maximum size of the window by checking the **Restrict Max Size** check box, if required, and then typing new **Width** and **Height** values in the **Maximum Size** section.
 - Define the region for the window to override the region defined in the containing group by selecting the **Override Group Region** check box, if required, and then typing new values into the **X**, **Y**, **Width**, and **Height** fields in the **Group Region Override** section.
 - Define the maximized position of the window by selecting the **Set Maximized Origin** check box, if required, and then typing new **X** and **Y** values in the **Maximum Position** section.
 - Change the sizeable property by selecting the applicable option button in the **Sizeable** section.
 - Change the ALWAYS ON TOP property by selecting the applicable option button in the **Always On Top** section.
- 6 To discard your changes and restore all the original property selections, click **Cancel**.
- 7 If any of properties you have configured affect how a placeholder is displayed and you want to update the placeholder display, click **Apply**.

NOTE: Selection of any of the following properties will affect a placeholder display and will activate the **Apply** button: **Set Init Position**, **Set Init Size**, **Draggable**, **Restrict Min Size**, **Restrict Max Size**, **Override Group Region**, **Sizeable**, and **Always On Top**.

- 8 Click **OK**. The **Window Specification** window disappears.
- 9 Check the workspace syntax and save the edited workspace.

NOTE: If you manually resize or reposition the placeholder for a window and click either **OK** or **Apply** on the **Window Style** page without executing the “Update From Placeholder” procedure, the following dialog box will appear:



- 10 Click **Yes** to update the values in the **Window Style** page to reflect the placeholder's current position and size.
Or,
Click **No** to modify the position and size of the placeholder based on the values currently shown in the **Window Style** page.

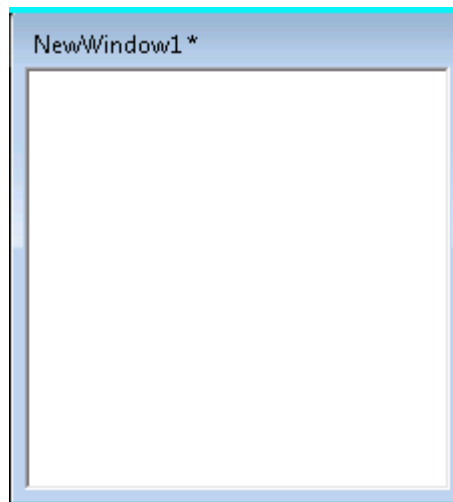
7.4.34 Editing the window specifications graphically

The graphical (manual) manipulation of a window is accomplished using its placeholder. If all the windows in a workspace are configured to have placeholders and if the sizes and positions for all windows are defined appropriately using user-defined constants, you can modify the entire workspace simply by editing just one window and then redefining its associated constant values, such as *myWindWidth* or *myWindHeight*. The procedure that follows explains how to perform this “broadcasting” function in detail.

- 1 Open the desired workspace. A tree view of the workspace appears in the **SafeView Editor** window. To understand how the text file for a workspace can be changed by this procedure, here is how lines that can be altered by this procedure appear in the file *initially*:

```
//User defined constants
...
...
number wndwidth = 330;
number wndheight = 225;
number wsleft = 50;
number wstop = 80;
```

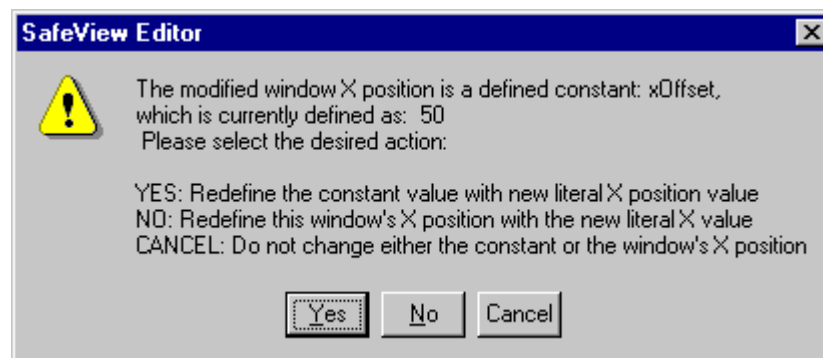
- 2 Minimize the **SafeView Editor** window if the placeholder for the window you want to edit is not visible.
- 3 Select the desired **Placeholder**. The title bar for the selected placeholder is highlighted.
- 4 Edit the placeholder by resizing and/or repositioning it.
If an asterisk (*) is after the placeholder title, the current size and position of the placeholder do not match the values in the **Window Style** page.



- 5 Double-click the edited placeholder to open the **Window Specification** window.
- 6 On the **Window Specification** window, click the **Window Style** tab to open the associated page. Note the contents of the **Initial Position** and **Initial Size** text boxes, which contain the original values.

| Initial Position | | Initial Size | |
|------------------|--------|--------------|-----------|
| X: | wsleft | Width: | wndwidth |
| Y: | wstop | Height: | wndheight |

- 7 Click **Update From Placeholder**. A dialog box appears, listing a **Window Style** page value that needs to be updated and offering three optional procedures, if the value includes a defined constant.



- 8 To recalculate the constant value for the property, do the following steps:
 - a Click **Yes**. The dialog box disappears.
 - b If another **Window Style** page value needs to be updated, a dialog box similar to the one shown in step 7 will appear, listing that value. Keep clicking **Yes** until no additional dialog boxes appear.
 - c Click **OK** on the **Window Style** page. The page will disappear, leaving the tree view of this workspace, with *all* placeholders that use any of the recalculated constant values resized and repositioned accordingly.
- 9 To recalculate and replace the constant value with a new literal value, do the following steps:
 - Click **No**. The dialog box disappears.
 - If another **Window Style** page value needs to be updated, a dialog box similar to the one shown in step 7 will appear, listing that value. Keep clicking **No** until no additional dialog boxes appear.
 - The **Window Style** page reappears. The contents of the **Initial Position** and **Initial Size** boxes have been changed to reflect the new literal values.

- Click **OK**. The **Window Style** page disappears, leaving the tree view of this workspace, with *all* placeholders that use any of recalculated constant values resized and repositioned accordingly.

| Initial Position | | Initial Size | |
|------------------|---|--------------|-----|
| X: | 4 | Width: | 349 |
| Y: | 4 | Height: | 248 |

- 10 To cancel the editing done earlier in Step 4, do the following steps:
- Click **Cancel**. The dialog box disappears.
 - If another **Window Style** page value needs to be updated, a dialog box similar to the one shown in step 7 will appear, listing that value. Keep clicking **Cancel** until no additional dialog boxes appear.
 - Click **Cancel**. The **Window Style** page disappears, leaving the tree view including the placeholders on display, in their initial format.

8 Managing a workspace (SafeView run time)

Related topics

“Functions overview” on page 98

“Command line options” on page 99

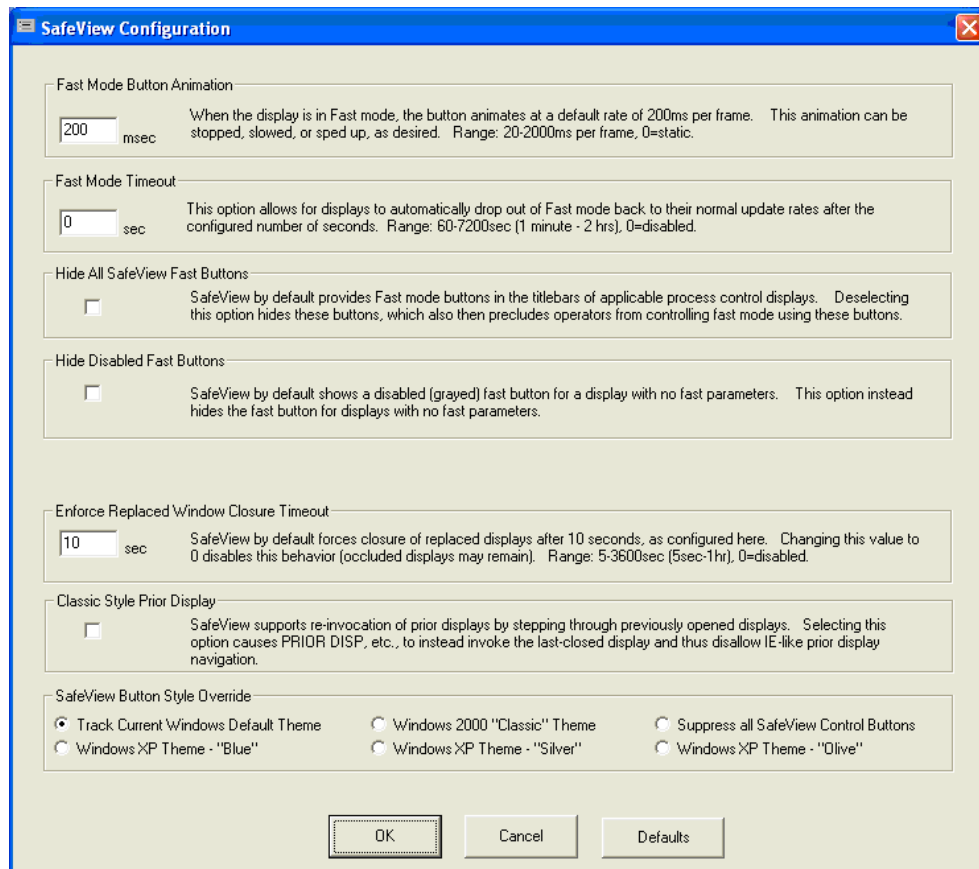
“Using SafeView to manage a workspace” on page 100

8.1 Functions overview

Once the system is properly set up, you can invoke SafeView in a number of ways, just as you would any executable program, such as through the file system or an application program, or by double-clicking a Program Item. The executable name is *SAFEVIEW.exe*.

With Experion R400, you can invoke the SafeView configuration from the Configuration Utility.

- Choose **Start > Programs > Honeywell Experion PKS > System Management**.
- Choose **Configure > SafeView > in the > Configuration utility > window**.



8.2 Command line options

SafeView may process the following command line options at startup. All erroneous command line options will be ignored. (Also see “SafeView's command line options” on page 261.)

| Command line option | Action |
|---------------------|---|
| /f <filename> | This option causes SafeView to attempt to load <filename>, expecting a file that represents a valid workspace configuration. Once a workspace configuration is successfully loaded, all subsequently invoked applications that can be managed according to this configuration are managed. (No space between /f and the filename) |
| /cs | This option causes SafeView to be case sensitive, with respect to the MATCH expression strings supplied in the workspace configuration. Typically, it is better to not be case-sensitive, which is the case if no /cs option is specified. |
| /mp | This option causes SafeView to attempt to manage pre-existing applications (those that existed at the time when SafeView was invoked) on each WDL load. |
| /cp | This option causes the Control Panel (see next option) to be hidden at startup of SafeView. The Control Panel can be displayed using the CTRL-ALT-W hot key. |
| /nw | This option hides the Control Panel and disables the ability to display the Control Panel. <i>Note that this option assumes that the Application Program Interface call, StopManager, will be used to stop SafeView.</i> |

8.3 Using SafeView to manage a workspace

8.3.1 Terminating SafeView management

When you terminate SafeView, it stops managing the application windows which then revert to their normal, unmanaged state. After terminating SafeView, if you maximize an application display that was formerly constrained under SafeView, it will again function normally and occupy the full screen. If SafeView crashes, applications will still run.

8.3.2 Starting SafeView

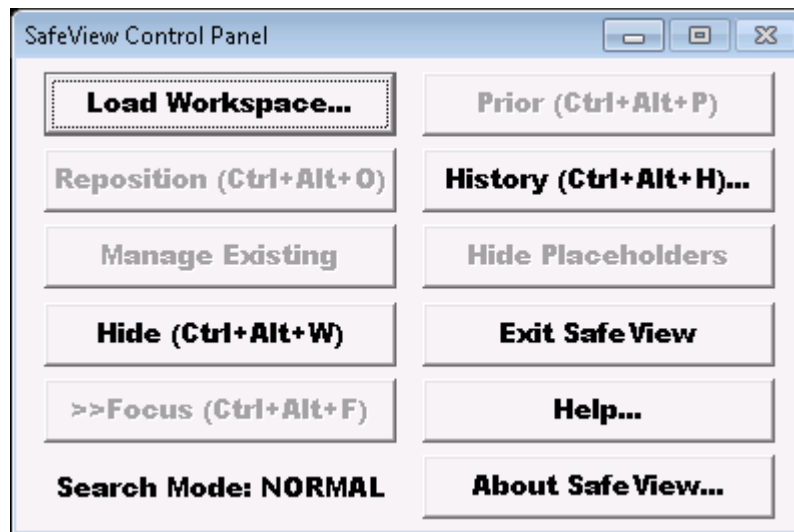
The **SafeView Control Panel** is the first menu to appear when SafeView is launched. It performs many workspace management functions. The following procedure describes how to call up the **SafeView Control Panel** from a Windows system with the SafeView package installed in accordance with Honeywell recommendations.

- Click **Start > Programs > Honeywell Experion PKS > SafeView > SafeView Runtime**.
Or,

Click **Start > Programs > Honeywell SafeView > SafeView**.

The **SafeView Control Panel** appears.

This example shows the **SafeView Control Panel** *before* any workspaces are loaded. Unavailable buttons are dimmed while all others are available.



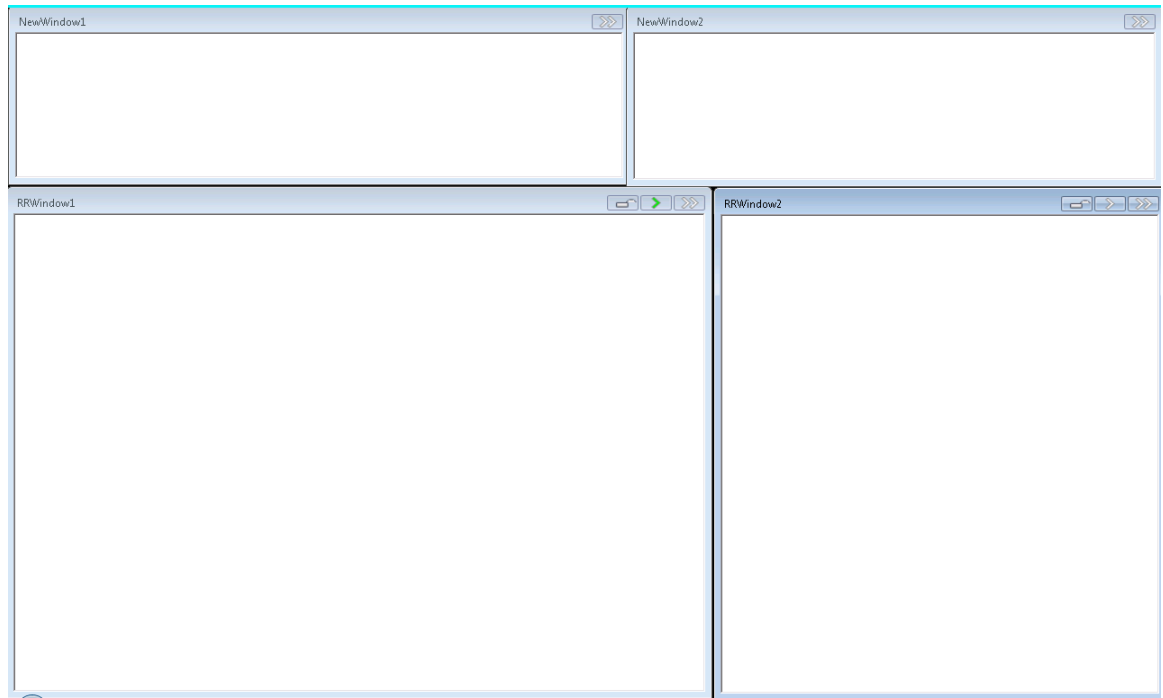
8.3.3 Loading a workspace

Loading a workspace is the procedure you use to get a specific workspace displayed on an operator station screen. Before you can load a workspace, it must be configured.

- On the **SafeView Control Panel**, click **Load Workspace**.
The **Open** dialog box appears.
- Select the directory and file name of the desired workspace configuration file.
- If you want to cancel this procedure at this point, click **Cancel**.
The procedure is aborted and the **Open** dialog box disappears.

4 Click **OK** to load the workspace.

The selected workspace is loaded and appears on the operator station, and the Control Panel is updated. The following is an example of a loaded workspace. For more examples, see ““Safeview workspace examples” on page 153.”



Control Panel with loaded workspace

- The following figure shows an example of the updated Control Panel. In the example,
 - The name of the configuration file (*SafeView1.wdl*) appears in the title bar.
 - All of the buttons are activated except **Prior (Ctrl+Alt+P)** and **Hide Placeholders**. If the current workspace does not have configured placeholders, the **Hide Placeholders** button is unavailable (dimmed).
 - The **Focus/Normal** search mode buttons do not appear on earlier releases.

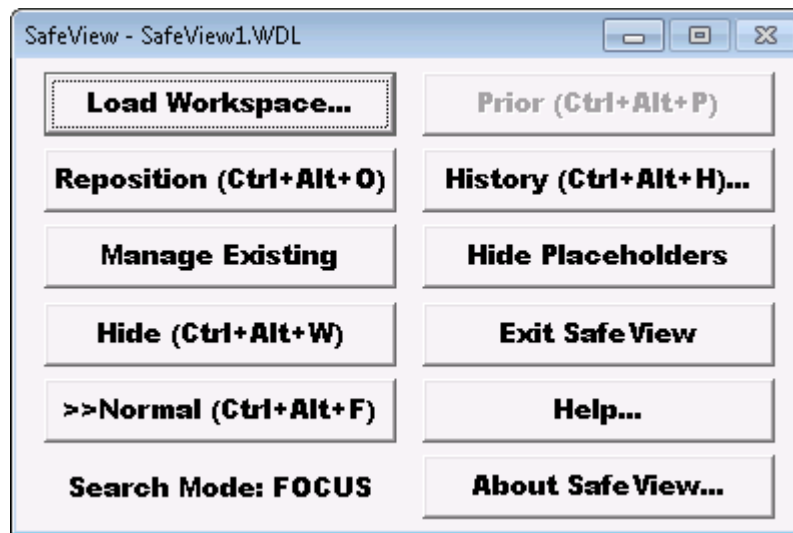


Figure 8: Example: Updated Control Panel

Configuration file load failure

- Referring to step 2 of “Loading a workspace” on page 100,” if the selected workspace configuration file is not syntactically correct, the file is not loaded and a failure message appears. Any workspace that was successfully loaded previously will remain on display. The following example shows a typical failure message.

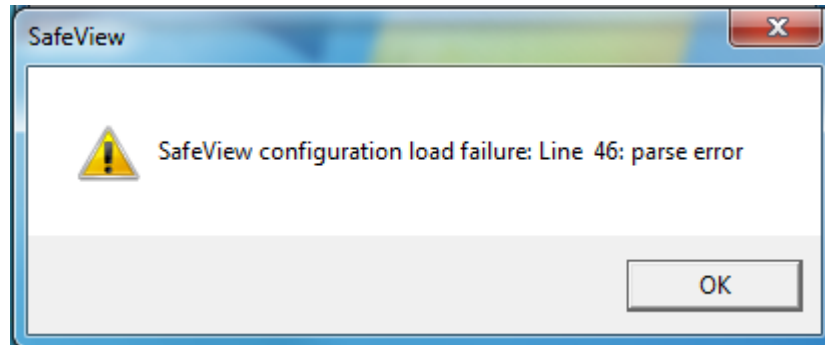


Figure 9: Example: Workspace Configuration Load Failure Message

8.3.4 Repositioning managed windows

Repositioning managed windows allows you to neaten a workspace after one or more displays have been moved, sized, iconized, or misplaced. To accomplish this, SafeView provides the **Reposition** command to automatically rearrange all application displays to their original locations.

- On the **SafeView Control Panel**, click **Reposition** (or press **Ctrl+Alt+O** simultaneously (the “O” signifies “original” position)).
All application displays are rearranged to their original locations.

8.3.5 Managing existing displays

“Existing” displays are application displays that were loaded before the SafeView workspace was configured using SafeView. Managing these displays puts them in their assigned locations by the after-the-fact workspace configuration. If registered displays predate the SafeView process, they will not be managed as registered (by category).

- On the **SafeView Control Panel**, click **Manage Existing**.
The displays for existing applications currently on the screen will be moved and resized, if required, into their respective placeholder locations and will now be managed by SafeView.
When you click **Manage Existing**, any minimized applications or displays will remain minimized and *not* managed within the SafeView workspace.

Example

Before configuring the workspace, you do the following steps:

- Load three displays which appear on your screen in various locations.
- Resize one or more of the displays.
- Configure the workspace and define placeholders for four displays (including the three displays already on the screen).
- Click the **Manage Existing** button.

The three displays are moved and resized, as required. The following figure shows how this workspace may appear.

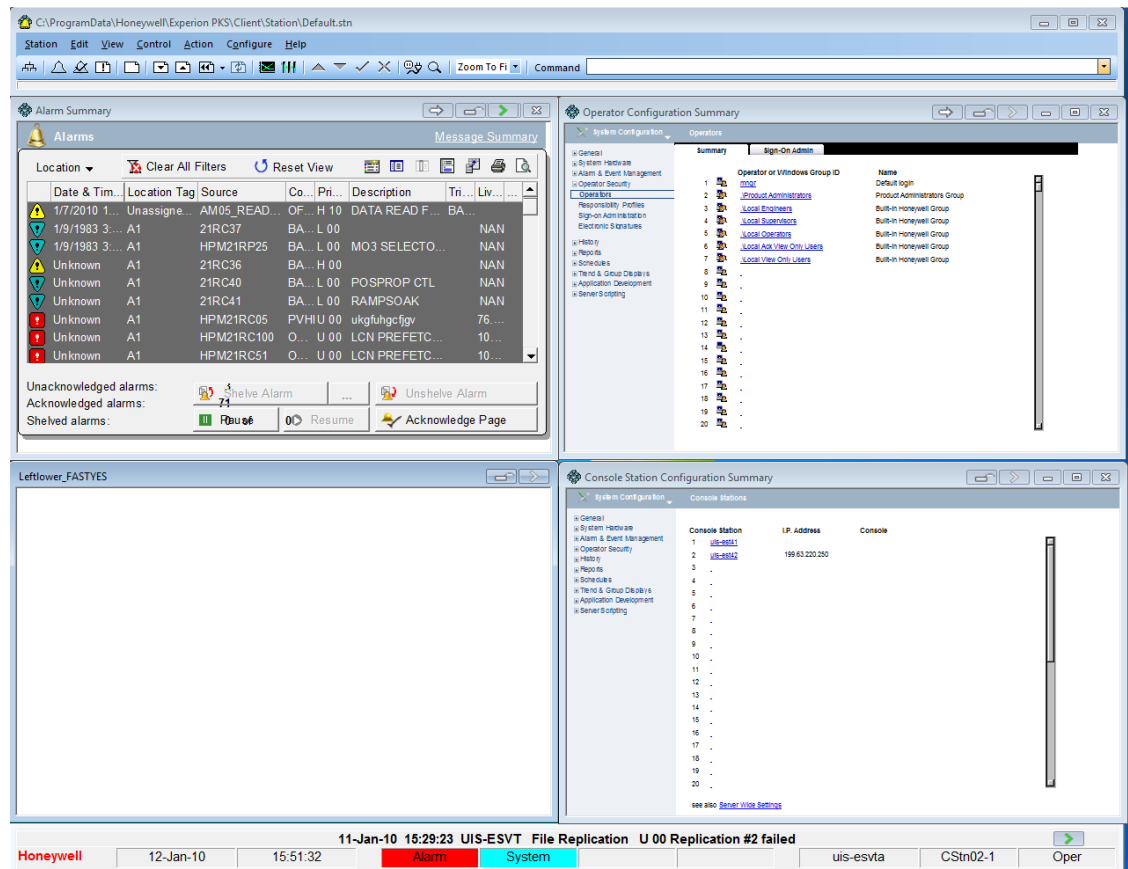


Figure 10: Example: Workspace with Three Displays

Manage existing manual select displays

- If more than one previously loaded application display is scheduled to be managed by a defined manual select group, click **Manage Existing** button. Each application display appears momentarily which is then replaced, except for the last one which remains on the screen.

8.3.6 Hiding/displaying the SafeView Control Panel menu

You can hide or display the SafeView Control Panel menu at any time.

- To hide the **SafeView Control Panel** menu, click **Hide** (or press **Ctrl+Alt+W** simultaneously). The menu will disappear.
- To display the **SafeView Control Panel** menu, press **Ctrl+Alt+W** simultaneously. The menu will appear.

8.3.7 Showing the prior display

The prior display is the last SafeView-managed display that was invoked in the current workspace.

- On the **SafeView Control Panel**, click **Prior** (or press **Ctrl+Alt+P** simultaneously). The most recently invoked managed operator station window is restored to its former location in the workspace.

8.3.8 Showing history: listing and invoking prior displays

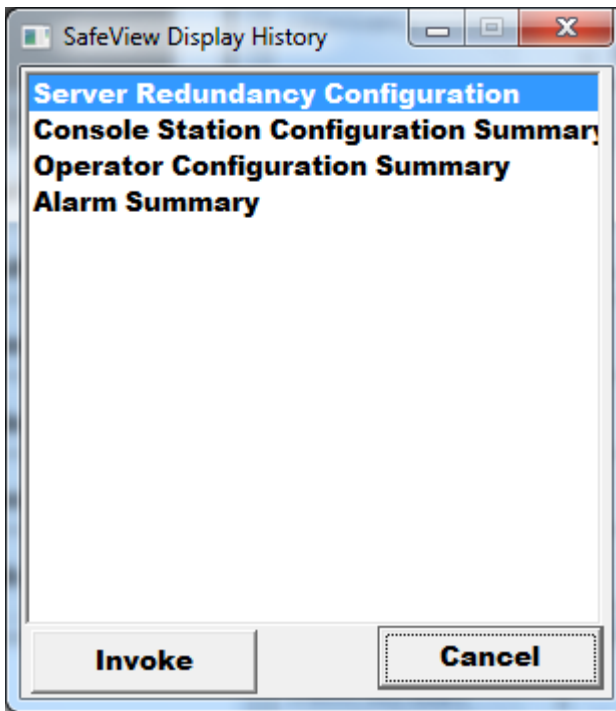
SafeView maintains a historical list of the registered application displays that have appeared in managed windows as they were invoked. Once you open this list, you can invoke any of the displays in the list. Note that registered windows that have crashed or terminated by the user will still appear in this list.

The oldest display appears at the bottom of the list and the newest one at the top. A maximum of fifty entries is maintained. When the list is full and a new window is invoked, the oldest entry in the list is removed to make space.

This dialog box can be resized and iconized. Vertical and horizontal scroll bars are provided if the contents do not fit in the displayed area. If you resize and/or move this dialog box, the original size and position are saved and the dialog box will be redisplayed as it was before it was removed.

- 1 On the **SafeView Control Panel**, click **History** (or press **Ctrl+Alt+H** simultaneously).

The **SafeView Display History** dialog box appears as follows. This dialog box lists twelve previously managed applications that have been invoked. When the list reaches its maximum of fifty entries, the oldest entry is removed from the list to make space.



- 2 If you want to invoke one of these applications again, select it and then click **Invoke** (or double-click the display name).
The selected application reappears.
- 3 Click **Cancel**.
- 4 The **SafeView Display History** dialog box disappears.

8.3.9 Errors

If the request to list prior displays fails, an error message appears that requires acknowledgement. The message explains the reason for the failure and shows the registered invocation command and the registered working directory.

The Microsoft Windows operating system command used to invoke prior displays (*CreateProcess*) does not access the working directory when it is searching for the application itself. The prior application (for example,

winword.exe) must therefore either be in the PATH environment variable or be explicitly defined in the registered invocation command, as shown the following sample error message.

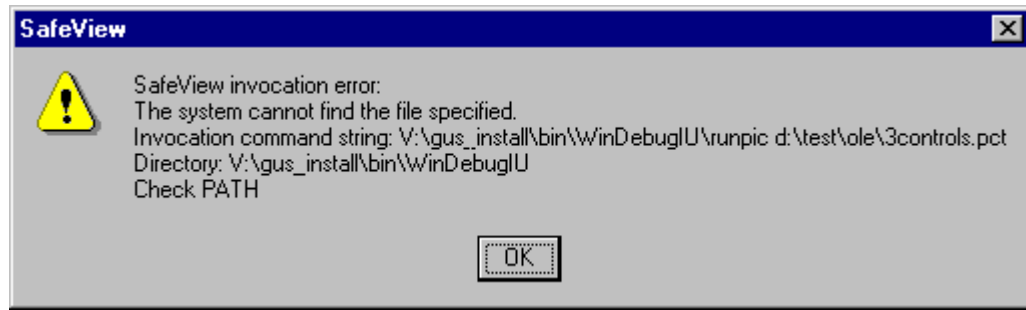


Figure 11: Example: Error Message

8.3.10 Showing/hiding placeholders

A “placeholder” is a blank window used to show where a display will appear on the screen when the SafeView is not actually managing an application display (such as a schematic) in a ViewPort. Placeholders are normally displayed only when a workspace is initially loaded. When managed applications are displayed, they replace the placeholders. If you close a managed application, the display disappears and a placeholder will reappear to fill the empty space.

When configuring a workspace, specifying placeholders for a window is *optional*. On the **SafeView Control Panel**, the **Show Placeholders** or **Hide Placeholders** button is available only if the current workspace is configured with at least one placeholder. If the current workspace has no placeholders, the button is unavailable (dimmed).

- On the **SafeView Control Panel**, click **Show Placeholders**.
Or,

On the **SafeView Control Panel**, click **Hide Placeholders**.

The placeholders disappear from the screen and the button label changes to **Show Placeholders**.

The following figure shows four placeholders as they appear immediately after loading. In this example screen, notice that only “Window1” has its output focus (>) button highlighted, indicating that it has the output focus. Notice also that the placeholders do not have Minimize, Maximize, or Close buttons.

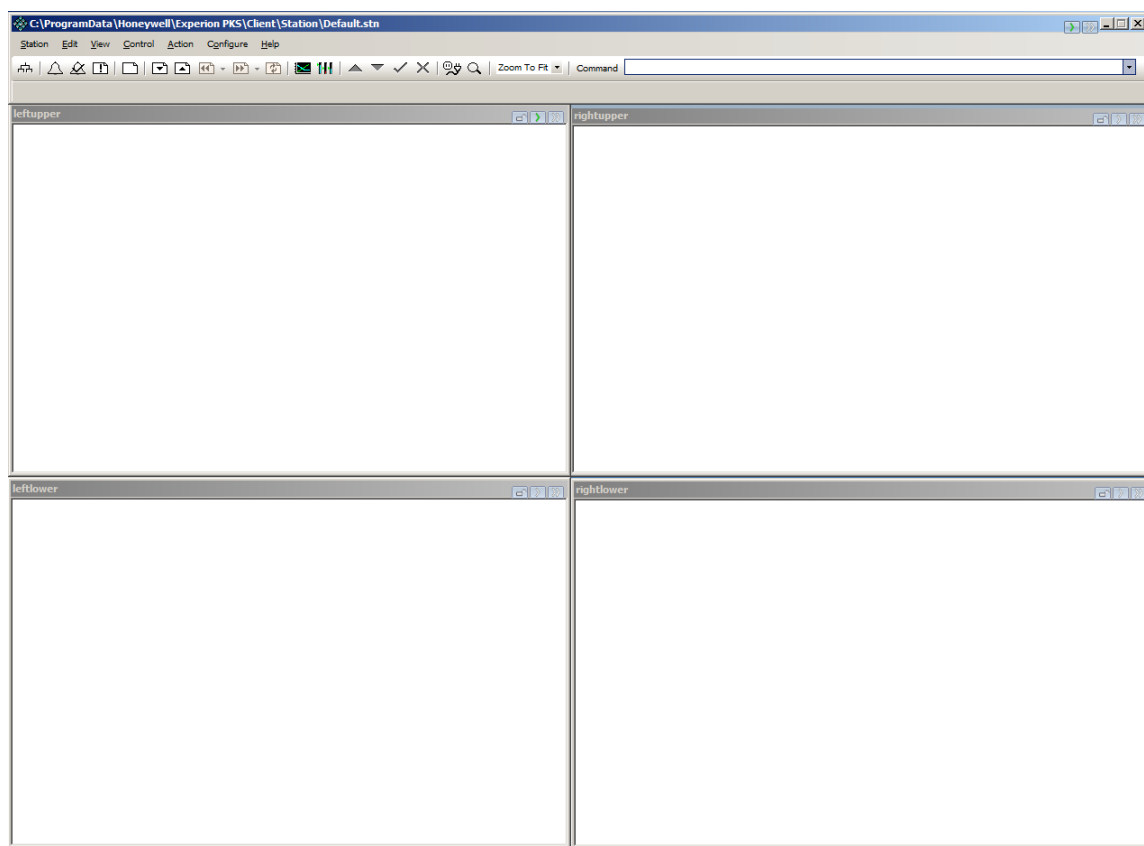


Figure 12: Example: Screen with Four Placeholders

8.3.11 Displaying information about SafeView

- 1 On the **SafeView Control Panel**, click **About SafeView**.
A dialog box appears with SafeView information (such as software version).
- 2 Review the information and then click **OK**.
The dialog box disappears.

8.3.12 Assigning output focus to a window

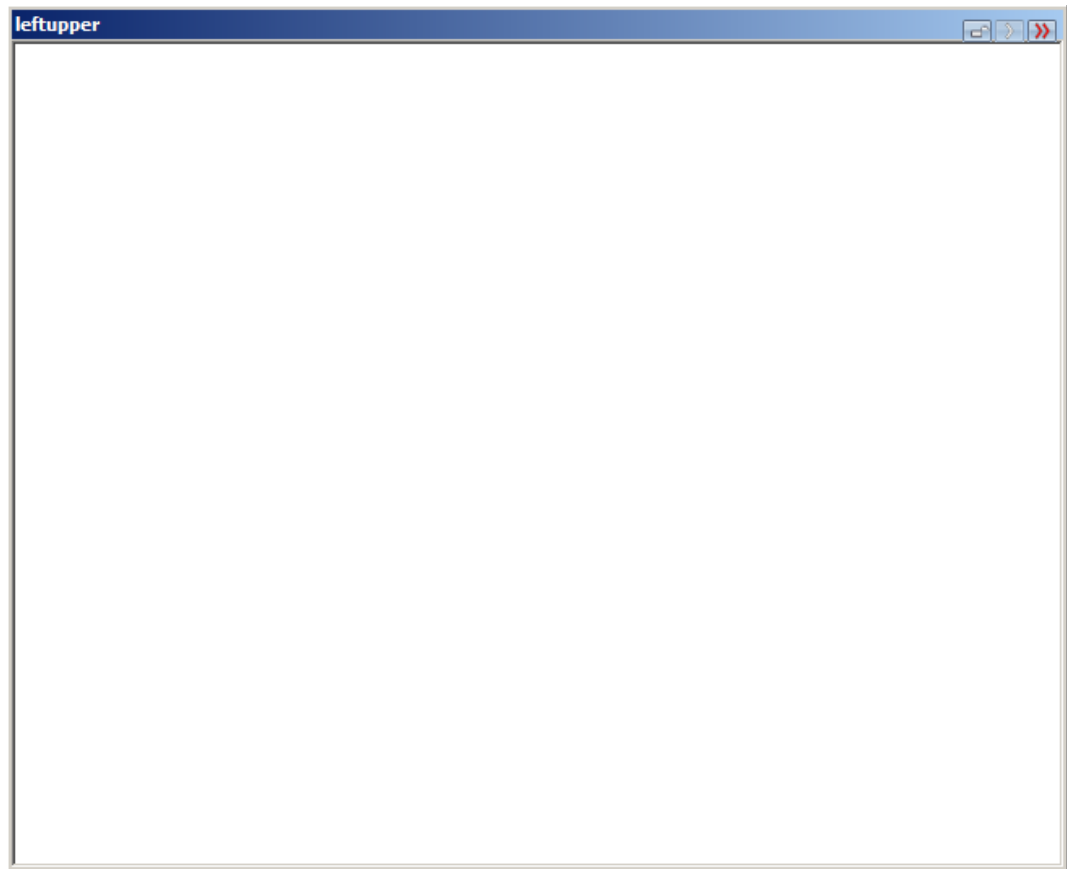
- 1 Click the output focus button on the window's title bar. The button turns green and appears pressed in, indicating that this window now has output focus.



- 2 Click the output focus button again to remove the output focus.

8.3.13 Assigning global output focus to a window

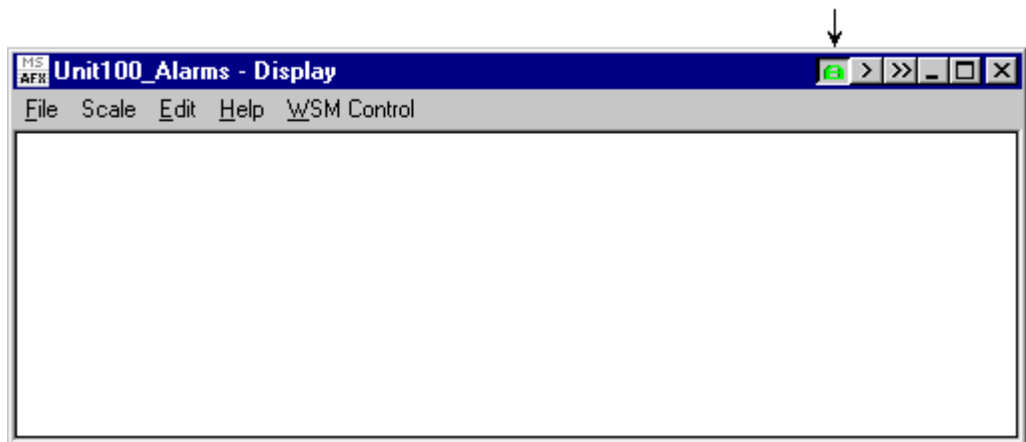
- 1 Click the global output focus button on the window's title bar. The button turns red and appears pressed in, indicating that this window now has global output focus. Note that the output focus button is unavailable.



- 2 Click the global output focus button again to remove the global output focus.

8.3.14 Locking a round robin window

- 1 Click the lock button on the window's title bar. The button turns green and appears pressed in, indicating that this round robin window is now locked.

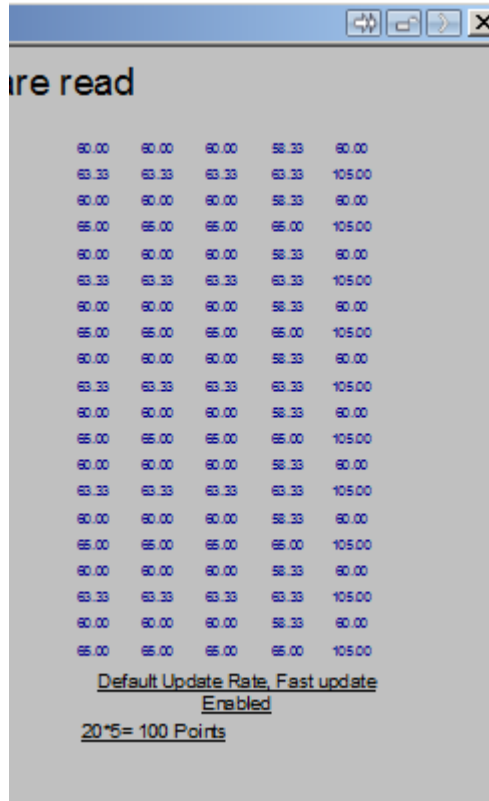


- 2 Press the lock button again to unlock the window.

8.3.15 Configuring Fast mode displays

Enable Fast mode

- 1 Click the Fast button on the window's title bar. The single arrow turns double arrow.



- 2 Click the double arrow to disable the fast mode.

Fast mode button animation

- When a display is in fast mode, the double arrow is animated. You can configure the animation rate using the **Fast Mode Button Animation** option available in the **SafeView configuration** window.
 - Default rate = 200msec per frame.
 - Range = 20 - 2000msec per frame.
 - A value of zero turns off all animation.

Fast mode timeout

- The display in a fast mode can be automatically dropped out of the fast mode, after a time-out event occurs. You can configure the time-out (in seconds) using the **Fast Mode Timeout** option available in the **SafeView Configuration** window.
 - Default time-out = 0sec (The display is not automatically dropped out of the fast mode).
 - Range = 60 - 7200sec.

Hide all SafeView fast buttons

- The SafeView Fast button is available by default in the Native Window, GUS displays and the Station displays. You can hide the Fast button using the **Hide All SafeView Fast Buttons** option available in the **SafeView Configuration** window.

Hide disabled Fast buttons

- The SafeView Fast button appears, but disabled, if the display does not have any fast mode parameters. You can hide the disabled Fast button using the **Hide Disabled Fast Buttons** option available in the **SafeView Configuration** window.

8.3.16 Using the Task Manager

Do *not* use the Task Manager to end a task associated with a placeholder or the *SAFEVIEW.exe* file. Doing so will terminate SafeView.

8.3.17 Exiting SafeView

To ensure that all active applications continue to run correctly when you terminate SafeView window management, ensure that no “modal dialogs” are currently displayed for any managed application. If you attempt to exit SafeView while this condition exists, a message will appear asking you to eliminate the modal dialogs.

An example of this is when the **About Microsoft Word** dialog box is displayed for Microsoft Word. In this scenario, clicking the **File** command on Word's menu bar is ignored, but a system “beep” alerts you to complete whatever action is required to remove the **About Microsoft Word** dialog box from the screen before executing a menu bar command.

In addition, if Lotus Notes is running, you must terminate it before exiting SafeView.

To safely exit SafeView, do the following procedure:

- 1 On the **SafeView Control Panel**, click **Exit SafeView**.

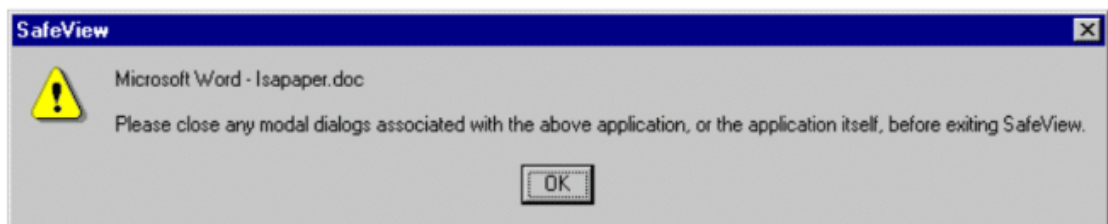
To prevent accidental shutdown, the following dialog box appears:

- 2 If you *do not* want to exit SafeView at this time, click **No**.
The dialog box disappears without affecting SafeView's management of active applications.

- 3 If you *do* want to exit SafeView, click **Yes**.

SafeView checks the applications that are currently running.

- If no modal dialogs are open, the dialog box and SafeView Control Panel disappear and SafeView's management of all active applications stops.
- If a modal dialog is open, the following dialog box appears:

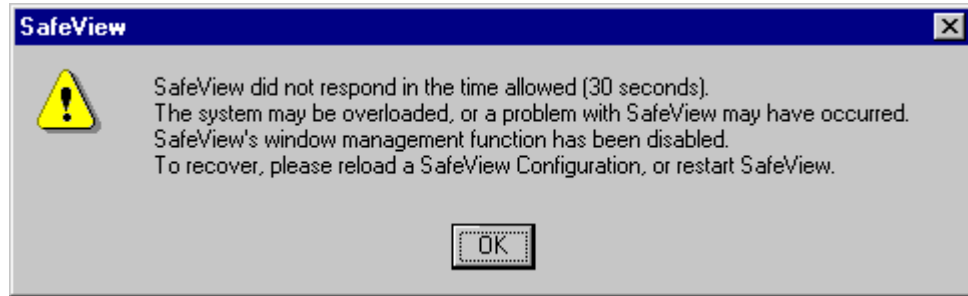


- 4 Note the specified application and instructions given in the dialog box and then click **OK**.
The dialog box disappears.
- 5 For the specified application, close any open modal dialog or the application itself, and then return to step 1.

8.3.18 System overload

If an application must wait at least 30 seconds for SafeView to reply to a management request, a severe system overload must be occurring that is preventing SafeView from executing in a timely manner. To avoid system response failure when such delays occur, SafeView is automatically disabled, allowing displays to present themselves. As a result, a SafeView dialog box appears. To correct this condition, do the following procedure:

- 1 If the following dialog box appears, note the instructions and then click **OK**.



The dialog box disappears, SafeView suspends management of the workspace, and Microsoft Windows assumes normal display management.

- 2 To recover from this condition, determine why the system is overloaded (or report a possible SafeView problem to Honeywell). Then, reload a SafeView configuration or restart SafeView.

8.3.19 Getting help for the Control Panel

The **Contents** page contains a list of all the SafeView Help system topics. It is organized like a Table of Contents in a book, enabling you to navigate through the system to find the category and subject of interest. The **Contents** page shows a “closed-book” icon to represent a category of topics and a question mark (?) to represent an individual topic

The **Index** page contains an alphabetical list of all the SafeView Help system topics. It is organized like an index in a book. It is particularly useful when you want help on a specific topic.

The **Search** page allows full-text searching for any word or phrase in the SafeView Help system. It is particularly useful when you want help on a specific topic.

- 1 On the **SafeView Control Panel**, click **Help**.
The **SafeView Documentation Help** window appears with three tabs: **Contents**, **Index**, and **Search**.
- 2 Click a tab name to open its corresponding page.

To get help using the Contents page

- 1 On the **SafeView Documentation Help** window, click the **Contents** tab.
The **Contents** page appears.
- 2 Double-click the closed-book icon that most likely contains your help topic.
The closed-book icon changes to an open-book icon, revealing a list of topics and possibly one or more closed books.
- 3 If your help topic is not listed, repeat step 2.
- 4 Click a topic to display its information in the right pane.
- 5 To print a selected item (a topic or an entire book), click the **Print** icon.
NOTE: If the selected item is a book that contains other books, all pages in all the nested books will be printed.

To get help using the Index page

- 1 On the **SafeView Documentation Help** window, click the **Index** tab.
The **Index** page appears.
- 2 Double-click the index item of interest, or type a word of interest and then double-click the resulting selection.

To get help using the Search page

- 1 On the **SafeView Documentation Help** window, click the **Search** tab.
- 2 Type a word of interest and then click **List Topics**.
- 3 Double-click a topic that was returned to view its information in the right pane.

8.3.20 Changing the SafeView search mode (Normal or Focus)

When a SafeView workspace is loaded, a search mode button (**Normal** or **Focus**) becomes available on the Control Panel as shown in the following figure.

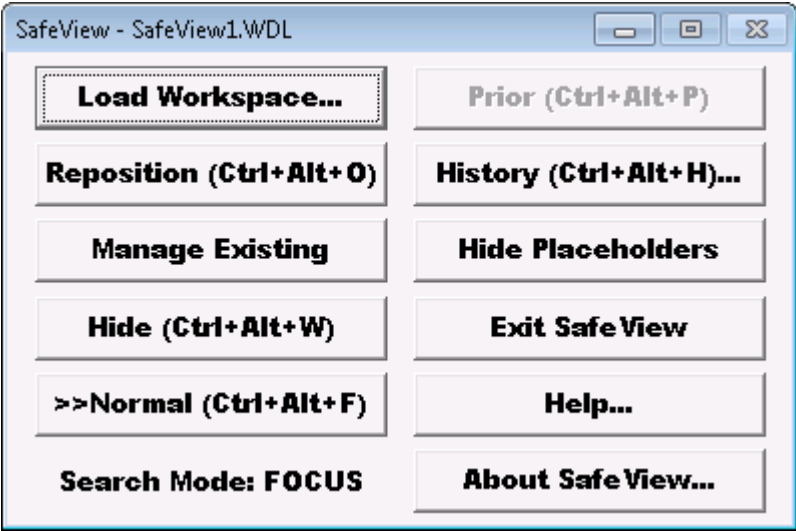


Figure 13: Example: Control Panel with an Active Search Mode Button (Focus)

The descriptive text below the search mode button indicates the current search mode. When configuring a SafeView workspace, you can set the search mode to FOCUS upon loading; otherwise, it will default to NORMAL mode. The following table describes the buttons and descriptive text. For details, see “Top-down or focus-based matching” on page 42.”

| Current Search Mode | Description |
|---------------------|---|
| NORMAL | Indicates that SafeView is in the NORMAL search mode in which it will search its workspace hierarchy from the top down. This was the only search mode available in earlier releases of SafeView. When SafeView is in the NORMAL mode, the >>Focus (Ctrl+Alt+F) button is available on the control panel to toggle SafeView to the FOCUS search mode. |
| FOCUS | Indicates that SafeView is in the FOCUS-based search mode in which it will search the group owning the currently active window first. When SafeView is in the FOCUS mode, the >>Normal (Ctrl+Alt+F) button is available on the control panel to toggle SafeView to the NORMAL search mode. |

9 Application Program Interface

Related topics

“Application Program Interface Overview” on page 114

“Guidelines for creating SafeView-aware applications” on page 116

“List of API calls” on page 118

9.1 Application Program Interface Overview

This section discusses workspace client concepts, followed by detailed descriptions of the SafeView Application Program Interface (API) calls. API calls are available for use by applications to perform a variety of functions including the following:

- Register windows for management by SafeView
- Control SafeView itself, by emulating commands to the Control Panel
- Emulate selection of the indicator buttons on the title bar of a managed window
- Retrieving data from window specifications
- Load workspace data

Application Program Interface calls can be made in the following three ways:

- Display builder scripting, Visual Basic scripting, and any other environment that can utilize an OLE in-process automation server (a registered OLE control) using the methods available,
- By custom Windows Visual C++ applications using a Visual C/C++ dll, in UNICODE or ANSI formats, and
- By generic COM-automation, for example HMIWeb displays and generic scripting applications.

9.1.1 Accessing API calls: GUS and Experion Station PKS displays

The Honeywell GUS and Experion PKS display script processors can access all of the Application Program Interface calls; thus, these calls are available from the displays, giving them complete control over workspace management, if desired.

Experion Station displays must explicitly create the SafeView automation object in order to programmatically interact with SafeView:

GUS display scripting provides, by default, a “workspace” object to access SafeView programmatically without making an explicit API call.

9.1.2 API calls related to focus-based search mode

Three SafeView API calls, “SetFocusBasedSearch,” “GetFocusBasedSearch,” and “GetWindowGroup,” are provided to change the SafeView search mode through automation or directly through C++.



Attention

- For automation based calls, the existing *wsmcliauto.dll* automation server (prog id: “Honeywell.Workspace.Client”) supports the above calls.

For C++ based calls, the parameters and return values are the same, but each of the routine names begins with “WS” as in *WSSetFocusBasedSearch*.

VBScript sample

The following script illustrates the use of those three SafeView API calls:

```
Dim sv As Object
Set sv = CreateObject("Honeywell.workspace.Client")
If Not sv.ismanageractive Then
MsgBox "SafeView NOT running!"
Else
'INDICATE CURRENT STATUS:
If Not sv.GetFocusBasedSearch Then
MsgBox "SafeView Focus-Based search is OFF!"
Else
MsgBox "SafeView Focus-Based search is ON!"
End If
End If
' TOGGLE THE FOCUS MODE:
If sv.GetFocusBasedSearch Then
sv.SetFocusBasedSearch (False)
Else
```

```

sv.SetFocusBasedSearch (True)
End If
' GET THE GROUP STRING FOR WINDOW HANDLE (this example requires window handle be typed in)
MsgBox sv.GetWindowGroup(InputBox("Enter window handle"))
Set sv = Nothing
End Sub

```

9.1.3 Script examples

This section contains GUS Display and Visual C++ script examples. For Experion Station display script examples, refer to the *Experion HMIWeb Display Building Guide*.

9.1.4 SafeView-aware applications

A SafeView-aware application is any application that uses one or more SafeView API calls to register its display window(s) and uses other features described in this section.

SafeView-aware applications specifically use the SafeView API calls to leverage the matching function associated with window categories. The primary responsibility of a SafeView-aware application is to supply SafeView with a window category and re-invocation information for each of the application's displays (top-level windows). The window category concept provides a powerful means for custom-organizing displays to be managed by SafeView.

Any application display can be SafeView-aware by using the “CreateWorkspaceClient” API to launch an application and provide the additional information required for any third-party application considered to be SafeView-aware. For more information, refer to the ““SafeView-aware” on page 28” definition.

9.1.5 Workspace configuration

Refer to the “Window specification” on page 35 .

9.2 Guidelines for creating SafeView-aware applications

9.2.1 Register window calls

A workspace client is simply a SafeView-aware Microsoft Windows program that makes one of these three API calls:

- The “RegisterWindowCategory” API call associates a display window with a specific window category string. If you use this call, however, the display will not be available as a historized display on the SafeView Display History dialog box and it cannot be invoked as a prior display. Avoid this call unless re-invocation as a prior display is explicitly not desired.
- The “RegisterWindowData” API call associates a display window with a window category string and also provides sufficient information to allow the display to be re-invoked. This information includes the command line (with parameters, if needed), working directory, and a prompt string for inclusion in the SafeView Display History list.
- The “RegisterWindowDataSpec” API call extends the RegisterWindowData call by also providing the specific name of a workspace configuration window specification that is to be used to manage this window, thus overriding the normal category-based disposition of this window. If the given window specification is not found in the current workspace, the display is managed according to the given category.

The names of the API calls are slightly different for those choosing to directly access the API through a Visual C++ application that includes the *wsmcli.h* header file. See ““Visual C++ sample” on page 240” for an example of the workspace client's *mainfrm.cpp* module.



Tip

For Visual C++, link *wsmcli.dll* into your project and include *wsmcli.h* header in your C++ file.

9.2.2 Determine the window category

To register a window for management, the workspace client application must supply a window category string to SafeView. SafeView-aware displays provide a means to associate this category string with the display at display build time. Otherwise, the identification of this category string value is the workspace client application's responsibility. Methods to supply the window category include

- Include the category as an argument to the program.
- Provide the category for each top level window owned by an application.
- Select the most appropriate category string dynamically, from a list of available categories, based upon run-time information such as user ID, display characteristics, and plant status.
- Use the CreateWorkspaceClient API.

You can also provide category values as a property of the displays that you develop.

9.2.3 When to register the window

SafeView-aware displays automatically register themselves when they are invoked. For custom applications, the Windows operating system informs an application, through a WM_CREATE message, whenever a window has been created, but before the window is actually displayed. This gives the application the opportunity to do some processing (in this case, registering the window with SafeView) before the window is displayed.

In Microsoft Visual C++ MFC-based applications, windows usually derive from MFC class CWnd. This class's OnCreate method is called at window creation time and this is a good place to register the window with SafeView.

9.2.4 Multiple top-level windows

SafeView-aware displays maintain a one-to-one relationship between a display-run time and a single display. For custom applications, however, it is acceptable for a given to maintain multiple top-level windows, each of which is individually registered, perhaps with different window categories. In this case, it might not be appropriate to re-invoke (using Display History or Prior Display) the entire application. Rather, the configured re-invoke information can be processed by such a multi-display custom application, to determine if the operator wishes to redisplay only a particular component-display. With regard to Display History, you can use the “RegisterWindowCategory” API call to exclude specific displays from the list.

9.2.5 Scale a display window

Depending on the given workspace configuration, display windows can be positioned and sized. It can be helpful, for applications that provide graphical renderings, to provide an option (such as /scale) that automatically scales the display to the current window size. Some displays provide a scaling option (“Zoom to Fit”).

9.2.6 Boolean return values

All workplace client API calls return *true* if the function succeeds and *false* if the function fails, unless the API is specifically requesting a *true/false* status (such as “IsManagerActive”), in which case *false* means “NO.” If more information is desired, the “GetLastWorkspaceError” API can be called, which returns an error code. For details, see “ “GetLastWorkspaceError” on page 122.”

9.2.7 HWND versus long value types

Each window within Microsoft Windows has a unique “handle,” which is a number that identifies a window to the operating system. This window handle is generally a value with a predefined type, called HWND, which is a “long” number.

In Visual C++ programs and other applications that interact directly with the *WsmCli.dll*, window handles are defined by using this HWND predefined type.

The type “long” is used to identify window handles in **Honeywell GUS** display scripts, **Experion Station PKS** display scripts, Visual Basic programs, and other applications that interact with SafeView using the SafeView Client API calls exposed using the WsmCliAuto OLE in-process server (instead of HWND).

9.2.8 Case sensitivity

By default, SafeView is *not* case-sensitive with respect to the values of any of the three MATCH expression properties (CATEGORY, MODULE, or TITLE) associated with registered windows. However, SafeView *is* case-sensitive with respect to API calls that include certain window specification names. The command line argument /cs makes SafeView itself case-sensitive with respect to the MATCH expression strings. For example, with the /cs option specified, a MATCH expression with MATCH = title(“*.Exe”) will only match windows with title extension “Exe.” You can use the Task Manager to get the exact configuration. Without the /cs option, MATCH = title(“*.Exe”) will match windows with title extensions “EXE,” “exe,” in addition to “Exe.”

9.3 List of API calls

9.3.1 CloseWindow

Description

“CloseWindow” closes the window with the handle, “hWnd.”

Visual C++ code:

```
BOOL WSCloseWindow(Long hwnd)
```

Visual Basic code:

```
Boolean CloseWindow(long window);
```

Parameters

hWnd

A window handle.

Return value

Returns *true* if the window with the given handle was successfully sent a WM_CLOSE message.

Remarks

The Visual C/C++ version is “WSCloseWindow(HWND hWnd).”

```
WSCloseWindow(pApp->m_pMainwnd->m_hwnd);
```

Example (Experion display script)

```
dim sv
dim bRtn
'set mywindowHandle as desired
set sv = CreateObject("honeywell.workspace.client")
mywindowHandle= window.parent.external.application.appwindow.hwnd
bRtn = sv.CloseWindow(mywindowHandle)
set sv = nothing
```

9.3.2 Configuration Loaded

Description

“ConfigurationLoaded” informs the caller whether or not SafeView has a workspace configuration file loaded.

Visual C++ code:

```
BOOL WSConfigurationLoaded()
```

Visual Basic code:

```
Boolean ConfigurationLoaded()
```

Parameters

None.

Return value

Returns *true* if SafeView has a workspace configuration loaded.

Remarks

This API call allows applications to determine if SafeView has a workspace currently loaded. The Visual C/C++ version is “WSConfigurationLoaded.”

See also

“IsManagerActive” on page 126, “StartManager” on page 145

Example (Experion display script)

```
dim sv
dim bRtn
set sv = CreateObject("honeywell.workspace.client")
bRtn = sv.ConfigurationLoaded()
```

9.3.3 CreateWorkspaceClient

Description

“CreateWorkspaceClient” launches a process and registers a window category, re-invocation data, and window specification name, on its behalf. This API call is used to launch a third party application (such as Microsoft Word) such that it becomes Safeview-aware, thereby enabling management by category and re-invocation from the SafeView history list and the **PRIOR DISP** button.

Visual C++ code:

```
int CreateWorkspaceClient(
LPCTSTR pszWindowSpec
LPCTSTR pszCategory,
LPCTSTR pszPromptText,
LPCTSTR pszCommandLine,
LPCTSTR pszWorkingDir,
);
```

Visual Basic code:

```
boolean CreateWorkspaceClient(BSTR WindowSpec, BSTR Category, BSTR PromptText, BSTR CommandLine,
BSTR WorkingDirectory);
```

Parameters Visual C++(VB)

pszWindowSpec(WindowSpec)

A pointer to a null-terminated string containing a window specification name. This string is used by SafeView to override the MATCH expression and to select a window specification (for use in managing this display) directly. If the given window specification name is not found in the loaded workspace configuration, the normal matching approach is applied in an effort to disposition the display for management.

Note: this parameter is generally best left set to “” (an empty string), so that the normal category matching approach to window management is utilized.

Because directly selecting a window overrides the way standard SafeView matching logic would perform this function, the output focus and global output focus buttons are not updated. For example, if the output focus button is lit on Window1 and the “CreateWorkspaceClient” API call selects Window3, the application will go to Window3, but the output focus button on Window1 will remain lit.

pszCategory(Category)

A pointer to a null-terminated string containing a window category. This string is used by SafeView to match an appropriate workspace window specification to this window.

pszPromptText(PromptText)

A pointer to a null-terminated string containing the text included in the Display History dialog for this application display. This string represents this display in the Display History dialog's list.

pszCommandLine(CommandLine)

A pointer to a null-terminated string that specifies the command line to execute. If the filename does not contain an extension, an .EXE pathname is assumed. If the filename ends in a period (.) with no extension, or if the

filename contains a path, the *.EXE* extension is not appended. If the filename does not contain a directory path, the system searches for the executable in the following order:

1. The current directory for the SafeView process.
2. The 32-bit Windows system directory. Use the “GetSystemDirectory” function to get the path to this directory. The name of this directory is “SYSTEM32.”
3. The Windows directory. Use the “GetWindows Directory” function to get the path to this directory.
4. The directories listed in the PATH environment variable.

pszWorkingDir(WorkingDirectory)

A pointer to a null-terminated string that specifies the current drive and directory for the child process. The string must consist of a full path and filename that includes a drive letter. If this parameter is NULL, the new process is created using the same current drive and directory as the calling process. This option is provided primarily for shells that need to start an application and specify its initial drive and working directory. Note that this parameter will *not* enable the system to find executable files that fail the searches listed in the *pszCommandLine* parameter information previously described.

Return value

Returns *true* if the function succeeds.

Remarks

This API call is made automatically by SafeView-aware displays when they are invoked, using the “RegisterWindowData” API call. The Visual C/C++ version is “WSCreateWorkspaceClient.”

Example (Experion display script)

```
Sub OnLButtonClick()
Dim bResult
Dim sv
set sv = CreateObject("Honeywell.Workspace.Client")
bResult = sv.createworkspaceclient(
"", // window spec name, if it were going to be used
"REPORTS",
"Lab Report: ShiftNotes",
"c:\\Program Files\\Microsoft Office\\Office\\winword.exe", "d:\\shiftrep\\shiftnotes.doc",
"d:\\shiftrep")
If bResult = True then
//do something
end if
set sv = nothing
End Sub
Sub OnLButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.Workspace.Client")
bResult = sv.createworkspaceclient(
"", // window spec name, if it were going to be used
"MISCAPPS",
"c:\\windows\\system32 notepad.exe")
"c:\\windows\\system32")
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.4 DisplayHistoryDialog

Description

“DisplayHistoryDialog” allows an application to ask SafeView, if executing, to display the Display History dialog. This API call emulates clicking the **Display History** button on the workspace Control Panel.

Visual C++ code:

```
BOOL WSDisplayHistoryDialog();
```


Visual Basic code:

```
boolean DisplayHistoryDialog();
```

Parameters

None.

Return value

Returns *true* if the function succeeds.

Remarks

Operators can view the last fifty historized displays and invoke any one of them through the Display History dialog or by pressing **Ctrl+Alt+H**. The Visual C/C++ version is “WSDisplayHistoryDialog.”

See also

“InvokePriorDisplay” on page 125, “IsManagerActive” on page 126

Example (Experion display script)

```
Sub OnLButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.displayhistorydialog
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.5 GetCurrentForwardDisplayIndex

Description

“GetCurrentForwardDisplayIndex” is useful in supporting custom “display forward/backward” navigation.

Return value

This API call returns the prompt string of the display just above the display that is currently highlighted in the display history list. That is, it returns the index of the next display to be invoked if the “InvokeForwardDisplay” API were to be called.

Remarks

DISP_FUNCTION(CWorkspaceClient, “GetCurrentForwardDisplayIndex”, GetCurrentForwardDisplayIndex, VT_I4, VTS_NONE)

9.3.6 GetCurrentPriorDisplayIndex

Description

“GetCurrentPriorDisplayIndex” is useful in supporting custom “display forward/backward” navigation.

Return value

This API call returns the prompt string of the display just below the display that is currently highlighted in the display history list. That is, it returns the index of the next display to be invoked if prior-display were selected.

Remarks

DISP_FUNCTION(CWorkspaceClient, “GetCurrentPriorDisplayIndex”, GetCurrentPriorDisplayIndex, VT_I4, VTS_NONE)

9.3.7 Get FocusBasedSearch

Description

“GetFocusBasedSearch” returns the current global search mode as *focus-based* or *normal*.

Visual C++ code:

```
BOOL GetFocusBasedSearch();
```

Parameters

None.

Return value

For the current search mode, returns *true* if focus-based or *false* if normal.

Remarks

The Visual C/C++ version is “WSSetFocusBasedSearch.”

Example (Experion display script)

```
dim sv
dim bRtn
set sv = CreateObject("honeywell.workspace.client")
bRtn = sv.GetFocusBasedSearch()
set sv = nothing
```

9.3.8 GetLastWorkspaceError

Description

“GetLastWorkspaceError” returns the most recent API error status from SafeView.

Visual C++ code:

```
long GetLastWorkspaceError()
```

Visual Basic code:

```
long GetLastWorkspaceError();
```

Parameters

None.

Return value

The return value is an error code as defined in the following table:

| Error Code | Internal Constant | Meaning |
|------------|-------------------------------|--|
| 0 | WSM_API_STATUS_OK | No workspace errors have occurred. |
| 1 | WSM_API_TIMEOUT | SafeView is not responding. Timed out. |
| 2 | WSM_API_NOT_EXECUTING | SafeView is not executing. |
| 3 | WSM_API_INVALID_BUFFER_PASSED | Invalid buffer size. |
| 4 | WSM_API_WINDOW_NOT_MANAGED | Specified window is not being managed. |
| 5 | WSM_API_STATUS_UNDEFINED | API return status not defined. |
| 6 | WSM_API_CREATE_PROCESS_FAILED | The operating system failed to create the process. |

| Error Code | Internal Constant | Meaning |
|------------|-----------------------------|---|
| 7 | WSM_API_WINDOW_DISABLED | The given window is disabled. Close message was not sent. |
| 8 | WSM_API_NOT_YET_DEFINED | API not defined. |
| 9 | WSM_API_INVALID_PARAMETER | Invalid parameter(s) provided to API call. |
| 10 | WSM_API_ALREADY_EXECUTING | SafeView already executing. |
| 11 | WSM_API_INSUFFICIENT_MEMORY | Insufficient memory to complete API call. |
| 12 | WSM_API_DATA_REJECTED | Supplied data rejected by SafeView. |

Remarks

The Visual C/C++ version is “WSGetLastWorkspaceError.”

See also

IsManagerActive

Example (Experion display script)

```

Sub OnButtonClick()
Dim lastError as long
Dim sv
set sv = CreateObject("Honeywell.workspace.client")
lastError = sv.getlastworkspaceerror
// e.g. msgbox per lastError case
Set sv = nothing
End Sub

```

9.3.9 GetPriorDisplayAtIndex

Description

“GetPriorDisplayAtIndex” is useful in supporting custom “display forward/backward” navigation.

Return value

Given a long index number, it returns the string value of the prompt string in the display history list at that index.

Remarks

DISP_FUNCTION(CWorkspaceClient, “GetPriorDisplayAtIndex”, GetPriorDisplayAtIndex, VT_BSTR, VTS_I4)

9.3.10 GetWindowGroup

Description

“GetWindowGroup” takes a decimal value window handle and, if the window handle is a valid handle of a SafeView-managed display, it returns a string containing all of the SafeView groups that contain that window.

All windows are contained by at least one (the main workspace) group, though windows may be nested in multiple groups. For example, if a window is in FPGroup1 and that group is in “BlueGroup” and that group is in the workspace “MainGroup,” then the string returned for this window would be: *MainGroup\BlueGroup\FPGroup1*.

DISP_FUNCTION (CWorkspaceClient, “GetWindowGroup”, GetWindowGroup, VT_BSTR, VTS_I4).

Remarks

The C++ version is identical except the routine is *LPWSTR WSGetWindowGroup(long)*.

9.3.11 GetWindowHandle

Description

“GetWindowHandle” returns the window handle of the first-found application display currently being managed by the given window specification. It returns 0 (zero) if the given window specification name is not found or if no application is currently being managed by using the given window specification.

Visual C++ code:

```
long GetWindowHandle(
LPCTSTR pszWindowSpec,
);
```

Visual Basic code:

```
Long GetWindowHandle(BSTR WindowSpec);
```

Parameters

pszWindowSpec

pointer to a string containing the name of a window specification.

Return value

Returns a window handle as a long value if the function succeeds. Returns 0 (zero) if the function fails.

Remarks

If the given window specification has MAXWINDOWS = *infinite*, the first-managed window found will be returned. The Visual C/C++ version is “WSGetWindowHandle((LPCWSTR) WindowSpec).”

Example (Experion display script)

```
Sub OnLButtonClick()
dim x As Long
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
x = sv.GetWindowHandle("window1")
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.12 GetWindowSpecName

Description

“GetWindowSpecName” returns the string which is the SafeView window specification associated with the given window handle.

Return value

Takes a long and returns a string.

Remarks

DISP_FUNCTION (CWorkspaceClient, “GetWindowSpecName”, GetWindowSpecName, VT_BSTR, VTS_I4)

9.3.13 HidePlaceholders

Description

“HidePlaceholders” allows an application to ask SafeView to hide and not to present placeholders, even though window specifications configured with PLACEHOLDERS = *true* can exist, with which no applications are

currently being managed. This call emulates clicking the **Hide Placeholders** button on the workspace Control Panel.

Visual C++ code:

```
BOOL HidePlaceholders();
```

Visual Basic code:

```
boolean HidePlaceholders();
```

Parameters

None.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is “WSHidePlaceholders.”

See also

“ShowPlaceholders” on page 145***

Example (Experion display script)

```
Sub OnLButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.hideplaceholders
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.14 InvokeForwardDisplay

Description

“InvokeForwardDisplay” is useful in supporting custom “display forward/backward” navigation. It invokes the display above the currently highlighted display in the display history list. For example, if the third item in the display history list is highlighted, then this API call would invoke the second item.

Remarks

DISP_FUNCTION(CWorkspaceClient, “InvokeForwardDisplay”, InvokeForwardDisplay, VT_BOOL, VTS_NONE)

9.3.15 InvokePriorDisplay

Description

“InvokePriorDisplay” allows an application to ask SafeView to invoke the prior *invoked* display. This API call emulates clicking the **Prior Display** button on the workspace Control Panel or the **Prior** button on the IKB/OEP.

Visual C++ code:

```
BOOL InvokePriorDisplay();
```

Visual Basic code:

```
boolean InvokePriorDisplay();
```

Parameters

None.

Return value

Returns *true* if the function succeeds.

Remarks

None.

Example (Experion display script)

```
Sub OnButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.invokepriordisplay
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.16 InvokePriorDisplayAtIndex*Description*

“InvokePriorDisplayAtIndex” is useful in supporting custom “display forward/backward” navigation. It invokes the display at the given position in the display history list. So, for example, passing “3” as the (long) numeric parameter would invoke the third item as shown in the display history list.

Remarks

DISP_FUNCTION(CWorkspaceClient, “InvokePriorDisplayAtIndex”, InvokePriorDisplayAtIndex, VT_BOOL, VTS_I4)

9.3.17 IsManagerActive*Description*

“IsManagerActive” reports whether or not SafeView process is currently executing.

Visual C++ code:

```
BOOL IsManagerActive()
```

Parameters

None.

Return value

Returns *true* if SafeView is executing.

Remarks

This call allows applications to determine if SafeView is running. The Visual C/C++ version is “WSIsManagerActive.”

See also

“StartManager” on page 145

Example (Experion display script)

```
Sub OnButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.ismanageractive
```

```

If bResult = True then
//do something
end if
set sv = nothing
End Sub

```

9.3.18 IsWindowFocus

Description

“IsWindowFocus” determines if the given window is managed and has output focus.

Visual C++ code:

```

BOOL IsWindowFocus(
Long hwnd
);

```

Visual Basic code:

```

boolean IsWindowFocus(long window);

```

Parameters

hwnd

A window handle.

Return value

Returns *true* if the window with the given handle is being managed and has output focus.

Remarks

The Visual C/C++ version is “WSGetIsWindowFocus(HWND hwnd).”

See also

“IsWindowGlobalFocus” on page 127

Example (Experion display script)

```

Sub OnLButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
hwnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.IsWindowFocus(hwnd)
if bReturn = True then
// do something
End If
set sv = nothing
End Sub

```

9.3.19 IsWindowGlobalFocus

Description

“IsWindowGlobalFocus” determines if the given window is being managed and has global output focus.

Visual C++ code:

```

BOOL IsWindowGlobalFocus(
Long hwnd
);

```

Visual Basic code:

```

boolean IsWindowGlobalFocus(long window);

```

Parameters

hWnd

A window handle.

Return value

Returns *true* if the window with the given handle is being managed and has global output focus.

Remarks

The Visual C/C++ version is “WSGetIsWindowGlobalFocus(HWND hWnd).”

See also

“IsWindowFocus” on page 127

Example (Experion display script)

```
Sub OnButtonClick()
dim sv
dim hWnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
hWnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.IsWindowGlobalFocus(hWnd)
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.20 IsWindowLocked

Description

“IsWindowLocked” returns *true* if the given window is being managed and currently locked.

Visual C++ code:

```
BOOL IsWindowLocked(
Long hWnd
);
```

Visual Basic code:

```
boolean IsWindowLocked(long window);
```

Parameters

hWnd

A window handle.

Return value

Returns *true* if the window with the given handle is managed and is currently locked.

Remarks

The Visual C/C++ version is “WSGetIsWindowLocked(HWND hWnd).”

Example (Experion display script)

```
Sub OnButtonClick()
dim sv
dim hWnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
hWnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.IsWindowLocked(hWnd)
if bReturn = True then
// do something
End If
```



```
set sv = nothing
End Sub
```

9.3.21 IsWindowManaged

Description

“IsWindowManaged” returns *true* if an application display with the given window handle is currently being managed.

Visual C++ code:

```
BOOL IswindowManaged(
Long hwnd
);
```

Visual Basic code:

```
boolean IswindowManaged(long window);
```

Parameters

hWnd

A window handle.

Return value

Returns *true* if the window with the given handle is being managed.

Remarks

The Visual C/C++ version is “WSGetIsWindowManaged(HWND hWnd).”

Example (Experion display script)

```
Sub OnButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
hwnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.IsWindowManaged(hwnd)
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.22 IsWindowRegistered

Description

“IsWindowRegistered” returns *true* if the given window has been registered (including if it was “auto-registered” as a third-party application).

Visual C++ code:

```
BOOL IswindowRegistered(
Long hwnd
);
```

Visual Basic code:

```
boolean IswindowRegistered(long window);
```

Parameters

hWnd

A window handle.

Return value

Returns *true* if the window handle has been registered.

Remarks

The Visual C/C++ version is “WSGetIsWindowRegistered(HWND hWnd).”

Example (Experion display script)

```
Sub OnLButtonClick()
dim sv
dim hwnd
dim bReturn as Boolean
set sv = CreateObject("Honeywell.workspace.Client")
hwnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.IsWindowManaged(hwnd)
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.23 IsWindowSpecFocus*Description*

“IsWindowSpecFocus” returns *true* if the given window specification name is found and currently has output focus.

Visual C++ code:

```
BOOL IsWindowSpecFocus(
LPCTSTR pszWindowSpec,
);
```

Visual Basic code:

```
boolean IsWindowSpecFocus(BSTR WindowSpec);
```

*Parameters**pszWindowSpec*

A pointer to a string containing the name of a window specification.

Return value

Returns *true* if the window specification is found, if it is being used to manage an application display, and if the display has output focus.

Remarks

The Visual C/C++ version is “WSGetIsWindowSpecFocus(LPCTSTR WindowSpec).”

See also

Note that *output focus* and *global output focus* are different. See “IsWindowSpecGlobalFocus” on page 131.

Example (Experion display script)

```
Sub OnLButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
bReturn = sv.IsWindowSpecFocus("window1")
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.24 IsWindowSpecGlobalFocus

Description

“IsWindowSpecGlobalFocus” returns *true* if the given window specification name is found and currently has global output focus.

Visual C++ code:

```
BOOL IsWindowSpecGlobalFocus(
LPCTSTR pszWindowSpec,
);
```

Visual Basic code:

```
boolean IsWindowSpecGlobalFocus(BSTR windowSpec);
```

Parameters

pszWindowSpec

A pointer to a string containing the name of a window specification.

Return value

Returns *true* if the window specification is found, and if it is being used to manage an application display, and if the display has global output focus.

Remarks

The Visual C/C++ version is “WSGetIsWindowSpecGlobalFocus(LPCTSTR WindowSpec).”

See also

Note that *output focus* and *global output focus* are different. See “IsWindowFocus” on page 127.

Example (Experion display script)

```
Sub OnLButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
bReturn = sv.IsWindowSpecGlobalFocus("window1")
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.25 IsWindowSpecLocked

Description

“IsWindowSpecLocked” returns *true* if the given window specification name is found and is currently locked.

Visual C++ code:

```
BOOL IsWindowSpecLocked(
LPCTSTR pszWindowSpec,
);
```

Visual Basic code:

```
boolean IsWindowSpecLocked(BSTR windowSpec);
```

Parameters

pszWindowSpec

A pointer to a string containing the name of a window specification.

Return value

Returns *true* if the window specification is found and is currently locked.

Remarks

The Visual C/C++ version is “WSGetIsWindowSpecLocked(LPCTSTR WindowSpec).”

Example (Experion display script)

```
Sub OnLButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
bReturn = sv.IsWindowSpecLocked("window1")
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.26 IsWindowSpecManaged

Description

“IsWindowSpecManaged” returns *true* if the given window specification name is found and is currently being used to manage an application display.

Visual C++ code:

```
BOOL IsWindowSpecManaged(
LPCTSTR pszWindowSpec,
);
```

Visual Basic code:

```
boolean IsWindowSpecManaged(BSTR windowSpec);
```

Parameters

pszWindowSpec

A pointer to a string containing the name of a window specification.

Return value

Returns *true* if the window specification is found and is being used for managing an actual display.

Remarks

If the given window specification has MAXWINDOWS= *infinite*, the first-managed window found will be returned. The Visual C/C++ version is “WSGetIsWindowSpecManaged(LPCTSTR WindowSpec).”

Example (Experion display script)

```
Sub OnLButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
bReturn = sv.IsWindowSpecManaged("window1")
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.27 LoadWorkspaceDialog

Description

“LoadWorkspaceDialog” allows an application to ask SafeView to present the File Open dialog to the operator, allowing the operator to navigate to a workspace configuration file to load. This API call emulates clicking the **Load Workspace** button on the SafeView Control Panel.

Visual C++ code:

```
BOOL LoadWorkspaceDialog();
```

Visual Basic code:

```
boolean LoadWorkspaceDialog();
```

Parameters

None.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is “WSLoadWorkspace.”

Example (Experion display script)

```
Sub OnLButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.loadworkspacedialog
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.28 LoadWorkspaceFile

Description

“LoadWorkspaceFile” allows an application to ask SafeView to load a particular workspace configuration file. This allows a display to respond to an event by actually loading a new workspace. It would also be appropriate to use the “ManageExisting” API call so that all existing applications would become managed. (This would be harmlessly redundant if SafeView was launched with the */mp* option, which would cause it to manage pre-existing applications on every workspace configuration load.)

Visual C++ code:

```
BOOL LoadWorkspaceFile(
LPCTSTR pszWorkspacePath
);
```

Visual Basic code:

```
boolean LoadWorkspaceFile(BSTR workspaceFile);
```

Parameters

pszWorkspacePath The path or filename of the workspace configuration file to be loaded.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is “WSLoadWorkspaceFile.”

Example (Experion display script)

```
Sub OnLButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.loadworkspacefile("d:\\wksp\\op_wksp1.wd1")
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.29 ManageExisting

Description

“ManageExisting” allows an application to ask SafeView to attempt to manage applications that existed before the current workspace configuration was loaded. This API call emulates clicking the **Manage Existing** button on the SafeView Control Panel.

Visual C++ code:

```
BOOL ManageExisting();
```

Visual Basic code:

```
boolean ManageExisting();
```

Parameters

None.

Return value

Returns *true* if the function succeeds.

Remarks

If a manual select group is defined and if there is more than one previously loaded application display scheduled to be managed by this group, when you make a “ManageExisting” API call, each application display will appear momentarily and then be replaced, except for the last one, which will remain on the screen.

See also

“StartManager” on page 145

Example (Experion display script)

```
Sub OnLButtonClick()
Dim bResult As Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.manageexisting
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.30 OriginalPositions

Description

“OriginalPositions” allows an application to ask SafeView to reposition all managed application displays to their originally displayed positions. This call emulates clicking the **Reposition** button on the SafeView Control

Panel. Note that if a display is moved and then replaced, the new display's "original position" is the old display's moved position.

Visual C++ code:

```
BOOL OriginalPositions();
```

Visual Basic code:

```
boolean OriginalPositions();
```

Parameters

None.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is "WSOrigPositions."

Example (Experion display script)

```
Sub OnButtonClick()
Dim bResult AS Boolean
Dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.originalpositions
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.31 RegisterWindowCategory

Description

"RegisterWindowCategory#8221; registers a window and its window category with SafeView. This is one of the API calls that make a non-Honeywell application SafeView-aware, thus enabling the benefits afforded to SafeView-aware displays. This API call is not necessary for Honeywell applications that are *already* SafeView-aware.

Visual C++ code:

```
BOOL RegisterWindowCategory(
Long hwnd, // window handle for window to be registered
LPCTSTR pszCategory // points to a string containing the category);
```

Visual Basic code:

```
boolean RegisterWindowCategory(long window, BSTR Category);
```

Parameters Visual C++(VB)

hwnd(Window)

The window handle for the window to be registered.

pszCategory(Category)

A pointer to a null-terminated string containing a window category. This string is used by SafeView to match an appropriate workspace window specification to this window.

Return value

Returns *true* if the function succeeds.

Remarks

This API call should be used only when this display is not intended to be available on the Display History list or as a Prior Display; in these cases, “RegisterWindowData” should be used. This call is not utilized by SafeView-aware displays. Such custom window registration calls should be made when the window procedure for this window receives the WM_CREATE message. That is, after the window handle is known, but before the window is actually shown. The Visual C/C++ version is “WSRegisterWindowCategory.”

See also

“RegisterWindowData” on page 136

Example (Visual C++)

```
::WSRegisterWindowCategory((LPCTSTR) sCategory, m_hwnd);
```

9.3.32 RegisterWindowData

Description

“RegisterWindowData” registers a window with a window category and the necessary information required to re-invoke the application display. This is the suggested call for use by workspace clients, because it allows displays to be historized and re-invoked as prior displays. This API call is not necessary for Honeywell process displays, as these displays are already SafeView-aware.

Visual C++ code:

```
BOOL RegisterWindowData(
Long hwnd, // window handle for window to be registered LPCTSTR pszCategory, // points to a string
containing the category
LPCTSTR pszPromptText, // string containing user-prompt to be include in Display History dialog
list
LPCTSTR pszCommandLine, // string containing command line and arguments to re-invoke the display
LPCTSTR pszWorkingDir, // string containing the working directory of the application for re-
invokation purposes);
```

Visual Basic code:

```
boolean RegisterWindowData(long window, BSTR Category, BSTR Prompt, BSTR CommandLine, BSTR
workingDirectory);
```

Parameters-Visual C++ (VB)

hWnd (Window)

The window handle for the window to be registered.

pszCategory (Category)

A pointer to a null-terminated string containing a window category. This string is used by SafeView to match an appropriate workspace window specification to this window.

pszPromptText (Prompt)

A pointer to a null-terminated string containing the text included in the Display History dialog for this application display. This string represents this display in the Display History dialog's list.

pszCommandLine (CommandLine)

A pointer to a null-terminated string containing the command line to re-invoke this display. This string is used by SafeView when the operator re-invokes this display from a prior display or from the Display History dialog.

pszWorkingDir (WorkingDirectory)

A pointer to a null-terminated string containing the working directory for this application. This string is included as the working directory parameter when this display is re-invoked from a prior display or from the Display History dialog.

Return value

Returns *true* if the function succeeds.

Remarks

This API call is made automatically by SafeView-aware displays when they are invoked. The Visual C/C++ version is “WSRegisterWindowData.”

See also

“RegisterWindowCategory” on page 135, “RegisterWindowDataSpec” on page 137

Example (Visual C++ OnCreate override)

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    HSI_RESULT hResult;
    BOOL bCategorySpecified = FALSE;
    BOOL bWindowSpecSpecified = FALSE;
    CString sCategory;
    CString sPromptText = _T("");
    CString sCommandLine;
    CString sWorkingDir =
        _T("v:\\display\\workspaces\\test\\pclient\\bitmaps");
    CCmdLineOptions cmdLineOptions;
    // Get command line
    sCommandLine = ::GetCommandLine();
    // Get category if specified, from command line
    if (cmdLineOptions.GetKeywordOptionValue(_T("w"), sCategory))
        bCategorySpecified = TRUE;
    if (cmdLineOptions.GetKeywordOptionValue(_T("x"), sWindowSpec))
        bWindowSpecSpecified = TRUE;
    // Get document if specified and cut off the path
    // name, leaving just the file name
    if (!cmdLineOptions.GetOptionValue(sPromptText))
        // get document into sPromptText
        sPromptText = _T("PClient - Untitled");
    Else
    {
        CString sTemp = sPromptText;
        // find last '\\' then set sPromptText to only
        // chars after the '\\'
        int i = sTemp.ReverseFind('\\');
        i = sTemp.GetLength() - (i+1);
        sPromptText = sTemp.Right(i);
    }
    // /d option specifies use of WSRegisterWindowData
    // API call instead of older WSRegisterWindowCategory
    // call.
    if (!cmdLineOptions.FlagOptionSpecified(_T("nd")))
    {
        if (!bCategorySpecified)
            sCategory = "picture";
        if (!bWindowSpecSpecified)
            hResult = ::WSRegisterWindowData(
                (LPCTSTR)sCategory,
                (LPCTSTR)sPromptText,
                (LPCTSTR)sCommandLine,
                (LPCTSTR)sWorkingDir,
                m_hwnd );
    }
    else // use old register-category API
    {
        if (bCategorySpecified)
            hResult =
                ::WSRegisterWindowCategory((LPCTSTR)sCategory, m_hwnd );
        Else
            hResult =
                ::WSRegisterWindowCategory((LPCTSTR)"picture", m_hwnd );
    }
    // Log error if need be
    BOOL debug = TRUE;
    if (HR_IS_FAILED(hResult, debug)) ;
    return 0;
}
```

9.3.33 RegisterWindowDataSpec

Description

“RegisterWindowDataSpec” is similar to “RegisterWindowData”, in that it registers a window with a window category and the necessary information required to re-invoke the application display. Also, this call provides the name of a window specification in the current workspace, with which the window is to be managed. Thus, this call does a kind of “temporary global output focus” for this window. The normal output focus indicators are not employed. This API call is NOT REQUIRED BY SAFEVIEW-AWARE DISPLAYS. IT IS DONE AUTOMATICALLY.

Visual C++ code:

```
BOOL RegisterWindowDataSpec(
Long hwnd,
// window handle for window to be registered
LPCTSTR pszWindowSpec,
// points to a string containing a window specification name
LPCTSTR pszCategory,
// points to a string containing the category
LPCTSTR pszPromptText,
// string containing user-prompt to be included in Display History dialog list
LPCTSTR pszCommandLine,
// string containing command line and arguments to re-invoke the display
LPCTSTR pszWorkingDir,
// string containing working directory of application for re-invoke purposes);
```

Visual Basic code:

```
boolean RegisterWindowDataSpec(long window, BSTR WindowSpec, BSTR Category, BSTR Prompt, BSTR
CommandLine, BSTR WorkingDirectory);
```

Parameters Visual C++(VB)

hwnd(Window)

The window handle for the window to be registered.

pszWindowSpec(WindowSpec)

A pointer to a null-terminated string containing a window specification name. This string is used by SafeView to override the MATCH expression and to directly select a window specification for use in managing this display. If the given window specification name is not found in the loaded workspace configuration, the normal matching approach is applied in an effort to disposition the display for management.

pszCategory(Category)

A pointer to a null-terminated string containing a window category. This string is used by SafeView to match an appropriate workspace window specification to this window.

pszPromptText(Prompt)

A pointer to a null-terminated string containing the text included in the Display History dialog for this application display. This string represents this display in the Display History dialog's list.

pszCommandLine(CommandLine)

A pointer to a null-terminated string containing the command line to re-invoke this display. This string is used by SafeView when the operator re-invokes this display from a prior display or from the Display History dialog.

pszWorkingDir(WorkingDirectory)

A pointer to a null-terminated string containing the working directory for this application. This string is included as the working directory parameter when this display is re-invoked from a prior display or from the Display History dialog.

Return value

Returns *true* if the function succeeds.

Remarks

This call is made automatically by SafeView-aware displays when they are invoked. The Visual C/C++ version is “RegisterWindowData.”

See also

“RegisterWindowCategory” on page 135, “RegisterWindowDataSpec” on page 137

Example (Visual C++ OnCreate override)

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    HSI_RESULT hResult;
    BOOL bCategorySpecified = FALSE;
    BOOL bWindowSpecSpecified = FALSE;
    CString sWindowSpec;
    CString sCategory;
    CString sPromptText = _T("");
    CString sCommandLine;
    CString sWorkingDir =
        _T(v:\\display\\workspaces\\test\\pclient\\bitmaps");
    CCmdLineOptions cmdLineOptions;
    // Get command line
    sCommandLine = ::GetCommandLine(); //
    cmdLineOptions.GetCommandLine();
    // Get category if specified, from command line
    if (cmdLineOptions.GetKeywordOptionValue(_T(w""),
        sCategory))
        bCategorySpecified = TRUE;
    if (cmdLineOptions.GetKeywordOptionValue(_T(x""),
        sWindowSpec))
        bWindowSpecSpecified = TRUE;
    // Get document if specified and cut off the path
    // name, leaving just the file name
    if (!cmdLineOptions.GetOptionValue(sPromptText))
        // get document into sPromptText
        sPromptText = _T(P"Client - Untitled");
    Else
    {
        CString sTemp = sPromptText;
        // find last '\\' then set sPromptText to only
        // chars after the '\\'
        int i = sTemp.ReverseFind('\\');
        i = sTemp.GetLength() - (i+1);
        sPromptText = sTemp.Right(i);
    }
    // /d option specifies use of WSRegisterWindowData
    // API call instead of older WSRegisterWindowCategory
    // call.
    if (!cmdLineOptions.FlagOptionSpecified(_T(n"d")))
    {
        if (!bCategorySpecified)
            sCategory = p"icture";
        if (!bWindowSpecSpecified)
            sWindowSpec = "";
        hResult =
            ::WSRegisterWindowDataSpec((LPCTSTR)sWindowSpec,
            (LPCTSTR)sCategory,
            (LPCTSTR)sPromptText,
            (LPCTSTR)sCommandLine,
            (LPCTSTR)sWorkingDir,
            m_hwnd);
    }
    else // use old register-category API
    {
        if (bCategorySpecified)
            hResult =
                ::WSRegisterWindowCategory((LPCTSTR)sCategory, m_hwnd );
        Else
            hResult =
                ::WSRegisterWindowCategory((LPCTSTR)p"icture", m_hwnd );
    }
    // Log error if need be
    BOOL debug = TRUE;
    if (HR_IS_FAILED(hResult, debug))
        return 0;
    }
}
```

9.3.34 .SetCurrentDisplayIndex

Description

“SetCurrentDisplayIndex” is useful in supporting custom “display forward/backward” navigation. This API call sets the highlighted element in the display history list to the index at the given (long) value.

Remarks

DISP_FUNCTION(CWorkspaceClient, “SetCurrentDisplayIndex”, SetCurrentDisplayIndex, VT_BOOL, VTS_I4)

9.3.35 SetFocusBasedSearch

Description

“SetFocusBasedSearch” sets the global search mode to either focus-based search (*true*), or normal search (*false*).

Visual C++ code:

```
BOOL SetFocusBasedSearch(BOOL bFocus);
```

Parameters

bFocus

Set *true* to instruct SafeView to search for where to position windows based on current window focus. Set *false* to instruct SafeView to search for where to position windows based on the top-down arrangement of windows in the workspace configuration file.

Return value

Returns *true* if the function succeeds.

Remarks

This setting is typically made within the workspace itself (focusbasedsearch = yes). It can also be set using the SafeView Control Panel. The Visual C/C++ version is “WSSetFocusBasedSearch.”

Example (Experion display script)

```
dim sv
dim brtn
set sv = CreateObject("Honeywell.workspace.Client")
msgbox sv.SetFocusBasedSearch(true)
set sv = nothing
```

9.3.36 SetGlobalOutputFocus

Description

“SetGlobalOutputFocus” sets the global output focus button-indicator for the given window specification to the given Boolean value.

Visual C++ code:

```
BOOL SetGlobalOutputFocus(
LPCTSTR pszWindowSpec,
BOOL bFocus
);
```

Visual Basic code:

```
boolean SetGlobalOutputFocus(BSTR windowSpec, boolean Focus);
```

Parameters

pszWindowSpec

The pointer to a string containing the name of a window specification, whose global output focus button indicator is to be set.

bFocus

Set *true* if the button is being selected, and *false* if the button is being deselected.

Return value

Returns true if the function succeeds.

Remarks

If the given window specification has MAXWINDOWS = *infinite*, the first-managed window will receive this message. The Visual C/C++ version is “WSSetGlobalOutputFocus.”

9.3.37 SetGlobalOutputFocusOnce

Description

For “SetGlobalOutputFocusOnce,” if you know your SafeView window specification name, (retrievable through recent API call, “wchar_t GetWindowSpecName(long Window).”) you can call this API to set GLOBALFOCUS “>>” override, which enables a one-time global focus (then, global focus is automatically disabled).

This API call helps SafeView to restore displays that should be appropriately positioned, such as on a multi-window flex station during server failover. At present, in that scenario, the displays are randomly positioned when re-invoked. This disorder is particularly confusing on focus-based workspaces. This API call will also be useful when you want to programmatically park a display in a particular position.

Visual Basic code:

```
boolean SetGlobalOutputFocusOnce(BSTR windowSpec);
SetGlobalOutputFocusWnd, SetLockWnd, SetLock, SetOutputFocus, SetOutputFocusWnd
```

Remarks

DISP_FUNCTION(CWorkspaceClient, “SetGlobalOutputFocusOnce”, SetGlobalOutputFocusOnce, VT_BOOL, VTS_BSTR)

Example (Experion display script)

```
Sub OnLButtonClick()
dim bResult As Boolean
dim sv
set sv = CreateObject(“Honeywell.workspace.Client”)
bResult = sv.setglobaloutputfocus(“wimdown1”, True)
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.38 SetGlobalOutputFocusWnd

Description

“SetGlobalOutputFocusWnd” sets the global output focus button for the given window (handle) to the given Boolean value.

Visual C++ code:

```
BOOL SetGlobalOutputFocusWnd(
Long hwnd,
BOOL bFocus
);
```

Visual Basic code:

```
boolean SetGlobalOutputFocusWnd(long window, boolean Focus);
```

*Parameters**hWnd*

A window handle to the window whose global output focus button indicator is to be set.

bFocus

Set *true* if the button is being selected and *false* if the button is being deselected.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is “WSSetGlobalOutputFocusWnd.”

See Also

“SetGlobalOutputFocus” on page 140, “SetLock” on page 142, “SetLockWnd” on page 143, “SetOutputFocus” on page 144, “SetOutputFocusWnd” on page 144

Example (Experion display script)

```
Sub OnButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
hwnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.setglobaloutputfocuswnd(hwnd, TRUE)
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

9.3.39 SetGlobalOutputFocusWndOnce*Description*

“SetGlobalOutputFocusWndOnce” is similar to “SetGlobalOutputFocusOnce,” but with a long window handle given instead of a string name of window specification.

Remarks

DISP_FUNCTION(CWorkspaceClient, “SetGlobalOutputFocusWndOnce”, SetGlobalOutputFocusWndOnce, VT_BOOL, VTS_I4)

9.3.40 SetLock*Description*

“SetLock” sets the lock button for the given window spec to the given Boolean value.

Visual C++ code:

```
BOOL SetLock(
LPCTSTR pszWindowSpec,
BOOL bLock
);
```

Visual Basic code:

```
boolean SetLock(BSTR windowSpec, boolean Locked);
```

*Parameters**pszWindowSpec*

The pointer to a string containing the name of a window specification whose lock button indicator is to be set.

bLock

Set *true* if the button is being selected, and *false* if the button is being deselected.

Return value

Returns *true* if the function succeeds.

Remarks

If the given window specification has MAXWINDOWS = *infinite*, the first-managed window will receive this message. The Visual C/C++ version is “WSSetLock.”

See Also

“SetGlobalOutputFocus” on page 140, “SetLockWnd” on page 143, “SetOutputFocus” on page 144, “SetOutputFocusWnd” on page 144

Example (Experion display script)

```
Sub OnLButtonClick()
dim bResult As Boolean
dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.SetLock("window1", TRUE)
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.41 SetLockWnd

Description

“SetLockWnd” sets lock button for the given window handle to the given Boolean value.

Visual C++ code:

```
BOOL SetLockWnd(
Long hwnd,
BOOL bLock
);
```

Visual Basic code:

```
boolean SetLockWnd(long window, boolean Locked);
```

Parameters

hWnd

The handle to the window whose lock button indicator is to be set.

bLock

Set *true* if the button is being selected, and *false* if the button is being deselected.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is “WSSetLockWnd.”

See Also

“SetFocusBasedSearch” on page 140

Example (Experion display script)

```

Sub OnLButtonClick()
dim sv
dim hwnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
hwnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.setlockwnd(hwnd, TRUE)
if bReturn = True then
// do something
End If
set sv = nothing
End Sub

```

9.3.42 SetOutputFocus*Description*

“SetOutputFocus” sets the group-local output focus button-indicator for the given window specification. This button cannot be deselected; rather, another must be selected.

Visual C++ code:

```

BOOL SetOutputFocus(
LPCTSTR pszWindowSpec,
);

```

Visual Basic code:

```

boolean SetOutputFocus(BSTR windowSpec);

```

*Parameters**pszWindowSpec*

A pointer to a string containing the name of a window specification whose output focus button indicator is to be set.

Return value

Returns *true* if the function succeeds.

Remarks

If the given window specification has MAXWINDOWS = *infinite*, the first-managed window will receive this message. The Visual C/C++ version is “WSSetOutputFocus.”

9.3.43 SetOutputFocusWnd*Description*

“SetOutputFocusWnd” sets the group-local output focus button-indicator for the given window handle. Since this button cannot be deselected; another must be selected.

Visual C++ code:

```

BOOL SetOutputFocusWnd(
Long hwnd,
);

```

Visual Basic code:

```

boolean SetOutputFocusWnd(Long window);

```

*Parameters**hwnd*

The window handle of the window whose output focus button indicator is to be set.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is “WSSetOutputFocusWnd.”

9.3.44 ShowPlaceholders

Description

“ShowPlaceholders” allows an application to ask SafeView to display the placeholders for any window specification configured with `PLACEHOLDERS = true`, and with which no applications are currently being managed. This call emulates clicking the **Show Placeholders** button on the SafeView Control Panel.

Visual C++ code:

```
BOOL ShowPlaceholders();
```

Visual Basic code:

```
boolean ShowPlaceholders();
```

Parameters

None.

Return value

Returns *true* if the function succeeds. Returns *false* if the current workspace was configured without any placeholders.

Remarks

The Visual C/C++ version is “WSShowPlaceholders.”

See Also

“HidePlaceholders” on page 124

Example (Experion display script)

```
Sub OnButtonClick()
dim bResult As Boolean
dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.showplaceholders
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.45 StartManager

Description

“StartManager” allows an application to request the operating system to launch SafeView.

Visual C++ code:

```
BOOL StartManager(
LPCSTR pszWsmArgs
);
```

Visual Basic code:

```
boolean StartManager(BSTR Options);
```

*Parameters**pszWsmArgs*

The command line arguments to be included in the call to invoke SAFEVIEW.EXE.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version of this API is “WSStartWSM.” The SAFEVIEW.EXE executable must be available in the system path.

See Also

“IsManagerActive” on page 126

Example (Experion display script)

This example uses the */f* option followed immediately by a workspace configuration file.

```
Sub OnLButtonClick()
dim bResult AS Boolean
dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.startmanager("/fd:\\wksp\\op_wksp2.wd1")
If bResult = True then
//do something
end if
set sv = nothing
End Sub
```

9.3.46 StopManager

Description

“StopManager” allows an application to ask SafeView to terminate. This call emulates clicking the **Exit Workspace** button on the SafeView Control Panel.

Visual C++ code:

```
BOOL StopManager(
  BOOL boperatorConfirm
);
```

Visual Basic code:

```
boolean StopManager(boolean Confirm);
```

*Parameters**bOperatorConfirm*

Setting this *true* allows the operator to confirm this request using the “Exit SafeView?” YES/NO dialog box. If this is set *true*, the operator can “veto” the exit.

If set *false*, SafeView is closed and the operator has no opportunity to intervene.

Return value

Returns *true* if the function succeeds. The Visual C/C++ version is “WSDisplayHistoryDialog.”

Remarks

None.

See Also

“IsManagerActive” on page 126

Example (Experion display script)

Stop SafeView with operator confirm:

```
Sub OnLButtonClick()
    dim bResult As Boolean
    dim sv
    set sv = CreateObject("Honeywell.workspace.Client")
    bResult = sv.stopmanager(TRUE)
    If bResult = True then
        //do something
    end if
    set sv = nothing
End Sub
```

9.3.47 TerminateOnClose

Description

“TerminateOnClose” terminates the process associated with a SafeView-managed window within a certain time period after the main window has been closed if it does not terminate on its own. This generic SafeView feature has a default value of 30 seconds but is user-configurable.

Visual C++ code:

```
long TerminateOnClose (
    long windowHandle, long DelaySeconds
);
```

Visual Basic code:

```
boolean TerminateOnClose(long window, int nDelaySeconds);
```

Parameters

WindowHandle

The handle of the window that, when closed, triggers the wait time.

DelaySeconds

The wait time after which SafeView terminates the process associated with the closed window if the process did not shut down on its own.

Return value

Returns S_OK (0) when the function succeeds in registering the window handle and delay time with SafeView.

Returns E_INVALIDARG. (0x80070057L) if the given window handle is not valid.

Returns E_FAIL (0x80004005L) if SafeView is not currently executing.

Remarks

This call is available through the existing SafeView COM automation server. The *DelaySeconds* value is automatically constrained to no less than 1 second, and no greater than 300 seconds (5 minutes). If the value is set to less than 1, it will be modified to 1; if the value is set to be greater than 500, it will be modified to 500. The suggested range for the *DelaySeconds* parameter is 10 to 60 seconds.

This API call is not recommended for “editor” type applications or other applications that first prompt the user to confirm or cancel the operation. For example, consider a Microsoft Word application in which the user makes changes, clicks **Close**, and is then prompted to confirm or cancel the Close. If this API call is being used, and the user cancels the Close operation or confirms it, but takes a long time browsing to a location to save the file, the application will still abruptly terminate after the specified delay time.

See Also

“WSTerminateOnClose” on page 150

Example

```
Public sv As Object
Private Sub Command1_Click()
```

```

dim bRes As long 'interpret as HRESULT
bRes = sv.TerminateOnClose(Form1.hwnd, 5)
' bRes = sv.TerminateOnClose(2, 5) 'test bad window
MsgBox bRes
End Sub
Private Sub Command2_Click()
MsgBox Form1.hwnd
End Sub
Private Sub Command3_Click()
MsgBox sv.ismanageractive
' is SafeView running?
End Sub
Private Sub Form_Load()
Set sv = CreateObject("Honeywell.workspace.Client")
End Sub

```

9.3.48 ToggleStandardDialog

Description

“ToggleStandardDialog” allows an application to ask SafeView to show or hide its Control Panel. If the Control Panel is hidden, it is shown, and vice versa.

Visual C++ code:

```
BOOL ToggleStandardDialog();
```

Visual Basic code:

```
boolean ToggleStandardDialog();
```

Parameters

None.

Return value

Returns *true* if the function succeeds.

Remarks

The Visual C/C++ version is “WSToggleStandardDialog.”

Example (Experion display script)

```

Sub OnLButtonClick()
dim bResult As Boolean
dim sv
set sv = CreateObject("Honeywell.workspace.Client")
bResult = sv.togglestandarddialog
If bResult = True then
//do something
end if
set sv = nothing
End Sub

```

9.3.49 UnmanageWindow

Description

Implementing “UnmanageWindow” is unnecessary for most applications. However, you should use this API call if your SafeView-managed application display intends to “hide, but remain executing” in response to a WM_CLOSE message from Windows (or the user clicking the “X” button to close the display).

Visual C++ code:

```

BOOL Unmanagewindow(
Long hwnd
);

```

Visual Basic code:

```
boolean Unmanagewindow(long window);
```

Parameters

hWnd

The handle to the window to be unmanaged.

Return value

Returns *true* if the function succeeds.

Remarks

This API call, already made on behalf of all GUS and Experion displays, releases the SafeView window specification (placeholder in the workspace) for use by others. Failing to make this call in this case would result in an empty space in the workspace, with the hidden window continuing to hold its place in the SafeView workspace. The Visual C/C++ version is “WSUnmanageWindow.”

See Also

“CloseWindow” on page 118

Example 1 (Experion display script)

```
Sub OnButtonClick()
dim sv
dim hWnd
dim bReturn
set sv = CreateObject("Honeywell.workspace.Client")
hWnd = window.parent.external.application.appwindow.hwnd
bReturn = sv.unmanagewindow(hWnd)
if bReturn = True then
// do something
End If
set sv = nothing
End Sub
```

Example 2 (Visual C++)

When a Station display managed by SafeView is closed, it should call the SafeView “UnManageWindow” API call to free up the SafeView placeholder window.

```
pApp->HideApplication();
::WSUnmanagewindow(pApp->m_pMainwnd->m_hwnd);
```

Good practice is to register your hWnd just before being shown, then unregister (WSUnmanageWindow) if you do not plan to actually destroy your window when handling your WM_CLOSE (from either the system or SafeView in a round robin replacement scenario).

9.3.50 WSConfigurationLoaded

Description

“WSConfigurationLoaded” is used to determine if SafeView has a workspace loaded.

Visual C++ code:

```
BOOL WSConfigurationLoaded();
```

Parameters

None.

Return value

Returns *true* if a workspace has been loaded, otherwise, it returns *false*.

Remarks

None.

Example 1 (GUS display script)

```
Sub OnLButtonClick()
dim bResult As Boolean
bResult = workspace.configurationloaded)
End Sub
```

Example 2 (Visual C++)

```
if (!::WSConfigurationLoaded( ))
{
// Load a workspace, etc.
}
```

9.3.51 WSGetWindowSpecification

Description

“WSGetWindowSpecification” allows an application to request, from SafeView, the specific window management data being applied to a currently managed window.

Visual C++ code:

```
BOOL WSGetWindowSpecification(
WSMWINDOWPROPERTIES* pData, Long hWnd
);
```

Parameters

pData

The pointer to a data buffer large enough to hold the structure WSMWINDOWPROPERTIES. The first WORD in the pData record should contain the length in bytes of the buffer. An invalid length signals an invalid version.

hWnd

The handle of the managed window for which the management data is being requested.

Return value

Returns *true* if the function succeeds.

Remarks

This call allows applications to adjust themselves based on the window properties SafeView is using to constrain them. For example, an application can use this API call to determine if it is being managed as an “always on top” window and adjust its menu items accordingly, so that it always reflects the current state. This is in contrast to another application whose menu option can disagree with the currently managed state. This API call is available only through direct Visual C/C++ access.

See Also

“IsManagerActive” on page 126

9.3.52 WSTerminateOnClose

Description

“WSTerminateOnClose” terminates the process associated with a SafeView-managed window within a certain amount of time after the main window has been closed if it does not terminate on its own.

```
Extern C HRESULT WSTerminateOnClose (
HWND WindowHandle, long DelaySeconds
);
```

Parameters

WindowHandle

The handle of the window that, when closed, triggers the wait time.

DelaySeconds

The wait time after which SafeView terminates the process associated with the closed window if the process did not shut down on its own.

Return value

Returns S_OK (0) when the function succeeds in registering the window handle and delay time with SafeView.

Returns E_INVALIDARG. (0x80070057L) if the given window handle is not valid.

Returns E_FAIL (0x80004005L) if SafeView is not currently executing.

Remarks

The *DelaySeconds* value is automatically constrained to no less than 1 second, and no greater than 300 seconds (5 minutes). If the value is set to less than 1, it will be modified to 1; if the value is set to be greater than 500, it will be modified to 500. The suggested range for the *DelaySeconds* parameter is 10 to 60 seconds.

This API call is not recommended for “editor” type applications or other applications that first prompt the user to confirm or cancel the operation. For example, consider a Microsoft Word application in which the user makes changes, clicks **Close**, and is then prompted to confirm or cancel the Close. If this API call is being used, and the user cancels the Close operation or confirms it, but takes a long time browsing to a location to save the file, the application will still abruptly terminate after the specified delay time. This call is available only through direct Visual C/C++ access.

See Also

“WSTerminateOnClose” on page 150

10 Safeview workspace examples

11 Workspace overview

This section provides several SafeView workspace examples. Workspaces with long titles may use abbreviations such as “T” (trend), “FP” (faceplate), “horiz” (horizontal), “cmd” (command), “MS” (manual select), “FB” (focus-based), and “SR” (status right),.

Each workspace is defined as either “focus-based” or not. Focus-based workspaces manage newly opened displays by favoring the location (actually, the owning group) of the current active display. On most of the workspace examples, the Windows taskbar is on the lower edge of the workspace. The workspace constant, *Taskbar_height*, controls the height of the taskbar. You can adjust the constant's value to zero for situations where the taskbar is not intended to be visible (for example, when auto-hidden or occluded).

Some workspace examples may contain windows that are not required for your purposes. In such cases, you may ignore or remove an unwanted window or reconfigure it for your purposes (for example, change a MATCH expression to match your particular application). In any case, unused windows will always be invisible and without affect to the operator, as long as you configure them *without* placeholders.



Tip

Any optional displays, such as the catch-all, trend, and faceplate, are typically never configured to have placeholder (blank) SafeView windows. They are only shown in the workspace examples herein to illustrate the layout of the entire workspace.

In all examples, trends and faceplates are “always on top” so as to never be hidden by the main displays. Also, these displays cannot be minimized, so as to not be lost in the task bar and taking up resources.

11.1 New SafeView APIs

SafeView's "prior display" function now supports the "forward/back" display navigation capability as used in Internet Explorer (for example, "jump" two displays forward or three displays back). This capability is implemented in the form of SafeView API calls that can be utilized by custom navigation displays.

The new SafeView APIs also support one-time, global focus window selection. This is helpful when you want to temporarily override normal SafeView management, but then immediately return to normal SafeView management (for example, to invoke a trend into a main display window, but just one time).

For more information on SafeView APIs, see "Application Program Interface" on page 113"

11.2 Display resolution

Most workspace examples provide a version for each of these display resolutions:

- For 4:3 aspect ratio workspaces:
 - 1280x1024
 - 1600x1200
- For 16:10 aspect ratio workspaces (for wide-screen monitors):
 - 1680x1050
 - 1920x1200

You can define other resolutions by simply adjusting the *xresolution* and *yresolution* constants provided in each workspace example. However, doing so may inadvertently change the aspect ratio of the displays (which is generally okay) but may require you to reconsider the content and layout of your displays.



Attention

Although the HMIWeb Solution Pack offers several display objects that provide specific SafeView-related capabilities (like global focus overrides and cross-screen invocation), it may not yet support the wide-screen resolutions (1680x1050, 1920x1200). Thus, as part of the planning process for your SafeView workspaces, make sure to verify the resolutions supported by the HMIWeb Solution Pack.

11.3 SafeView Text Editor and Graphical Editor

SafeView provides a Text Editor and Graphical Editor that allow you to view, edit, and save an example workspace. However, to retain the flexibility of changing an entire workspace by adjusting only a few constants, avoid editing a window position graphically (that is, moving a placeholder window and then selecting “Update from placeholder”). Doing so will replace the constants from that placeholder with the literal values of its new location.

When using or modifying a workspace example, particularly across different screen resolutions, you should use the Graphical Editor to view the workspace and the Text Editor to edit the constants that describe the windows' positions, regions, and so on. As an alternative, you can build a new workspace from scratch and configure it by using the Graphical Editor to view, drag, and size the placeholders as necessary.

Note: For the main displays that are to remain fixed at all times, it is recommended that you region-constrain these windows to their configured position and size.

For more information on SafeView editors, see “ “Configuring a workspace” on page 65.”

11.4 Faceplates

The faceplates in these example workspaces are constrained to their respective screen areas so that they cannot be dragged to a different screen. In some examples, faceplates are constrained to an area twice their normal size so that, for example, when maximized, they double in size while remaining on the right side of the screen. Only these faceplates are configured as maximizable. See other examples for dedicated space for faceplates that do not occlude main displays at all.

12 Experion single window, single screen

12.1 Single window, single screen workspace description

This single-screen workspace is for Station running in single-window mode. It provides an additional window for running any other kind of display without fully obscuring the Station display. Examples include: TPS Native Window, GUS display, Microsoft Word, or any other Honeywell and/or third-party display. A separate window is available for non-Experion faceplates such as the GUS “classic” faceplate. (Experion faceplates in single-window mode are controlled and positioned by Station and not managed by SafeView.) **This workspace is not focus-based.**

Single window, single screen Comments

This workspace automatically launches Station and allows the user to configure the default display. For advice about launching displays and applications from SafeView at workspace load time, see “ “Populating a workspace automatically on startup” on page 250.” If you do not want to launch Station automatically when SafeView loads this workspace, perhaps because you have already loaded it, you still have the option of launching a set of Station and/or other displays through SafeView's startup commands. For more information, see “ “Station's configured startup page vs. Station startup commands” on page 257.”

12.2 Single window, single screen associated .wdl file names

- *experionsinglewindow_singlescreen_main_catchall_fp_1280x1024.wdl*
- *experionsinglewindow_singlescreen_main_catchall_fp_1600x1200.wdl*
- *experionsinglewindow_singlescreen_main_catchall_fp_1680x1050.wdl*
- *experionsinglewindow_singlescreen_main_catchall_fp_1920x1200.wdl*

12.3 Single window, single screen human interfaces

The following figures illustrate the various interfaces for this workspace.

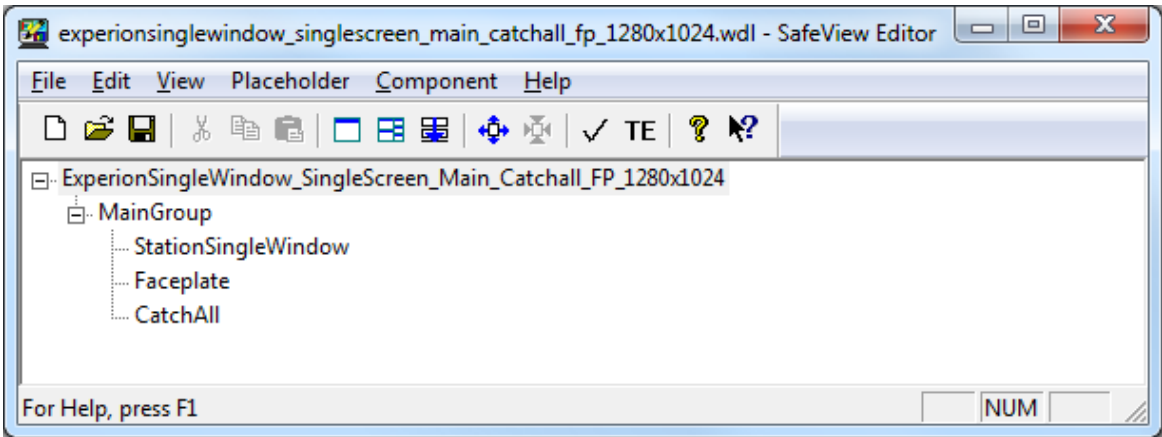


Figure 14: SafeView Graphical Editor

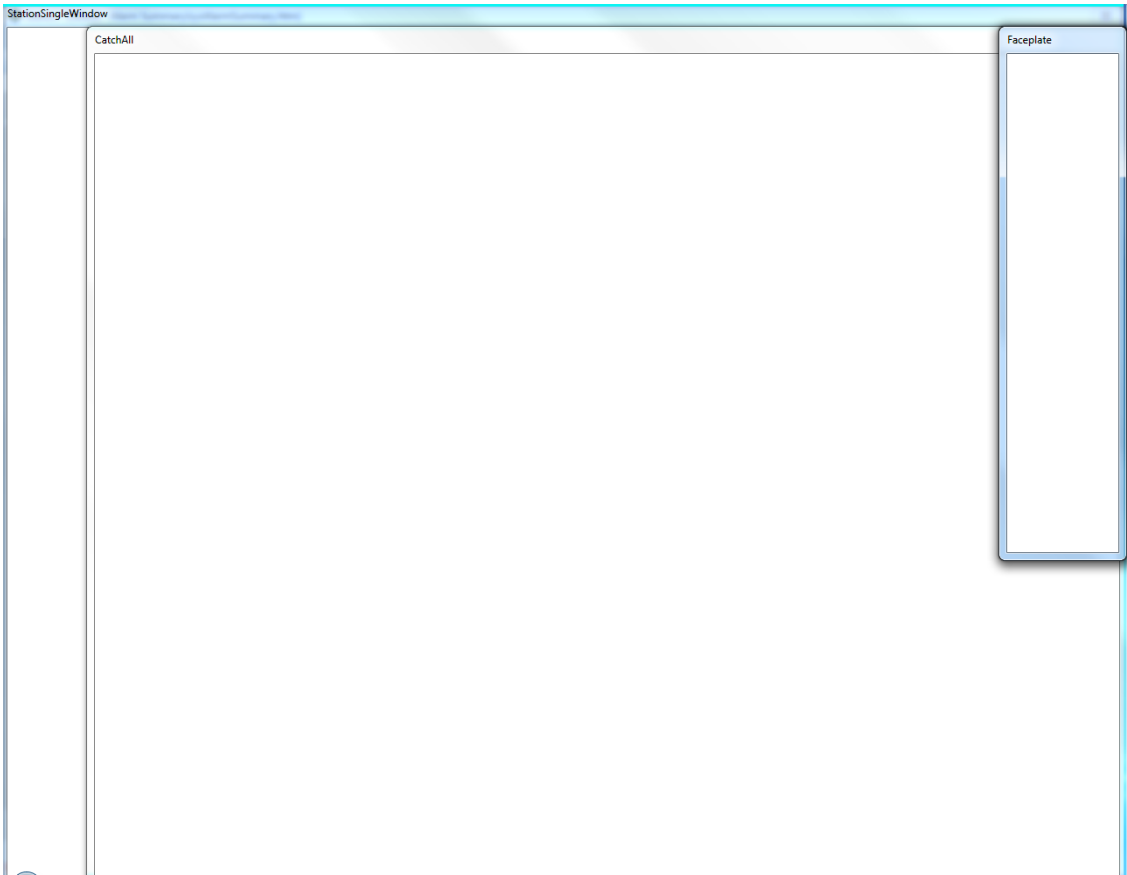


Figure 15: Experion Single Window, Single Screen (4:3 aspect ratio)

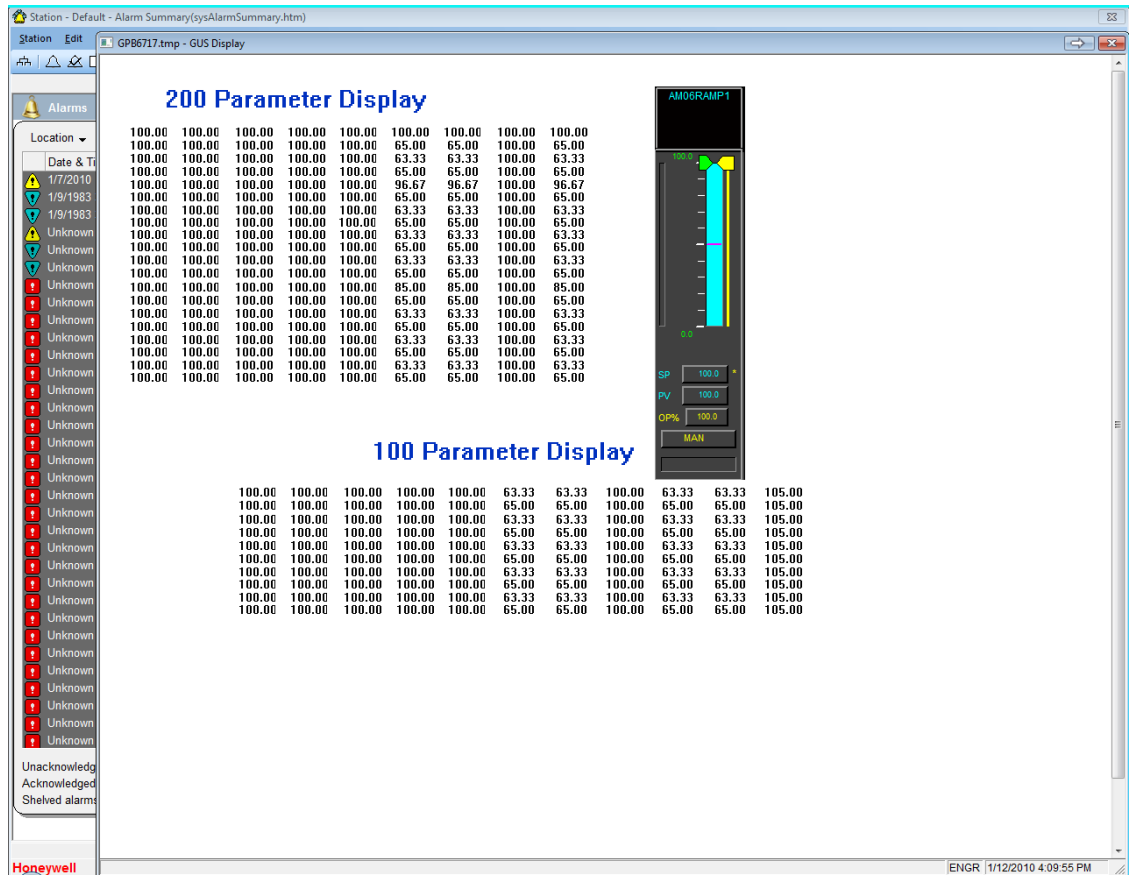


Figure 16: Sample Experion Single Window, Single Screen (4:3 aspect ratio)

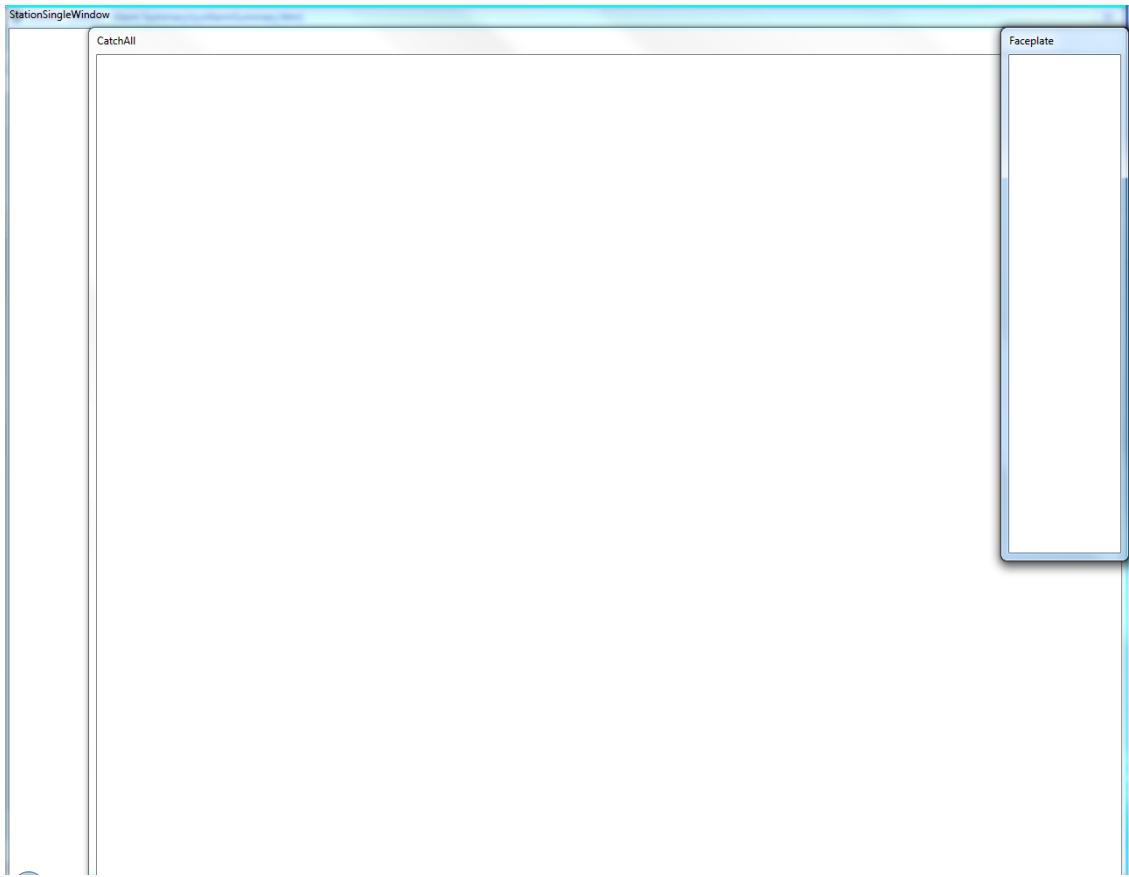


Figure 17: Experion Single Window, Single Screen (16:10 aspect ratio)

13 Experion single window, dual horizontal

13.1 Single window, dual horizontal Workspace description

This single-screen workspace manages Station running in single-window mode on the left screen and Native Window on the right screen. This workspace is similar to “GUS NW catch-all, dual horizontal” on page 169.”
This workspace is *not* focus-based.

Single window, dual horizontal Comments

Typically, a single catch-all window supports one other display at a time. To support multiple catch-all displays, consider the options given in “Supporting multiple catch-all windows” on page 270.”

14 GUS NW catch-all, dual horizontal

14.1 GUS NW catch-all, dual horizontal Workspace description

This GUS workspace manages Native Window on the left screen and GUS displays on the right screen. A catch-all window, provided on the right side, is offset by 100 pixels so as to not fully occlude the main right screen. The catch-all window supports all other Honeywell and/or third party applications. A faceplate window for non-Experion faceplates is also provided on the right side. (In single-window mode, faceplates are directly managed by Station.) This workspace is similar to “Experion single window, dual horizontal” on page 167.” **This workspace is not focus-based.**

GUS NW catch-all, dual horizontal Comments

Typically, a single catch-all window supports one other display at a time. To support multiple catch-all displays, consider the options given in “Supporting multiple catch-all windows.”

15 Experion single window, dual vertical

15.1 Single window, dual vertical workspace description

This workspace manages Station running in single-window mode on the lower screen, while its upper screen supports Native Window (or any other main application) and a catch-all window for all other applications. A GUS faceplate window is also provided. This workspace is similar to “GUS NW dual vertical, catch-all” on page 173.” **This workspace is not focus-based.**

Single window, dual vertical Comments

Experion Station single-window mode faceplates are simple pop-up windows that are not explicitly managed by SafeView, which is why they are excluded from this workspace.

16 GUS NW dual vertical, catch-all

16.1 GUS NW dual vertical, catch-all Workspace description

This GUS workspace manages Native Window on the lower screen, GUS on the upper screen, and a catch-all window for all other applications. A GUS faceplate window is also provided. This workspace is similar to “Experion single window, dual vertical” on page 171.” **This workspace is *not* focus-based.**

GUS NW dual vertical, catch-all Comments

Experion Station single-window mode faceplates are simple pop-up windows that are not explicitly managed by SafeView, which is why they are excluded from this workspace.

17 Experion multi-window, single screen

17.1 Multi-window, single screen Workspace description

This single-screen workspace is for Station running in multi-window mode. This workspace features two faceplates and two trend displays, each in a round robin group, and a catch-all window for all other applications. The catch-all window is offset so that the Station display is not fully occluded. **This workspace is not focus-based.**

Multi-window, single screen Comments

The main station, command, and status windows extend across the width of the monitor. Thus, when faceplates are closed, the display has no unused space. The catch-all window is offset to provide access to the main station window without extending into the faceplate field. In this example, the catch-all window does not extend into the faceplate field; however, you can change this view by adjusting the appropriate constants.

The following figures appear complex as they demonstrate a fully loaded, multi-window workspace on a single screen. Your operators' typical experience may be less complex if they decide not to present every possible display on a single screen.



Tip

To prevent faceplates from losing actual size, only the 16:10 aspect ratio faceplates are allowed to occlude the far right side of the command and status windows (which is typically non-essential space). But you can view this area at any time by moving or closing the faceplate (such as when viewing the log-in priority). This workspace example provides a dedicated faceplate field that does not occlude the main station, command, and status windows.

18 Experion multi-window, single screen, 2FP, dedicated

18.1 Multi-window, single screen, 2FP, dedicated Workspace description

This single-screen workspace for Station has a dedicated space for faceplate windows to prevent occlusion of the main display or trend areas when faceplates are shown. This workspace is similar to “Experion multi-window, single screen” on page 175.” **This workspace is not focus-based.**

Multi-window, single screen, 2FP, dedicated Comments

Use this workspace when faceplates should not occlude main displays. As an exception to the general rule where faceplates do not have placeholders, these faceplates (because of their dedicated space) *are* configured with SafeView placeholder windows. Thus, when faceplates are not in use, a placeholder appears instead of the desktop. Faceplates cannot be moved or resized but can be closed, at which point placeholder windows appear. Only wide-screen examples are provided (1680x1050 and 1920x1200).

Alternatively, you might consider not having faceplate placeholders shown when faceplates are not in use. Instead, consider a modified desktop background that minimizes the distraction of having the desktop shown when faceplates are not in use.



Tip

- As an alternative to offering placeholders, leave this option disabled to reveal the desktop. Then, consider a custom generic desktop that would appear when these faceplates are not in use.
 - If it is critical to not occlude any essential information on main displays, you can simply not place information in the faceplate area of your main displays.
-

19 Experion multi-window, dual horiz, MS, 2T, 4FP

19.1 Multi-window, dual horiz, MS, 2T, 4FP Workspace description

This dual-screen workspace is for Station running in multi-window mode. It also supports other displays such as GUS, Native Window, and other Honeywell and/or third-party displays. Also provided are a separate display “on top” for a trend display, and one faceplate for each monitor, each of which can display an Experion faceplate or a GUS faceplate.

This workspace is similar to that of ““Experion multi-window, dual horiz, MS, hidden taskbar, 2T, 4FP” on page 181” except that the *Taskbar_height* constant in the hidden taskbar version is set to zero. Because the other windows are constant-driven, the other workspace windows resize accordingly to utilize all available screen area. **This workspace is not focus-based.**

Multi-window, dual horiz, MS, 2T, 4FP Comments

In a manual select workspace, click the “>” *output focus* button in the upper right title bar area to assign the main displays to the left or right screen. The faceplates are positioned so as to not cover the main display title bar areas. If the faceplates are too small in 16:10 aspect ratio, consider moving the status bar to the other screen, or using a round robin or focus-based group to control output while enabling larger faceplates.



Tip

For round robin groups, you can control where the next display goes using the > and/or **Lock** buttons. The **Lock** button precludes a given round robin window from being replaced. (Not all windows can be locked simultaneously.)

20 Experion multi-window, dual horiz, MS, hidden taskbar, 2T, 4FP

20.1 Multi-window, dual horiz, MS, hidden taskbar, 2T, 4FP Workspace description

This dual-screen workspace is for Station running in multi-window mode. It supports other displays such as GUS, Native Window, and other Honeywell and/or third-party displays. Also provided are a separate display “on top” for a trend display, and one faceplate for each monitor; each can display an Experion faceplate or a GUS faceplate. This workspace is similar to “ “Experion multi-window, dual horiz, MS, 2T, 4FP” on page 179” except that the *Taskbar_height* constant in the hidden taskbar version is set to zero. Because the other windows are constant-driven, the other workspace windows resize accordingly to utilize all available screen area. **This workspace is not focus-based.**

Multi-window, dual horiz, MS, hidden taskbar, 2T, 4FP Comments

In a manual select workspace, click the > icon in the upper right title bar area to assign the main displays to the left or right screen. The faceplates are positioned so as to not cover the main display title bar areas. If the faceplates are too small in 16:10 aspect ratio, consider moving the status bar to the other screen, or using a round robin or focus-based group to control output while enabling larger faceplates.



Tip

For round robin groups, you can control where the next display goes using the > and/or **Lock** buttons. The **Lock** button precludes a given round robin window from being replaced. (Not all windows can be locked simultaneously.)

21 Experion multi-window, dual horiz, MS, 6T, 5FP

21.1 Multi-window, dual horiz, MS, 6T, 5FP workspace description

This is a dual, wide-screen (16:10) only workspace for Station running in multi-window mode. It supports six trend displays on the right screen, three faceplates on the right screen, and two faceplates on the left screen, all configured in round robin groups. The main displays are in a manual select group, thus allowing the operator to determine where the next main display will be displayed (through the “>” *output focus* button in the upper right title bar area). The main displays can accommodate any Station display or any other display including GUS, Native Window, and third-party. **This workspace is not focus-based.**

Multi-window, dual horiz, MS, 6T, 5FP Comments

This workspace supports six trends but can be configured to support up to twelve trends simultaneously. Faceplates on the right screen support global focus, which should always be used with care (that is, disabled after use). Global focus on these faceplates enables you to load the right screen with trends and faceplates associated with points from the left screen. However, you will need to adjust the number of available Experion processes to support the number of simultaneous displays allowed by this workspace.



Tip

This workspace supports up to six simultaneous trend displays. Other workspace examples may support up to twelve (six per screen, though it is assumed that an operator would not use both screens fully loaded at the same time). You must consider whether your system has sufficient performance and data throughput capability to support this level of simultaneous demand. If not, you can pair down the number of available trends (for example, three per screen) or consider training operators to use discretion when requesting multiple simultaneous displays.



Tip

Remember that you can size and position faceplates to suit your purposes. This is particularly helpful if moving from 4:3 to 16:10 aspect ratios but with shorter physical screens (for example, moving from 1600x1200 to 1680x1050). In this case, the actual size of the faceplates may decrease unless you make appropriate changes as specified in these examples.

22 Experion multi-window, dual horiz, MS, NavWin, 6T, 5FP

22.1 Multi-window, dual horiz, MS, NavWin, 6T, 5FP Workspace description

This is a dual, wide-screen (16:10) only workspace for Station running in multi-window mode. It supports a customizable Navigation Window that can be any display that may have buttons and/or menus for starting various displays. The workspace includes six trend displays on the right screen, three faceplates on the right screen, and two faceplates on the left screen, all configured in round robin groups. The main displays are in a manual select group, thus allowing the operator to determine where the next main display will be displayed (through the “>” output focus button in the upper right title bar area). The main displays can accommodate any Station display or any other display including GUS, Native Window, and third-party. **This workspace is not focus-based.**

Multi-window, dual horiz, MS, NavWin, 6T, 5FP Comments

Faceplates on the right screen have global focus enabled. Use global focus with care because if you forget to disable it, a main display may be forced into a faceplate window (typically, this is why global focus is disabled for faceplates). Global focus enables you to load the right screen with faceplates from the left screen. Optionally, you can place the Navigation Window on top of the right screen, similar to the left command window. For example, use the command window as the Navigation Window and then use dual command windows as described in “ “Experion multi-window, dual horiz, dual cmnd, FB, SR, 4T, 4FP” on page 191.”



Tip

Custom navigation displays can use SafeView API calls to enable (for example) control of “cross-screen” operation through programmatic selection of the manual select output focus. They can also use the new one-time global focus API call that automatically cancels after use.

23 Experion multi-window, dual horiz, MS, SR, 2T, 4FP

23.1 Multi-window, dual horiz, MS, SR, 2T, 4FP Workspace description

This dual-screen workspace is for Station running in multi-window mode. It has two main displays, one on each screen, that support any Station display (except trend and faceplate), or any other Honeywell or third party display. A separate two-window, round robin group is provided on the right screen for Station trend displays. Two round robin faceplates are provided per monitor. **This workspace is not focus-based.**

Multi-window, dual horiz, MS, SR, 2T, 4FP Comments

This workspace places the Status Bar on the right screen thus achieving more consistency between main displays and faceplates per monitor. For even more consistent sizing, see examples that include a Station command window on both the left and right screens.

24 Experion multi-window, dual horiz, FB, SR, 4T, 4FP

24.1 Multi-window, dual horiz, FB, SR, 4T, 4FP Workspace description

This dual-screen workspace is for Station running in multi-window mode. Its main displays support any Station display (except trend and faceplate) or any other Honeywell or third party display. Trends are “cross-screen” in that trends invoked from a display on the left screen appear on the right screen so as to not occlude the main display from which it was invoked. In the 16:10 version left screen, faceplates are larger and fill the height of the screen. System, Group, Detail, and Alarm Summary displays are shown only on the right screen. **This workspace is focus-based.**

Multi-window, dual horiz, FB, SR, 4T, 4FP Comments

To better understand how to customize your workspace to best control how main displays are opened in focus-based mode, see ““Command line options” on page 99.” If you want trends to appear on the screen from which they were opened, simply move each of the trend groups to the other screen group.

**25 Experion multi-window, dual horiz, dual cmnd, FB,
SR, 4T, 4FP**

25.1 Multi-window, dual horiz, dual cmnd, FB, SR, 4T, 4FP Workspace description

This dual-screen workspace is for Station running in multi-window mode. Each main screen has a command window that invokes displays on that screen. The exception is when trends invoked from the left screen appear on the right screen, and vice versa. This workspace accommodates any type of display in either screen, not just Station displays. It supports two trends and faceplates per screen. Positioning the status bar on the right screen allows for more consistent aspect ratio and sizing of main displays across screens. **This workspace is focus-based.**

Multi-window, dual horiz, dual cmnd, FB, SR, 4T, 4FP Comments

This workspace has two command windows. To configure Station accordingly, see ““Supporting multiple command windows in Station” on page 253.”

**26 Experion multi-window, dual horiz, dual cmnd, FB,
SR, 12T, 6FP**

26.1 Multi-window, dual horiz, dual cmnd, FB, SR, 12T, 6FP Workspace description

This is a wide-screen only (16:10) workspace for Station running in multi-window mode. It has two command windows as well as six trends and three faceplates per screen. This workspace is similar to “Experion multi-window, dual horiz, dual cmnd, FB, SR, 4T, 4FP.” **This workspace is focus-based.**

Multi-window, dual horiz, dual cmnd, FB, SR, 12T, 6FP Comments

This workspace has two command windows. To configure Station accordingly, see ““Supporting multiple command windows in Station” on page 253.”

This workspace uses the wide-screen aspect ratio to offer more simultaneous trends. As an exception to the rule, this workspace *does* allow faceplates to have global focus, thus allowing you to utilize faceplates from the opposite side. For example, the right screen can have five trends and three faceplates that support a main display on the left screen.



Tip

This workspace supports up to twelve simultaneous trend displays. Before deploying this workspace, be sure to carefully analyze the impact of having that many trend displays up at once. Your system may not have the performance and data throughput capability to support this scenario. You can reduce the potential performance impact by removing some of the available trends, or by training your operators to use only one screen at a time, and so on. Also, you will need to configure Station to support more than the default number of simultaneously available displays. For instructions on increasing this number, see ““Supporting additional displays in multi-window Station” on page 256.”



Tip

Some users leverage stock scripting in their display “Page OnUnload” events to auto-close any faceplates associated with the active display when the main display is replaced (that is, when a new display is invoked). This same approach, based on scripting calls to the SafeView API, may also be leveraged in this workspace to auto-close trends as well. So, for example, when a new display is invoked from the left command window, the invocation could be accompanied by an automatic closure of trends and/or faceplates on the right. For an example of a faceplate auto-close script, see ““Appendix B: Workspace client Visual C++ code” on page 239.”

As a variation on this theme, a newly invoked display could check for trends and/or faceplates that exist on one side or the other (or both) and, if existing, query the operator to have them closed or not.

**27 Experion multi-window, quad horiz, quad cmd, FB,
SR, 4T, 8FP**

27.1 Multi-window, quad horiz, quad cmnd, FB, SR, 4T, 8FP Workspace description

This quad-screen, horizontal workspace is for Station running in multi-window mode. Each screen has a command window that facilitates focus-based display invocation such that displays invoked from a given command window appear on that screen. Each screen can be customized to support any kind of display such that particular displays only show on particular screens (for example, Native Window or Group displays only show on the right screen, Alarm Summary displays only show on the upper left screen, and so on). **This workspace is focus-based.**

Multi-window, quad horiz, quad cmnd, FB, SR, 4T, 8FP Comments

This workspace has four command windows. To configure Station accordingly, see ““Supporting multiple command windows in Station” on page 253.”

All trends and faceplates are region-constrained to their respective screens. Unlike the dual horizontal examples in which trends are invoked to the opposite screen, in this workspace, each trend is invoked to the screen (active window) from which it was invoked. Alternatively, trends can be invoked to opposite screens by simply copying the trend window specifications in the workspace file accordingly. For example, copying *TrendScreen1* into *Screen2 group*, and *TrendScreen2* into *Screen1 group*, and so on.



Tip

To customize this workspace to support multiple trends per screen, use the Text Editor to copy a trend group from another workspace example, paste it into this group, and then update the location constants accordingly.

28 Experion multi-window, dual vertical, FB, 2T, 4FP

28.1 Multi-window, dual vertical, FB, 2T, 4FP Workspace description

This vertical workspace is for Station running in multi-window mode. Its two main windows accommodate Station displays and other application displays. Each screen supports two round robin faceplates but the upper screen also supports two round robin trends. As a focus-based workspace, newly invoked displays (other than trends which always go to the upper screen) will occupy the screen that currently has the active window. **This workspace is focus-based.**

Multi-window, dual vertical, FB, 2T, 4FP Comments

In this workspace, faceplates are larger because they are allowed to fill the vertical space of each screen (except for the bottom taskbar). That is, they can occlude the right side of the command and status windows (areas that typically do not have interesting data) and can always be viewed by closing a faceplate. Larger faceplates are particularly important for wide-screen (1680x1050) resolution screens because the change from 4:3 resolutions will result in smaller actual size faceplates.



Tip

If dedicating space to faceplates, consider a custom Windows desktop background that minimizes distraction to the operator when faceplates are not in use. See the previous examples that provide dedicated space for faceplates on the right side. Similarly, you can tweak this workspace for wide-screen only such that faceplates have their own field. This will:

- Prevent the command, status, main windows, and other displays from expanding into the faceplate field;
- Help to maintain consistent aspect ratios for (as an example) a library of custom displays in the older 4:3 format (1280x1024 or 1600x1200).



Tip

Focus-based workspaces with only a single command window work well with IKB/OEP or other configured buttons for invoking displays because displays can be easily invoked into the screen without using the command window.

**29 Experion multi-window, dual vertical, dual cmnd,
FB, 4T, 4FP**

29.1 Multi-window, dual vertical, dual cmnd, FB, 4T, 4FP Workspace description

This dual-screen workspace is for Station running in multi-window mode. Each screen has a command window that supports Station displays and other application displays. Each screen also supports two round robin faceplates while the upper screen also supports two round robin trends. As the workspace is focus-based, newly invoked displays (other than trends which always go to upper screen) will occupy the screen that currently has the active window. **This workspace is focus-based.**

Multi-window, dual vertical, dual cmnd, FB, 4T, 4FP Comments

This workspace has two command windows. To configure Experion Station accordingly, see ““Supporting multiple command windows in Station” on page 253.”

Multiple command windows are helpful in a focus-based workspace where the set of displays allowed in each main display overlaps considerably and/or when displays are invoked often without use of IKB/OEP/PC keyboards. You can manage specific displays (such as Group displays) in only the upper (or lower) window by simply excluding that window's category from the other window's MATCH expression. That is, adding the *and not category* (“*HW_System_Group*”) to the MATCH expression of the upper window will cause Group displays to always appear in the lower window and vice versa.



Tip

You can modify station command windows to create your own menus and toolbar buttons, thus creating your own custom Navigation Bar.

**30 Experion multi-window ICON, dual cmnd, FB, SR,
8T, 8FP**

30.1 Multi-window ICON, dual cmnd, FB, SR, 8T, 8FP Workspace description

This is a quad-screen ICON workspace for Station running in multi-window mode. As with most examples, this workspace accommodates any type of display in each of the four screens, not just Station displays. It supports four round robin trends in each upper screen and two round robin faceplates in each of the four screens. Each lower screen has a command window that invokes displays that are then displayed on that screen or that side (left or right). **This workspace is focus-based.**

Multi-window ICON, dual cmnd, FB, SR, 8T, 8FP Comments

This workspace has two command windows. To configure Station accordingly, see “Supporting multiple command windows in Station” on page 253.”

The dual command zones in a focus-based workspace provide an intuitive means to launch displays into the lower left and right screens and/or the left and right sides. As in all workspaces, there is ample capability to customize such that specific displays (such as Alarm Summary, system displays, and various custom graphics) may appear on specific screens, such as the upper or lower tier only.



Tip

You can use wildcards to make your MATCH expressions more generic. For example, instead of using `category("HW_System_Alarm_Summary")` you can use `category(".*summary?")` MATCH expressions are not case-sensitive, unless SafeView is launched with the `/cs` command line option.

31 Experion multi-window ICON, dual cmnd, FB, SR, 12T, 8FP

31.1 Multi-window ICON, dual cmnd, FB, SR, 12T, 8FP Workspace description

For wide-screen only, this quad-screen ICON workspace is for Experion Station running in multi-window mode. It has two command windows and supports up to six trends for each upper screen. **This workspace is focus-based.**

Multi-window ICON, dual cmnd, FB, SR, 12T, 8FP Comments

This workspace has two command windows. To configure Station accordingly, see ““Supporting multiple command windows in Station” on page 253.”

See applicable comments for other examples regarding important performance impact and data throughput considerations with twelve available trends.

32 Experion multi-window ICON, dual cmd, FB, SR, 12T, 12FP

32.1 Multi-window ICON, dual cmnd, FB, SR, 12T, 12FP Workspace description

For wide-screen only, this quad-screen ICON workspace is for Station running in multi-window mode. It has two command windows and supports up to six trends on each upper screen. This workspace can support a large number of simultaneous trend and faceplate displays (if so chosen by the operator) but protects against excessive simultaneous main displays, as those are limited to the four main windows. Therefore, it requires an increase in the number of Station processes (described elsewhere in this document).

Note how this workspace supports a different number and arrangement of faceplates. The command window on each lower screen invokes displays that are then shown on that screen or that side (left or right). This workspace demonstrates the ease with which one can decide to control specific displays to either the upper or lower tier. Thus, all Group displays (category *HW_System_Group*) are managed only on the upper screens, and all system displays (category *HW_System_Display*) are managed only in the lower tier. See the following comments for implementation details. **This workspace is focus-based.**

Multi-window ICON, dual cmnd, FB, SR, 12T, 12FP Comments

This workspace has two command windows. To configure Station accordingly, see “Supporting multiple command windows in Station” on page 253.”

To accommodate group displays only on the upper screens and other system displays only on the lower screens, we exclude each from the MATCH expressions of the other set of main displays from this workspace as follows:

```
window MainDisplayLowerLeft
match = module("?*") and not category("?*faceplate?*") and not category("?*trend?*") and
category("?*HSC_Station*") and
not category("HW_System_Group");
window MainDisplayLowerRight
match = module("?*") and not category("?*faceplate?*") and not category("?*trend?*") and
not category("?*HSC_Station*") and not category("HW_System_Group");
window MainDisplayUpperLeft
match = module("?*") and not category("?*faceplate?*") and not category("?*trend?*") and
category("?*HSC_Station*") and not category("HW_System_Display");
window MainDisplayUpperRight
match = module("?*") and not category("?*faceplate?*") and not category("?*trend?*") and
category("?*HSC_Station*") and not category("HW_System_Display");
```



Tip

This (or any) workspace supports automatic “yoking” of faceplates to main display invocations. That is, when a custom graphic is invoked, its own OnPageComplete event could be scripted to automatically invoke one, two, or (in this example) three associated faceplates. These would go into the faceplate field (same screen) associated with the new main display. A benefit of yoking would be insurance that any previous and/or unrelated faceplates would be replaced. For an example, see “Yoking faceplates to main displays” on page 265.”

33 Experion multi-window ICON, quad cmd, FB, SR, 8T, 8FP

33.1 Multi-window ICON, quad cmnd, FB, SR, 8T, 8FP Workspace description

This quad-screen ICON workspace is for Station running in multi-window mode. As with most examples, this workspace accommodates any type of display in each of the four screens, not just Station displays. It supports four round robin trends in each upper screen and two round robin faceplates in each of the four screens. Each of the four screens has a command window that invokes displays that are then displayed on that screen or that side (left or right). **This workspace is focus-based.**

Multi-window ICON, quad cmnd, FB, SR, 8T, 8FP Comments

This workspace has four command windows. To configure Station accordingly, see ““Supporting multiple command windows in Station” on page 253.”

With a command window in each of the four screens, focus-based invocation of generic main displays is consistent across all screens. Faceplates are larger as they are allowed to occlude the right part of a command window, which would typically not be a problem. You can manage particular types of displays (such as groups, details, and your custom displays) to the left or right sides, or upper or lower tiers, or to specific screens, simply by excluding them from the other main display MATCH expressions.

34 EPKS single screen

34.1 EPKS single screen Workspace description

This updated example from earlier releases of the Experion Station product supports Station running in multi-window mode. It has a single physical screen with four main screens. The upper left screen shows Alarm Summary displays and the upper right screen shows Trend and Group displays. The lower left screen shows detail displays and the lower right screen shows all other Honeywell/Experion displays and includes one faceplate. The upper edge has a command window and the lower edge has a status window. **This workspace is not focus-based.**

EPKS single screen Comments

This workspace does not manage non-Honeywell applications. Such displays simply appear and are unaffected by SafeView. You can modify which displays are supported in each of the main screens by simply modifying the MATCH expressions of each one.

35 EPKS dual screen

35.1 EPKS dual screen Workspace description

This updated example from earlier releases of the Experion Station product supports Station running in multi-window mode. The left screen shows Trend and Group displays, Details, and Alarm Summary displays, while the right screen shows all other Honeywell/Experion displays and includes one faceplate. The upper left edge has a command window and the lower left edge has a status window. **This workspace is not focus-based.**

EPKS dual screen Comments

This workspace does not manage non-Honeywell applications. Such displays simply appear and are unaffected by SafeView.

36 EPKS quad 1x4 screen

36.1 EPKS quad 1x4 screen Workspace description

This updated example from earlier releases of the Experion Station product supports Station running in multi-window mode. The left screen shows Details while the middle left screen shows all other Honeywell/Experion displays and includes one faceplate. The middle right screen shows Alarm Summary displays while the right screen shows Trend and Group displays. The upper left edge has a command window and the lower left edge has a status window. **This workspace is not focus-based.**

EPKS quad 1x4 screen Comments

This workspace does not manage non-Honeywell applications. Such displays simply appear and are unaffected by SafeView. You can modify which displays are supported in each of the main screens by simply modifying the MATCH expressions of each one.

37 EPKS quad 2x2 screen

37.1 EPKS quad 2x2 screen Workspace description

This updated example from earlier releases of the Experion Station product supports Station running in multi-window mode. The upper left screen shows Details while the upper right screen shows Trend and Group displays. The lower left screen shows all other Honeywell/Experion displays while the lower right screen shows Alarm Summary displays and includes one faceplate. The lower left screen has a command window and a status bar. **This workspace is not focus-based.**

EPKS quad 2x2 screen Comments

This workspace does not manage non-Honeywell applications. Such displays simply appear and are unaffected by SafeView. You can modify which displays are supported in each of the main screens by simply modifying the MATCH expressions of each one.

38 Appendix A: Workspace Definition Language

Related topics

- “Workspace Definition Language files” on page 218
- “Syntax notation” on page 219
- “Lexical conventions” on page 220
- “Workspace syntax and semantics” on page 221
- “Group syntax and semantics” on page 222
- “Window syntax and semantics” on page 224
- “MATCH expressions syntax and semantics” on page 228
- “Expressions syntax and semantics” on page 232
- “Regular expression pattern syntax” on page 234
- “Predefined constants” on page 236
- “Miscellaneous” on page 237

38.1 Workspace Definition Language files

Workspaces are designed by configuration files that are human-readable UNICODE text files. These configuration files are built with the SafeView Text Editor, using a text-based language called the Workspace Definition Language (WDL).

Essentially, WDL is a set of syntactic and semantic rules that govern the correct serialization (reading/writing) of any workspace configuration. Because SafeView's behavior is completely controlled by the currently active workspace configuration, this definition of the WDL provides a formal description of the behavior of SafeView.

To manage displays, the workspace server must read in a workspace configuration at startup time. This file tells SafeView how to manage application windows running on an operator station. Only one configuration can be active at a time. Typically, if different users or classes of users are required to perform different functions, configuration files will have to be built for a number of different workspaces.

38.1.1 WDL file format

A WDL file contains three kinds of information required by SafeView:

- Attributes of the workspace;
- Window groups that set the mode in which the windows they contain are managed;
- Window specifications that contain management properties for real windows.

The general format of a WDL workspace configuration file is as follows:

```
WORKSPACE <workspace name>...<workspace description>
...<focusbasedsearch option> <application startup list>
<number and string constants> GROUP <top-level group name>
...<group description> <sub groups> WINDOW <window
name> <window description> <window specification properties>
END WINDOW END GROUP END WORKSPACE
```

38.2 Syntax notation

In the syntax notation used here, syntactic categories are indicated by *italics*. Keywords are indicated by characters in UPPERCASE. An optional item is within brackets “[]”. Items that can be repeated zero or more times are within braces “{ }”. Required punctuation is enclosed in apostrophes (“ ”). Alternatives are listed on separate lines.

38.3 Lexical conventions

This section lists the lexical conventions that SafeView configuration files must follow.

38.3.1 Tokens

There are five kinds of tokens: identifiers, keywords, literals, operators, and other separators. Blanks, tabs, new-lines, and other common forms of “white space” are ignored except as they serve to separate tokens.



CAUTION

Users should document SafeView workspace entries through the use of workspace item descriptions (**Description** = “**comments**”), rather than with “/* */” delimited comments. The workspace item **Description** = “**comments**” is recognized by both SafeView editors, while any “/*” delimited comments are discarded when the .wdl file is read into the standard graphical SafeView Editor. If this behavior is not a concern, the characters /* may start a comment that terminates with the characters */. The characters // may start a comment that terminates with the end of a line.

Identifiers

An identifier is an arbitrarily long sequence of letters and digits. The first character must be a letter. Upper and lower case letters are all different.

Keywords

Some identifiers are reserved for use as keywords. Keywords, which are *case-insensitive*, define specific language constructs, and thus cannot be used as names by the user.

Literals

Literals include integer constants, floating point constants, and string literals. Most “C” style escape sequences are allowed in string literals.

Strings

A string is a series of 1 to 500 ASCII characters enclosed by quotes, such as “Flag01.”

38.4 Workspace syntax and semantics

This section presents the syntax and semantics of language elements related to configuring general workspace attributes.

38.4.1 Workspace Syntax

| | |
|-------------------------------|---|
| <i>workspace:</i> | WORKSPACE <i>workspace_name</i> { <i>workspace_defn_item</i> } END WORKSPACE |
| <i>workspace_name:</i> | <i>Identifier</i> |
| <i>workspace_defn_item:</i> | [<i>workspace_description</i>] [<i>focusbasedsearch</i>] [<i>startup_specification</i>] [<i>constant_declaration</i>] <i>window_group</i> |
| <i>workspace_description</i> | description '=' <i>string</i> ';' |
| <i>Focusbasedsearch:</i> | focusbasedsearch '=' <i>bool</i> ';' |
| <i>startup_specification:</i> | STARTUP [<i>string</i> {';' <i>string</i> }] END STARTUP |
| <i>constant_declaration:</i> | NUMBER <i>identifier</i> '=' <i>constant_expression</i> ';' STRING <i>identifier</i> '=' <i>string_expression</i> ';' |
| <i>window_group:</i> | |

38.4.2 Workspace Semantics

startup_specification

The workspace startup specification contains zero or more *command-line* strings, each string separated by a comma. Each string must contain a file path for an executable file, followed optionally by any arguments needed by the program. The workspace will invoke the programs listed in the startup block immediately after window management is initiated. The programs are invoked in order of occurrence in the list by passing the string to the “CreateProcess” API call as the *lpzCommandLine* parameter. However, the application displays can appear out of order because one application can actually appear before another one, even though it was invoked just slightly after the first one.

constant_declaration

A workspace can contain zero or more constants. A constant is a user-defined identifier that represents a constant numeric or string value. A numeric constant declaration begins the keyword NUMBER. A string constant declaration begins with the keyword STRING. The scope of the constant is from the point of declaration to the end of the file. A numeric constant can be used wherever a number can appear. A string constant can be used wherever a string value can appear. The workspace provides some predefined constants as listed in ““Predefined constants” on page 236.”

window_group

A workspace must have one, and only one, top-level window group such as described in the section that follows.

38.5 Group syntax and semantics

This section describes the syntax and semantics of language items related to the definition of a window group.

38.5.1 Group Syntax

| | |
|---------------------------|--|
| <i>window_group:</i> | <i>GROUP group_name</i> <i>('group_behavior')</i> <i>{window_group_item} END GROUP</i> |
| <i>group_name:</i> | <i>identifier</i> |
| <i>group_description</i> | description '=' <i>string</i> ',' |
| <i>group_behavior:</i> | <i>string</i> |
| DEFAULT | '=' <i>boolean_expression</i> ',' |
| MATCH | '=' <i>match_expression</i> ',' |
| REGION | '=' <i>windowbox</i> ',' |
| <i>window_group_item:</i> | <i>window_specification</i> <i>window_group</i> |

38.5.2 Group Semantics

group_name

The group name item supplies a name for the group. The group name must be unique within the workspace. The lexical rules for a group name are that of an identifier.

group_behavior

Group behavior is a string containing a behavior identifier. Behavior identifiers are predefined by Honeywell. The behavior identifier determines the behavior of the group. There are four aspects of group behavior:

- **Matching Policy:** How a group is searched to find a window specification for a newly created window;
- **Nested Group Policy:** Whether or not nested groups are allowed and, if so, the meaning of a nested group;
- **Replacement Policy:** A group behavior can override the MAXWINDOWS attribute of contained window specifications to enforce a different, group-wide replacement policy;
- **Window Management Policy:** A group can add window management capabilities.

The group behavior applies only to immediately contained group items; it does not propagate down into contained groups. The following summarizes the four aspects of group behavior for each group type: **first match**, **round robin**, and **manual select**.

First match group behavior

| | |
|-----------------------------|---|
| Matching policy: | Search the group starting with first item in the list. The first successful match terminates the search. When groups are encountered in a search, search that group according to its matching policy. |
| Nested group policy: | Allowed. Nested groups participate in a search for a window specification. See matching policy. |

| | |
|----------------------------------|---|
| Replacement policy: | Search the group starting with first item in the list. The first successful match terminates the search. When groups are encountered in a search, search that group according to its matching policy. |
| Window management policy: | Use window management defined in window specifications. |

Round robin group behavior

| | |
|----------------------------------|--|
| Matching policy: | Matching occurs at the group level. That is, there is only one MATCH expression for all windows in a round robin group. A “next window specification” concept is maintained by the group. If this window specification is currently associated with an application or a placeholder, this window is seen by the user as “window of output focus.” Upon a group-level match, the “next window specification” is used to manage the window and the next window specification in the round robin group (or to the first one if the current window specification is the last one listed) becomes the “next.” |
| Nested group policy: | Not allowed. |
| Replacement policy: | The least-recently-used window specification is used. |
| Window management policy: | Use window management defined in window specifications. |

Manual select group behavior

| | |
|----------------------------------|--|
| Matching policy: | Exactly similar to round robin group (above) except that there is no automatic incrementing of the “next” window specification. The user manually selects a given window to change the window of output focus. |
| Nested group policy: | Not allowed. |
| Replacement policy: | The window specification currently associated with the window of output focus is used. |
| Window management policy: | Use window management defined in window specifications. |

window_group_item

Window group items are the items contained in the group. A group contains a list of one or more group items. Group items can be window specifications or other (sub)groups. When a group is contained within another group, it is said to be a nested group. Round robin and manual select groups do not support nested groups; however they do support DEFAULT, MATCH, and REGION group items as described in the ““window_property”” topic that follows.

38.6 Window syntax and semantics

This section describes the syntax and semantics of language elements related to the definition of a window specification.

38.6.1 Window Syntax

| | |
|------------------------------|--|
| <i>window_specification:</i> | WINDOW <i>window_spec_name</i> [<i>window_description</i>] { <i>window_property</i> } END WINDOW |
| <i>window_spec_name:</i> | <i>identifier</i> |
| <i>window_description</i> | description '=' <i>string</i> ',' |
| <i>window_property:</i> | ALWAYS ON TOP '=' <i>boolean_expression</i> ',' CLOSABLE '=' <i>boolean_expression</i> ',' DEFAULT '=' <i>boolean_expression</i> ',' DRAGGABLE '=' <i>boolean_expression</i> ',' GLOBALFOCUS '=' <i>boolean_expression</i> ',' MATCH '=' <i>match_expression</i> ',' MAXPOS '=' <i>point</i> ',' MAXSIZE '=' <i>size</i> ',' MAXWINDOWS '=' <i>expression</i> ',' MINIMIZABLE '=' <i>boolean_expression</i> ',' MINSIZE '=' <i>size</i> ',' PLACEHOLDER '=' <i>boolean_expression</i> ',' POSITION '=' <i>point</i> ',' REGION '=' <i>windowbox</i> ',' SIZE '=' <i>size</i> ',' SIZEABLE '=' <i>boolean_expression</i> ',' |

38.6.2 Window Semantics

window_spec_name

The window specification name item supplies a name for the window specification. The window specification name must be unique within the workspace. The lexical rules for a window specification name are that of an identifier.

window_property

Window properties are configurable values that control many aspects of window behavior and appearance. Most window properties do not need to be specified if the default value is acceptable.

The following table gives the meaning and possible values for all of the window properties. The “Values” column also gives the default value supplied by SafeView if the window property is not specified. For more information on window properties, see ““Window group items”.”

| Name | Meaning | Values |
|---------------|--|--|
| ALWAYS ON TOP | If <i>true</i> , the window will be “always on top” unless it is overlaid by another “always on top” window. If <i>false</i> , it will have normal window z-order behavior. | <i>true/false</i> , defaults to <i>false</i> . |
| CLOSABLE | If <i>false</i> , the system Close command for the application is disabled. If <i>true</i> , it is not. | <i>true/false</i> , defaults to normal Windows behavior. |
| DEFAULT | If <i>true</i> , the MATCH expression for this window becomes the default MATCH expression for all registered windows (workspace client displays associated with a given category using the WSRegisterWindowCategory API) that do not match their workspace configuration. If <i>false</i> , the MATCH expression is ignored. | <i>true/false</i> , defaults to <i>false</i> . Only one DEFAULT = <i>true</i> can appear in a workspace configuration file. |
| DRAGGABLE | If <i>true</i> , the window can be moved dragging. If <i>false</i> , the window cannot be repositioned by dragging. | <i>true/false</i> , default to <i>true</i> . |
| GLOBAL FOCUS | If <i>false</i> , the window is configured such that the global focus button is not available. If <i>true</i> , it is available. | <i>true/false</i> , default is <i>true</i> . |
| MATCH | This is an expression, rather than a property that defines the logic SafeView uses to determine if a given window should be managed in this window group. At workspace run time, when the creation of a top-level window is detected, the properties of the newly created window are compared to each window specification in the expression until a match is found. | expression. If no MATCH expression is specified, the window specification will not participate in the automatic matching of windows. |
| MAXPOS | The maximized position (upper left coordinates) of the specified window. | x, y, defaults to normal Windows behavior. |
| MAXSIZE | The maximum width and height allowed for the specified window. The window must fit inside REGION, if region is configured. | width, height, defaults to normal Windows behavior. For first match and manual select, minimum height is 26 pixels and minimum width is 102 pixels. For round robin, minimum height is 26 pixels and minimum width is 120 pixels. |
| MAXWINDOWS | The maximum number of real windows that can exist with this window specification. When the number is exceeded, the oldest window using this window specification is sent a WM_CLOSE message. The keyword “INFINITE” can be used, assuring that no windows managed according to this window specification will be automatically closed by SafeView. | 1/infinite, defaults to 1. This property is applicable only for window specifications in first match window groups. If you try to use it for any other group type, you will get a syntax error. |
| MINIMIZABLE | If <i>false</i> , the ability to minimize (iconize) this window is removed. If <i>true</i> , the minimize function is not changed in any way. | <i>true/false</i> , defaults to normal Windows behavior. |
| MINSIZE | The minimum size for a window. | width, height, defaults to normal Windows behavior. For first match and manual select, minimum height is 26 pixels and minimum width is 102 pixels. For round robin, minimum height is 26 pixels and minimum width is 120 pixels. |

| Name | Meaning | Values |
|-------------|--|--|
| PLACEHOLDER | If <i>true</i> , SafeView will display a placeholder for this window specification when no actual application display is currently being managed. If <i>false</i> , the workspace will not display a placeholder. See the placeholder concept below. | <i>true/false</i> For first match and round robin groups, defaults to <i>false</i> . For manual select groups, defaults to <i>true</i> , the only legal value. Setting it to <i>false</i> would make it impossible to select this window. For a new window, the default placeholder size is 768 x 518 pixels. |
| POSITION | The initial position (upper-left coordinates) of the window. | x,y defaults to the position specified by the operating system or the application. If this position violates other window constraints, such as region, the window position is modified to make it conform. |
| REGION | This value takes precedence over a region specified at the group level. See the region concept below. | x, y, width, height, defaults to the region specified at the group level. If region is not specified for the group, the default is to the normal Windows behavior, the entire display surface. |
| SIZE | The initial width and height of the window. | width, height defaults to the size specified by the operating system or application. If this size violates other window constraints, such as region, the window size is modified to make it conform. For first match and manual select, minimum height is 26 pixels and minimum width is 102 pixels. For round robin, minimum height is 26 pixels and minimum width is 120 pixels. |
| SIZEABLE | If <i>true</i> , the window can be resized by moving the window's frame. If <i>false</i> , the Maximize button will not change the size of the display, but normal Windows Minimize and Maximize functions are still available. The window style is modified such that the standard "thick-frame" and "size-handles" are removed. Note that it is possible for an application to reestablish its sizable window style (thick frame, size-handles). | <i>true/false</i> defaults to <i>true</i> , with normal Windows, or application-determined behavior. |

38.6.3 The region concept

A window specification can optionally contain a REGION attribute to define a rectangle that constrains the position of the window. If the window is moveable, it can be moved only within the specified region. Several window specifications can have overlapping regions. The POSITION attribute must specify a rectangle inside the region. The dimensions of the region override the combination of MAXPOS and MAXSIZE, when a window is maximized. MAXSIZE cannot exceed the size of the region.

38.6.4 The placeholder concept

A placeholder is a visual representation of a window specification. The use of placeholders is selected by the `PLACEHOLDER` window specification attribute. A placeholder is a window displayed in the workspace that shows the location and some properties of the window specification. A placeholder is displayed only if there are no real windows associated with the window specification. A placeholder is useful in groups that show output focus, or that allow operator control of the selection of window specifications for real windows. It provides a location on the screen for the operator selection gestures and provides a visual representation of the workspace.

38.6.5 Meaning of coordinate values

The coordinate values are relative to the upper left corner of the display, which has the coordinates (0, 0).

38.6.6 Multiply specified properties

It is an error to specify the following properties more than once within the specification for a single window:

`ALWAYS ON TOP`, `DRAGGABLE`, `MAXPOS`, `MAXSIZE`, `MAXWINDOWS`, `MINSIZE`, `PLACEHOLDER`, `POSITION`, `REGION`, `SIZEABLE`.

When a `TITLE` or `CATEGORY` property is specified more than once in a single window specification, the lists from each occurrence are concatenated to form a single list for the window specification.

38.6.7 Relationships between `SIZE` and `POSITION` attributes

The values specified for `REGION`, `POSITION`, `MAXPOS`, and `MAXSIZE` must conform to the following rules:

- Size of the `POSITION` windowbox must be less than or equal to `MAXSIZE`.
- Size of the `POSITION` windowbox must be greater than or equal to `MINSIZE`.
- The `POSITION` table must be within or on the boundary of `REGION`, if `REGION` was specified.
- Size of the `REGION` windowbox must be greater than or equal to `MAXSIZE`.
- `MAXPOS` must fall within the `REGION` windowbox.

38.7 MATCH expressions syntax and semantics

This section presents the syntax and semantics of MATCH expressions. MATCH expressions are used in the MATCH attribute of window specifications and window groups. A MATCH expression specifies the conditions that must be true for a window specification to be used to manage a real window.

At run time, when the creation of a window is detected, SafeView searches for a MATCH expression that evaluates to *true* based on the properties of the real window. The first MATCH expression that matches the given application window is used to determine the window specification to manage the real window. The workspace uses a depth first search of the workspace configuration.



Attention

Displays with the Category property configured to the empty string, can only be managed by title and/or module. They cannot be managed by category nor by the DEFAULT statement in a SafeView configuration.

It is best to always apply a valid category to displays. If no other category is desired, set the category to “ ” or “blank”.

The following characters are not allowed in the category field of Display/Properties: ‘?’, ‘*’, ‘\’, or ‘”’

38.7.1 MATCH expressions Syntax

| | |
|---------------------------|--|
| <i>match_expression</i> : | <i>match_expression</i> OR <i>match_and_expr</i> <i>match_and_list</i> |
| <i>match_and_list</i> : | <i>match_and_list</i> AND <i>match_unary</i> <i>match_unary</i> |
| <i>match_unary</i> : | ‘(<i>match_expression</i>)’ NOT <i>match_unary</i> <i>match_operator</i> |
| <i>match_operator</i> : | CATEGORY ‘(<i>string_expression</i>)’ TITLE ‘(<i>string_expression</i>)’ MODULE ‘(<i>string_expression</i>)’ |

38.7.2 MATCH expressions Semantics

match_expression

A MATCH expression is a Boolean expression, which means that it evaluates to *true* or *false*. A MATCH expression is evaluated at workspace run time, during the search for a window specification to use to control a newly created real window. The search process is described elsewhere in this document.

match_operator

Match operators are specified for category, title, and module MATCH expressions. Match operator operands can contain regular expressions. During the search for a window specification to be used to control a newly created real window, the strings representing the actual category, title, or module of a real window are compared to the regular expression pattern in the operand of the match operator, and a *true* or *false* is returned. If a *true* is returned, the match operator operands are used to control the window. If a *false* is returned, the strings representing the actual category, title, and module of a real window are used. The following describes the match operators in detail.

- **Category:** Compares the category of the real window to the value of the string expression that is the operand of the category match operator. The string expression can contain a regular expression. Returns *true* if the category matches the regular expression pattern in the string expression and *false* if it does not. If the real window is not owned by a SafeView-aware workspace application, it will not have a category. The

category match operator returns *false* for windows that do not have a category. Note also that for this match operator, null or empty strings are *not* valid.

- **Title:** Compares the title (caption) of the real window to the value of the string expression that is the operand of the title match operator. The string expression can contain a regular expression. Returns *true* if the window title matches the regular expression pattern in the string expression and *false* if it does not.
- **Module:** Compares the module file name (the full path of the executable file that was loaded for the process that owns the real window) of the process that owns the real window to the value of the string expression that is the operand of the module match operator. The string expression can contain a regular expression. Returns *true* if the module file name matches the regular expression pattern in the string expression and *false* if it does not. The module file name is the full path of the executable file that was loaded for the process that owns the real window. Because the full path is always provided by the system, consider using `match = module ("?myprog.exe")`.

38.7.3 Example MATCH expressions

See ““Safeview workspace examples” on page 153” for complete workspace definition files.

Example 1: Titles and any category

```
MATCH = category("?*") or title("Mine?") or
title("FreeCell") or
title("?*PowerPoint?") or title("File Man?") or
title("Microsoft Wor?") or title("Sol?");
```

This MATCH statement causes displays or applications with the following characteristics to be acceptable to a window:

- any Category, or
- titles beginning with “Mine”, “File Man”, “Microsoft Wor”, or “Sol”, or
- titles containing “PowerPoint”,
- the title “FreeCell”

Example 2: Any title

```
MATCH = title("?*");
```

This MATCH statement causes all displays or applications that have the “title” property to be acceptable to a window:

Example 3: Any category, title, or module (“catch-all”)

```
MATCH = category("?*") or title("?*") or module("?*");
```

This MATCH statement causes any Category, or any Title, or any Module to be acceptable.



Tip

This statement is suitable for a catch-all window. If used, it should be included in the last window definition of the `.wdl` file, since the statement “catches” everything, and may not be suitable for the other windows.

Example 4: Categories and titles

```
MATCH = category("schematic");
MATCH = category("trend");
MATCH = category("alarm?*");
MATCH = category("faceplate") or
title("?*Microsoft word?") or
title("?*Microsoft Excel?");
title("?*notepad?*");
MATCH = category("controlpanel");
MATCH = category("schematic") or category("nativewindow");
```

These MATCH statements cause displays or applications with the following characteristics to be acceptable to a window:

- with the Category: “schematic”
- with the Category “trend”
- with the Category “alarm”
- with the Category “faceplate”, or with Titles containing “Microsoft Word”, “Microsoft Excel”, or “notepad”
- with the Category “controlpanel”
- with the Category “schematic” or “nativewindow”

Example 5: Any category except specified category

```
match = category("?*") and not category("alarm?*");
```

This MATCH statement causes displays or applications with any Category, except a category beginning with “alarm”, to be acceptable to the window:

Example 6: Modules

```
match = module ("?*lcwindow?*");
match = module ("?*excel?*") or ("?*word?*");
```

This match statement causes displays or applications with the following characteristics to be acceptable to a window:

- a module containing “lcwindow”
- a module containing “excel ” or “word”

Example 7: Categories, titles, and modules

```
match = category("schem?*") or title("FreeC?*") or title("File Man?*") or title("Sol?*") or
module("?*notepad.exe");
match = category("tren?*") or title("Mine?*");
match = category("alar?*") or title("Sol?*");
```

This MATCH statement causes displays or applications with the following characteristics to be acceptable to a window:

- a Category beginning with: “schem”, or a Title beginning with “FreeC”, “File man”, or “Sol”, or with a module ending with “notepad.exe”
- a Category beginning with “tren”, or a Title beginning with “Mine”
- a Category beginning with “alar”, or a Title beginning with “Sol”

Example 8: Titles and modules

```
match = title("Acrobat Reader");
match = title("LCN Overview");
match = title("?*awafp?*");
match = title("?*tbl?*");
match = title ("?*Microsoft PowerPoint?*");
match = title("?*pct");
match = module ("?*lcwindow?*");
match = module ("?*word?*");
match = module ("?*gpb?*");
```

This MATCH statement causes displays or applications with the following characteristics to be acceptable to a window:

- a Title of “Acrobat Reader”
- a Title of “LCN Overview”
- a Title containing “awafp”

- a Title containing “ Microsoft PowerPoint ”
- a Title ending with “pct”
- a Module containing “lcnwindow”
- a Module containing “word”
- a Module containing “gpb”

38.8 Expressions syntax and semantics

This section presents the syntax and semantics of expressions. The expression syntax of WDL is a subset of the expression syntax found in the “C” programming language. Generally, expressions can be used where numeric and string values are required.

38.8.1 Expressions Syntax

| | |
|-----------------------------|--|
| <i>boolean_expression:</i> | <i>constant_expression</i> |
| <i>string_expression:</i> | <i>constant_expression</i> |
| <i>constant_expression:</i> | <i>Expression</i> |
| <i>expression:</i> | <i>expression OR and_expr and_expr</i> |
| <i>and_expr:</i> | <i>and_list</i> |
| <i>and_list:</i> | <i>and_list AND binary binary</i> |
| <i>binary:</i> | <i>binary '!=' binary binary '>' binary binary '>=' binary binary '<' binary binary '<=' binary binary '==' binary binary '*' binary binary '/' binary binary '+' binary binary '-' binary unary</i> |
| <i>unary:</i> | <i>(' expression ')</i> <i>number</i> <i>constant</i> <i>string</i> <i>'-' unary</i> <i>NOT unary</i> |

38.8.2 Expressions Semantics

Logical and arithmetic expressions

Logical and arithmetic expressions in WDL are a subset of expressions in the “C” language. Refer to a “C” language reference book for information on the usage of expressions. The support for expressions in WDL is different from “C” language expressions in the following ways:

- String type is a base type in WDL. Strings are not treated as arrays of characters, as they are in the “C” language. It is an error to mix string and number types in expressions;
- The expression operators do not generally support string type values.

boolean_expression

A Boolean expression is a numeric expression that is interpreted as *true* or *false*. If the expression evaluates to a zero value (0.0), it is interpreted as *false*. If it evaluates to a nonzero value, it is interpreted as *true*. Using predefined constants is suggested. Here is a list of the constants you can use to define Boolean positive and negative values correctly.

| Positive Value | Negative Value |
|----------------|----------------|
| <i>TRUE</i> | <i>FALSE</i> |
| <i>True</i> | <i>False</i> |
| <i>true</i> | <i>false</i> |
| <i>YES</i> | <i>NO</i> |

| Positive Value | Negative Value |
|----------------|----------------|
| <i>Yes</i> | <i>No</i> |
| <i>yes</i> | <i>no</i> |
| 1 | 0 |

string_expression

A string expression is one that evaluates to a string value. String expressions can consist only of a string literal or a string constant.

constant_expression

A constant expression is one that evaluates to a constant number or string value.

38.9 Regular expression pattern syntax

This section contains a description of the rules for building regular expression patterns.

38.9.1 Regular expressions in MATCH operators

MATCH operators are specified for category, title, and module MATCH expressions. During the search for a window specification to be used to control a newly created real window, the strings representing the actual category, title, or module of a real window are compared to the regular expression pattern in the operand of the MATCH operator, and a *true* or *false* is returned. If *true* is returned, the MATCH operator operands are used to control the window. If *false* is returned, the strings representing the actual category, title, and module of a real window are used. Using regular expressions enables you to use the “wildcard” feature to avoid having to manually code every string.

38.9.2 Syntax and semantics of regular expressions

- A regular expression pattern consists of these elements:

| | |
|--|--|
| c | Literal character |
| ? | Any character except newline |
| ^ | Beginning of line |
| \$ | End of line |
| [...] | Character class (any one of these characters) |
| [~...] | Negated character class (all but these characters) |
| * | Closure (zero or more occurrences of previous pattern) |
| Note: Typically, a MATCH expression will probably use only “?” “*” or “c.” | |

- Special meaning of characters in a text pattern is lost when escaped, inside [...], or for:

| | |
|----|------------------|
| ^ | Not at beginning |
| \$ | Not at end |
| * | At beginning |

- Special meaning of characters in a character class is lost when escaped or for:

| | |
|---|---------------------|
| ~ | Not at beginning |
| - | At beginning or end |

- Character class consists of zero or more of these elements, surrounded by [and]:

| | |
|------------|---|
| c | Literal character, including [|
| a-c | Range of characters (digits, upper or lower case) |
| ~ | Negated character class if at beginning |

Examples

- “?*WORD?*” - matches any string with the pattern “WORD” somewhere in the string.
- “Micro?*” - matched any string beginning with “Micro”

38.10 Predefined constants

The workspace provides several predefined constants for improving readability of workspace definitions through the consistent use of names for common values. The predefined constants, their type, and values are listed in the following table.

| Name | Type | Value |
|--------------|--------|-------|
| <i>YES</i> | Number | 1.0 |
| <i>NO</i> | Number | 0.0 |
| <i>Yes</i> | Number | 1.0 |
| <i>No</i> | Number | 0.0 |
| <i>yes</i> | Number | 1.0 |
| <i>no</i> | Number | 0.0 |
| <i>TRUE</i> | Number | 1.0 |
| <i>FALSE</i> | Number | 0.0 |
| <i>True</i> | Number | 1.0 |
| <i>False</i> | Number | 0.0 |
| <i>true</i> | Number | 1.0 |
| <i>false</i> | Number | 0.0 |

38.11 Miscellaneous

This section contains descriptions of language items that are common to the workspace, group or window specifications.

38.11.1 Miscellaneous Syntax

| | |
|--------------------|---|
| <i>windowbox:</i> | <i>expression, expression, expression, expression</i> |
| <i>size:</i> | <i>expression, expression</i> |
| <i>point:</i> | <i>expression, expression</i> |
| <i>identifier:</i> | See ““Lexical conventions” on page 220” |
| <i>string:</i> | See ““Lexical conventions” on page 220” |
| <i>constant:</i> | <i>identifier</i> |

38.11.2 Miscellaneous Semantics

windowbox

A window box specifies the position and size of a window. It is specified with four numbers. The first two numbers give the x and y coordinates of the upper-left corner. The third number gives the width and the fourth number gives the height. An example usage is the value of the POSITION attribute in a window specification. The values can be arithmetic expressions.

size

A size specifies the height and width of a rectangle. An example usage is the value of the MAXSIZE attribute in a window specification. The two values are numbers and can be arithmetic expressions.

point

A point specifies a position in terms of its x and y coordinates. The two values can be expressions. An example usage is the ICON POSITION attribute of the window specification.

string

A string is either a string literal or an identifier representing a string constant.

identifier

Refer to ““Lexical conventions” on page 220” for a definition of this item.

constant

An identifier that has been declared in a *constant_declaration*. A constant represents the value assigned in its declaration. The type of a constant is either string or number, depending on the form of the constant declaration.

39 Appendix B: Workspace client Visual C++ code

Related topics

“Visual C++ sample” on page 240

39.1 Visual C++ sample

```
// This code is Visual C++ code, using the Microsoft Foundation Class Library.
// mainfrm.cpp : implementation of the CMainFrame class
//
#include "stdafx.h"
#include "pclient.h"
#include "command.h" // this is a Honeywell proprietary command line parser, not provided here
#include "mainfrm.h"
#include "winprop.h"
#include "wsmcli.h"
#include "HsiResult.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////
// CMainFrame
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
//{{AFX_MSG_MAP(CMainFrame)
ON_WM_CREATE()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////
// CMainFrame construction/destruction
CMainFrame::CMainFrame()
{
// TODO: add member initialization code here
}
CMainFrame::~CMainFrame()
{
}
////////////////////////////////////
// CMainFrame message handlers
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
return -1;
HSI_RESULT hResult;
BOOL bCategorySpecified = FALSE;
CString sCategory;
CString sPromptText = _T("");
CString sCommandLine;
CString sWorkingDir = _T("v:\\display\\workspaces\\test\\pclient\\bitmaps");
CCmdLineOptions cmdLineOptions;
// Get command line
sCommandLine = ::GetCommandLine();
// Get category if specified, from command line
if (cmdLineOptions.GetKeywordOptionValue(_T("w"), sCategory))
{
bCategorySpecified = TRUE;
}
// Get document if specified and cut off the path name, // leaving just the file name
if (!cmdLineOptions.GetOptionValue(sPromptText)) // get document into sPromptText
sPromptText = _T("PClient - Untitled");
else
{
CString sTemp = sPromptText;
// find last '\\' then set sPromptText to only chars // after the '\\'
int i = sTemp.ReverseFind('\\');
i = sTemp.GetLength() - (i+1);
sPromptText = sTemp.Right(i);
}
// /d option specifies use of WSRegisterWindowData API // call instead of older
WSRegisterWindowCategory call
if (!cmdLineOptions.FlagOptionsSpecified(_T("nd")))
{
if (bCategorySpecified)
{
hResult = ::WSRegisterWindowData((LPCTSTR)sCategory,
(LPCTSTR)sPromptText,
(LPCTSTR)sCommandLine, (LPCTSTR)sWorkingDir, m_hwnd );
}
else // no category specified, so use "testcategory"
{
hResult = ::WSRegisterWindowData((LPCTSTR)" testcategory",
(LPCTSTR)sPromptText, (LPCTSTR)sCommandLine
```



```

(LPCTSTR)sWorkingDir, m_hwnd );
}
}
else // use old register-category API
{
    if (bCategorySpecified)
        hResult = ::WSRegisterWindowCategory((LPCTSTR)sCategory, m_hwnd );
    else
        hResult = ::WSRegisterWindowCategory((LPCTSTR)"testcategory", m_hwnd );
}
// log error if required
BOOL debug = TRUE;
if (HR_IS_FAILED(hResult, debug))
    // Issue appropriate error message.
    ;
return 0;
}

```


40 Appendix C: SafeView error messages

Related topics

“Error messages list” on page 244

40.1 Error messages list

| Error message | Explanation | Suggestion |
|---|--|--|
| A window specification may contain only one <i><attribute name></i> attribute. | More than one instance of the given attribute was specified in a given window specification. Window specification attributes (properties) include: POSITION, SIZE, MINSIZE, MAXSIZE, REGION, DRAGGABLE, ALWAYS ON TOP, PLACEHOLDER, and MAXPOS. | Limit the use of a given attribute to once per window specification. |
| The POSITION attribute value must be within the specified region. | The configured POSITION point (window's upper left corner) is not within the region area. | Modify the POSITION value so that it falls within the specified region. |
| The MAXPOS attribute value must be within the specified region. | The configured MAXPOS point (window's upper left corner when maximized) is not within the region area. | Modify the MAXPOS value so that it falls within the specified region. |
| The MAXSIZE attribute value must be within the specified region. | The configured MAXSIZE (maximum size of the window) is larger than the size of the region. | Modify the MAXSIZE value so that it is not larger than the area of the specified region. |
| The SIZE attribute must be smaller than or equal to the MAXSIZE attribute. | The configured SIZE (window's initial width and height) is larger than the configured MAXSIZE property. | Modify the SIZE or MAXSIZE value so that SIZE does not exceed MAXSIZE. |
| The SIZE attribute must be greater than the MINSIZE attribute. | The window's configured SIZE (initial width and height) is smaller than the configured MINSIZE property. | Modify the SIZE or MINSIZE value such that SIZE is not less than MINSIZE. |
| Maximum expression size exceeded. | Too many components have been included in a MATCH expression. The limit, for performance reasons, is 100. | Limit the and'd / or'd, etc., components of a MATCH expression to less than 100 items. |
| Type mismatch between operands of comparison operator. | A logical in the configuration file contains types which are not comparable. | Make the components of the offending expression type-compatible. |
| Invalid operands for comparison operators | The operands provided for the given expression are not appropriate for the comparison operation. | Modify the operands to the expression. |
| Type mismatch between operands of arithmetic operator. | The given expression has type-incompatible operands | Modify the operands to the expression. |
| Invalid type for arithmetic operator. | The given expression has a type-incompatible operand. | Modify the operand to the expression. |
| Expression evaluation Stack overflow. | The expression has too many components. It is too complex for the SafeView expression evaluator. | Simplify the expression. |
| Division by zero. | A division by zero error has been detected. | Correct the arithmetic expression so that the denominator is not zero. |
| SafeView workspace configuration properties must appear before the workspace group. | Workspace level properties (startup list, constants) were detected within a workspace group; that is, they were not at the "workspace level" | Declare startup list construct and workspace constants before declaring the top-level workspace group. |
| <i><X, Y, width or height></i> value of windowbox item must be greater than or equal to zero. | A rectangular area, such as REGION, has been specified using a negative value for one of parameter values. | Do not configure a window or region with negative numbers. |

| Error message | Explanation | Suggestion |
|--|--|---|
| <height, width> value of windowbox item must be greater than or equal to zero. | A negative value has been provided for width or height. | Use a non-negative value. |
| <height, width> value of SIZE item must be greater than or equal to zero. | A negative value has been provided for a SIZE value. | Use a non-negative value. |
| <X, Y> value of point item must be greater than or equal to zero. | A negative value has been supplied for a point. | Use a non-negative value. |
| Expecting a numeric expression. | A non-numeric or non-logical expression was found where a numeric expression (including logical) was expected. | Values such as <i>true/false</i> , etc., are considered “numeric” but strings are not. Provide a valid numeric or logical value. |
| Expecting a string expression. | A string-valued expression is expected. | Provide a string type value. For example, a string is expected as the group type. |
| Invalid type for operand of <OR, AND>. | Invalid operand, likely in the MATCH expression. | Check syntax for MATCH expression. |
| Invalid type for operand of unary minus operator. | Invalid type for operand of unary minus operator found. | Provide a numeric operand. |
| Invalid type for operand of logical negation operator. | Invalid type for operand of unary “not” operator found. | Provide logical operand. |
| String termination in escape sequence. | An escaped character sequence contained a string termination. | Check syntax and correct. |
| Invalid character in escape sequence. | An invalid character has been found in an escape sequence. | Check syntax and correct. |
| Missing comment terminator. | A comment, beginning with “/*” is missing the terminator “*/” in a text-edited .wdl file. | Terminate the comment correctly. |
| String is missing the closing quote. | String is missing the closing quote. | Add the closing quote. |
| Non-digit after sign. | Non-digit found after a + or -sign. | Follow a sign operator with a numeric digit. |
| Non-digit after decimal. | Non-digit found after a decimal. | Follow a decimal point with a numeric digit. |
| Invalid character (hex) <value>. | Invalid character found, displayed in hex format. | Replace or remove the invalid character. Check syntax. |
| A SafeView workspace configuration can only have one top level window group. | More than one group is defined at the “workspace level.” | Place all subgroups within a single, top-level workspace group. |
| Invalid character(s) found in SafeView workspace configuration file. | Invalid character(s) found in SafeView workspace configuration file. | Check syntax and correct. |
| Invalid SafeView configuration file. | Invalid SafeView configuration file. | Check syntax and correct. |
| <string value> is an invalid group behavior mode. | An invalid group behavior found. | Legal values include “round robin,” “manual select,” and “first match.” |
| More than one window specification within a group with the same name is not allowed. | More than one window specification within a group with the same name found. | Keep window spec names unique within a group. If using window spec names in API calls, you should keep them unique across the entire workspace configuration. |

| Error message | Explanation | Suggestion |
|--|---|--|
| SafeView configuration load failure. Do you wish to open another SafeView configuration file? | A problem was detected while loading a given file. | Check the syntax on the file in the SafeView editor. |
| Error: file not found. | Startup list contained a file that was not found. | Provide correct file and path name. |
| Process creation error occurred. | SafeView could not start the given process or application. | Verify that the application could be started using similar parameters given directly to the operating system (for example, “run” command). Perhaps the system could not find the file in the current path, or the file does not exist. |
| SafeView workspace group names must be unique. | The same name was used for more than one workspace. | Make group names unique across a configuration file. |
| Only first-match groups can have subgroups. | A round robin or manual select group was found with a subgroup. | Only provide subgroups within first match group. |
| Configured MAXSIZE dimensions must be greater than MINSIZE dimensions. | The value of MAXSIZE is less than the value of MINSIZE. | Make MAXSIZE larger than MINSIZE. |
| Only one DEFAULT item allowed within a workspace. | More than one DEFAULT keyword was discovered in a configuration file. | Make only one DEFAULT property in a configuration file. (See help on “default.”) |
| Match expression not allowed in window specifications contained in groups of this type. | First match group found with MATCH expression. | In first match group, MATCH expressions belong to the group's component subgroups and windows. |
| Group properties must appear before the any window specifications or nested groups. | Group properties were included after either a window specification or nested group. | Place group properties (match, region) before the subgroups and window specifications. |
| DEFAULT attribute not allowed on groups of this type. | DEFAULT should always be paired with a MATCH expression. | Check syntax and correct. |
| DEFAULT attribute not allowed in window specifications contained in groups of this type. | DEFAULT should always be paired with a MATCH expression. | Check syntax and correct. |
| No window or group identified as DEFAULT match in this workspace. | DEFAULT should always be paired with a MATCH expression. | Check syntax and correct. |
| Match expression not allowed in window specifications contained in groups of this type. | Only windows in first match groups may have their own MATCH expressions. | For window specifications in round robin or manual select groups, always place a common MATCH expression at the group level. |
| No window or group identified as DEFAULT match in this workspace. | Warning only. It is a good idea to identify one MATCH expression as the default, so unknown categories can be managed. | If desired, provide a DEFAULT MATCH expression. |
| SafeView is already executing. | More than one copy of SafeView is running at the same time. | Close or use existing copy of SafeView. |
| An older version of WSMRES.DLL was found. Some strings may not be displayable. Do you wish to continue anyway? | SafeView found and loaded a resource-only DLL (WSMRES) which is not as new as the built-in version of the resources. You may proceed, but know that not all resources are necessarily there or up-to-date. The program MAY CRASH. | Remove or rename the old WSMRES.DLL and use the built-in (English) version. Acquire an up-to-date version of the resource-only DLL if a non-English version of SafeView is required. |

| Error message | Explanation | Suggestion |
|---|---|--|
| More than one MATCH expression on this group or window specification is not allowed. | More than one MATCH expression was found in this group or window specification. | Include only one MATCH expression per group or window specification. |
| A window specification may contain only one SIZEABLE attribute. | More than one SIZEABLE attribute found in a given window specification. | Limit SIZEABLE to once per window specification. |
| Given the current window POSITION, its SIZE does not allow it to fit into the specified region. | A given POSITION and SIZE does not allow the window to fit into the given region. | Modify any of these attributes so that the window fits into the region. |
| A window SIZE will not fit into its specified region. | A window is sized so that it will not fit into its specified region. | Modify SIZE so that window fits into region. |
| The invocation of the historized display was not successful. | An invocation of a prior display (perhaps from the SafeView History Dialog) failed. | Verify that the parameters with which the application was registered with SafeView (using CreateWorkspaceClient, or built-in to the application) are valid. Check syntax and correct. |
| The only valid values for MAXWINDOWS are 1 and "infinite." | An invalid value for MAXWINDOWS was found. | Set to "1" or "infinite". This value specifies whether a first match window specification can be simultaneously used by "infinite" number of displays, or only by one display at a time. |
| The MAXWINDOWS property is only valid for first match groups. | The MAXWINDOWS property was included in a window specification that is not in a first match group. | Only place MAXWINDOWS in window specifications that are in a first match window group. |
| PLACEHOLDERS = <i>false</i> not allowed for windows in Manual Select groups. | PLACEHOLDERS = <i>false</i> statement was detected in a manual select group. | Manual select groups must have placeholders, otherwise, the operator cannot select a window for use. |
| A window's configured <MINSIZE, MAXSIZE, SIZE> property is too narrow. The Minimum window width for round robin windows is 120. | Windows cannot be configured narrower than 120; otherwise SafeView buttons begin to cover the left-side icon in the title bar of a display. | Configure this value greater than 120 pixels. |
| A window's configured <MINSIZE, MAXSIZE, SIZE> property is too short. The Minimum window height for all windows is 26. | Windows cannot be configured shorter than 26; otherwise SafeView buttons cannot be displayed. | Configure with value greater than 26 pixels. |
| A window specification may contain only one CLOSABLE attribute. | More than one CLOSABLE attribute was configured for a given window specification. | Limit to one CLOSABLE attribute per window specification. |
| A window specification may contain only one GLOBALFOCUS attribute. | More than one GLOBALFOCUS attribute was configured for a given window specification. | Limit to one GLOBALFOCUS attribute per window specification. |
| Invalid expression. | | Correct the expression. Verify that constants are correctly spelled. |
| Invalid expression found (see highlighted field). | | Correct the expression that is highlighted in the dialog box. |
| Redefinition of a built-in constants not allowed. | The constant is a system-defined built-in constant that cannot be user-redefined. | Choose another constant name. |
| Name cannot include embedded blanks. | The name string contains one or more blanks. | Remove embedded blanks from the name string. |

| Error message | Explanation | Suggestion |
|---|--|--|
| Invalid MATCH expression. | An invalid MATCH expression, such as including the “match = ” found in TextEdit files, in the dialog edit box for MATCH expressions. | Correct the MATCH expression. For example, title(“?*Micro?*”) and category(“?*schems?*”) are good entries. |
| Only one default MATCH expression allowed within a workspace. | More than one MATCH expression is configured for this workspace. | Do not select the Default Match Expression check box if it is already selected for another window or group. |
| Can't start more than one properties Dialog. | This occurs when double-clicking a placeholder window to bring up its properties dialog. | Close the existing properties dialog before attempting to open another one. |
| One or more placeholders have not been updated. Continue? | | If you want to save the changes to placeholder position, update the placeholder data and save the workspace before proceeding. |
| Error in window: <window name> -> <nature of the error> Open it's property sheet? | | Open the property sheet and correct the error. |
| Warning: there is no default MATCH expression specified in this workspace (used to manage displays registered with\n unknown categories). | This occurs with new displays with new category values that are unknown when the <i>SafeView.wdl</i> file is created. | Consider whether you want to have a default window specification (or group). The default would be used to match on category-registering applications that register a category value that does not successfully match anywhere else in the workspace definition file. |
| SafeView TextEdit program not found. | The TextEdit and SafeView GWE programs are not in the same directory. | Put the TextEdit program (textedit.exe) and the SafeView GWE program (svedit.exe) in the same directory. |
| The workspace itself cannot be deleted by the editor. | | Do not attempt to delete the workspace (tree) item itself. |
| Cannot delete a non-empty window group. | This version of SafeView GWE does not allow cut/paste of groups with items (that is, component windows). | First cut the items, then cut the group. To “move” a group and its components, you must (1) cut and paste the items to a temporary location, (2) cut and paste the group to its desired location, then (3) cut and paste the components back into the group. |
| Cannot delete the top level workspace window group. | | Do not attempt to delete this top level workspace (tree) item itself. |
| Can't paste item at the workspace level. | | Paste the item under the main (top-level) group or under one of its components. |
| SafeView window configured SIZE property is too narrow. Minimum window width for round robin windows is 120. | The SIZE value you have entered is too small. | Set SIZE equal to 120 or greater.’ |

41 Appendix D: SafeView tips and tricks

Related topics

- “Populating a workspace automatically on startup” on page 250
- “Normal search mode vs. focus-based search mode” on page 251
- “Supporting multiple command windows in Station” on page 253
- “Supporting additional displays in multi-window Station” on page 256
- “Station's configured startup page vs. Station startup commands” on page 257
- “SafeView's graphical editor vs. text-based editor” on page 258
- “The focus-based, programmatic, multi-display launch problem” on page 259
- “SafeView's command line options” on page 261
- “SafeView's facilitation of Station client tracing” on page 262
- “Two MATCH-expression options for managing Experion faceplates” on page 263
- “Inclusive/exclusive 'everything except' MATCH expressions” on page 264
- “Yoking faceplates to main displays” on page 265
- “Enhancing consistency and safety with region constraints” on page 266
- “Closing faceplates automatically on main display close” on page 267
- “Managing cross-screen, focus-based displays” on page 269
- “Supporting multiple catch-all windows” on page 270
- “Using wildcard character-matching” on page 271
- “Constant-based vs. literal-based coordinates” on page 272

41.1 Populating a workspace automatically on startup

A convenient feature in SafeView is the capability to automatically populate a workspace upon startup. When loading a workspace, you can launch Experion Station and/or any number of other applications. If you typically use only one workspace and only launch SafeView on restart and/or login, it is best to launch Station and/or other primary displays and/or applications directly from SafeView.

When specifying path names in the workspace file directly, be sure to include a double backslash (\\) for each backslash (\) in the path name. For example, the following startup commands launch Station, two Station displays, a Native Window, and a GUS display:

```
//User defined application invocations
startup
"c:\\program files\\honeywell\\experion pks\\client\\station\\ station.exe \"c:\\program files\\
\\honeywell\\experion pks\\ client\\station\\mystation.stn\" -T c:\\temp\\stationtrace.txt /windows
sysAlarmSummary sysEventSummary";
end startup
"lcnwindow.exe";
"runpic gpbdisplay1.pct"
end startup
```

The startup commands include an example for launching Station, specifying (optionally) a particular station (".stn") file to use, and also (optionally) specifying several displays to launch at startup. Note that to support an embedded string in the parameter, quotation marks (") are used to specify "mystation.stn." Note also that "mystation.stn" is intended to represent a customized version of "default.stn" in the same directory. By default, there is no actual "mystation.stn" file.

41.2 Normal search mode vs. focus-based search mode

SafeView provides three primary types of window groupings: **first match**, **manual select**, and **round-robin**. Each group may contain windows, though only first match groups may contain nested window groups.



Tip

You can configure normal and focus-based workspaces to explicitly always position specific applications and/or displays in a specific screen location. Thus (for example), even if you want to manage custom displays in a focus-based manner, you can always have the Alarm Summary displayed in the lower left screen and the Native Window displayed in the upper right screen.

41.2.1 Normal search mode

In normal search mode, SafeView searches the workspace windows in a “top-down” manner to find a matching window. A match is determined when a match expression (in the window specification of the workspace) returns *true* for a newly-invoked display. To determine a match, SafeView uses these criteria:

- Window title
- File path of the process that created and owns the window
- Pre-registered category string associated with that window's handle according to the SafeView API call



Tip

When matching by title, it is important to know that SafeView uses the title as available at the instant the display is to be shown. If an application modifies an application's window title after the display has been shown, SafeView will not have the benefit of the new window title when it processes the display.

41.2.2 Focus-based search mode

In focus-based search mode, when searching for a matching window specification, SafeView's logic is to “search the active window's owning group, then that group's owning group, and so on, until the entire workspace is searched.” This means that (for example) if a window on the left screen is in a SafeView first match group that also includes a faceplate or faceplate group, SafeView will search for a match in that first match group along with its faceplate or faceplate group. If SafeView cannot find a match there, it will search elsewhere. If a match is not found before the required outer group is searched, SafeView essentially reverts to the normal search mode.

Focus-based search mode offers greater flexibility in managing multi-screen displays in that identical (or similar) displays can be easily and intuitively placed at various screen locations based on your current area of focus. This means that (for example) instead of forcing the alarm summary or a particular set of custom graphics on a particular screen, you can display them on any physical screen based on your current area of focus (the currently active window). Put simply, the general behavior for focus-based workspaces is “the next display goes near or where the currently active display is.”



Tip

When you click in a window, the Windows operating system instantly activates that window. Thus, clicking a hyperlink or button on a display activates that display, which then becomes the “center of focus” as SafeView determines where to place the newly-invoked display. Consequently, when looking at an active display (with title bar highlighted) on the left screen, and then clicking a button on the right screen to invoke a new display, SafeView dispositions the newly-invoked display according to the right screen being “in focus” because it was the last active display at the time of invocation. Thus, the new display will (typically) go to the right screen and not the left screen.

With focus-based workspaces, you must understand the typical mechanism for invoking main displays into a given screen. One simple, consistent method is to leverage a focus-based workspace with multiple Station command windows, where displays are invoked from a given command window to that same screen. This method is particularly useful when all main screens are intended to handle the same set of displays (such as all

non-trend, non-faceplate displays). To launch displays, you can use a custom navigation window instead of the standard Station command window.

If using fewer Station command windows (or custom navigation displays) than physical screens, you must consider how operators will invoke main displays to a given screen. In a focus-based workspace, there are two general mechanisms for invoking main displays to screens without a navigation or command window. They are:

- To manage specific types of displays only on a given screen, and/or
- To utilize the keyboard (IKB/OEP, or PC keyboard) to launch displays. With this, the operator replaces an active main display by clicking a button that launches a display (without changing window focus in the process).

41.3 Supporting multiple command windows in Station

The Station multi-window mode supports multiple command windows, such as the one that follows:

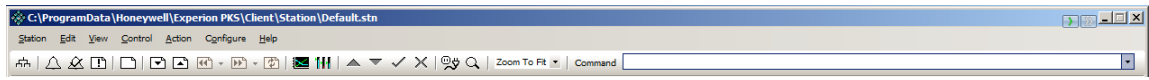


Figure 18: Typical Station Command Window

Multiple command windows work well in a multi-screen environment in conjunction with SafeView's focus-based search mode. Simply park a command window on each of the lower screens (or even on all four screens for an ICON node). Then, any display you invoke through a given command window can show on that same screen.

One benefit of using multiple command windows is that errors and problems are more noticeable because they are displayed in the message zone of each command window. When only one command window is available, for example, such messages may not be noticeable on the left screen when the operator is working on the right screen. For more details on command windows, see Experion documentation.

Another benefit of using multiple command windows is their ability to control focus-based display invocation from each respective command window, assigning the invoked display to the screen for that command window. Additionally, main display sizes are more consistent between screens because they have more consistent aspect ratios.

Since you can configure a command window to customize (or hide) menu and toolbar elements, consider using it as a standard navigation bar. You can also use the command window to communicate with the operator through its message/interaction zone. You can use the standard Windows **Display Properties** page to set which tooltip color to assign to the Station message zone. To do so, choose **Start > Control Panel**. Then select **Appearance and Personalization**. Then select **Display**. Change the appearance of your displays. Select **Personalization** to change the theme, color and other options. Then choose **Appearance > Advanced**. On the resulting **Advanced Appearance** page shown in the following figure, make your appropriate selections from the **Item** and **Color** options. The figure also shows the resulting Command windows.

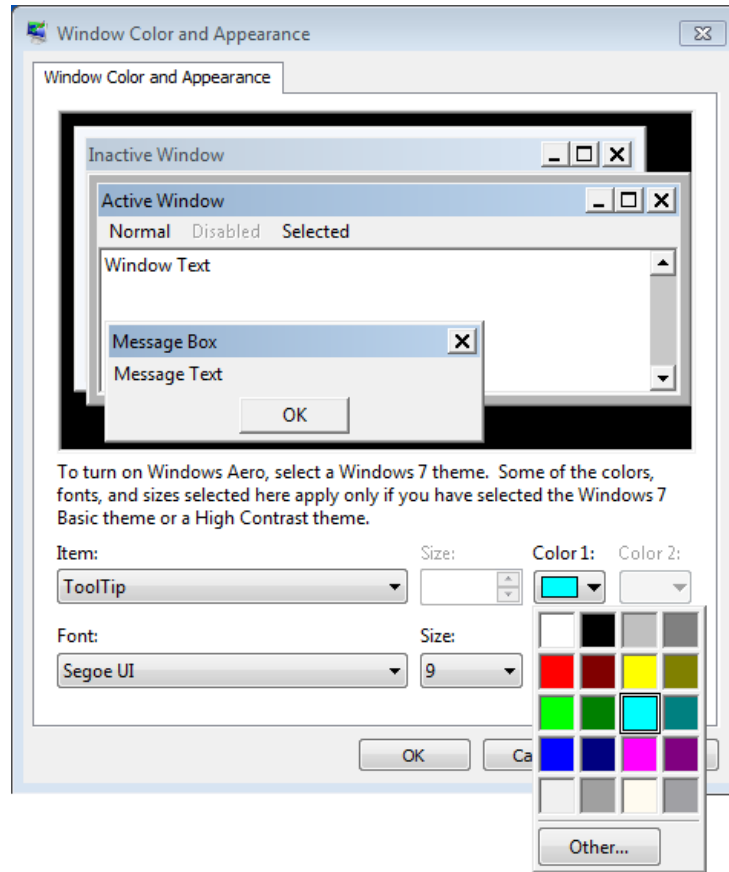
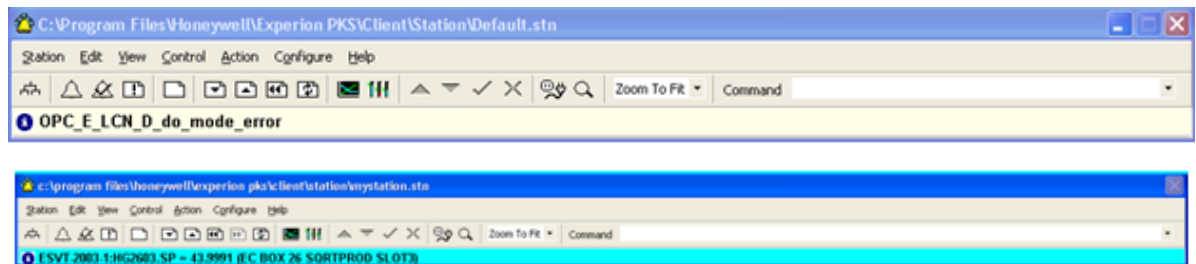
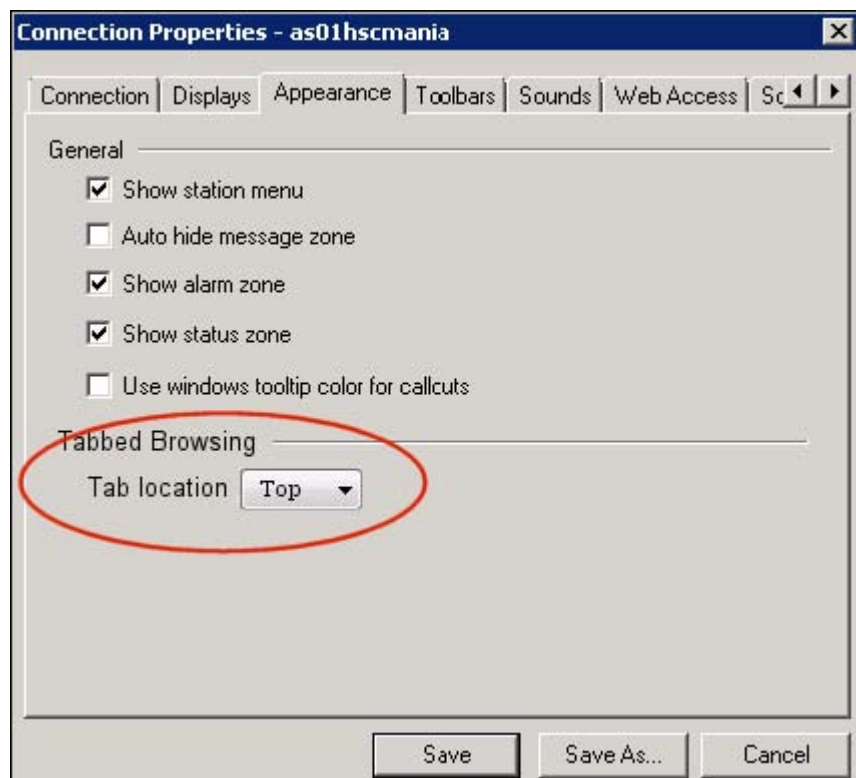
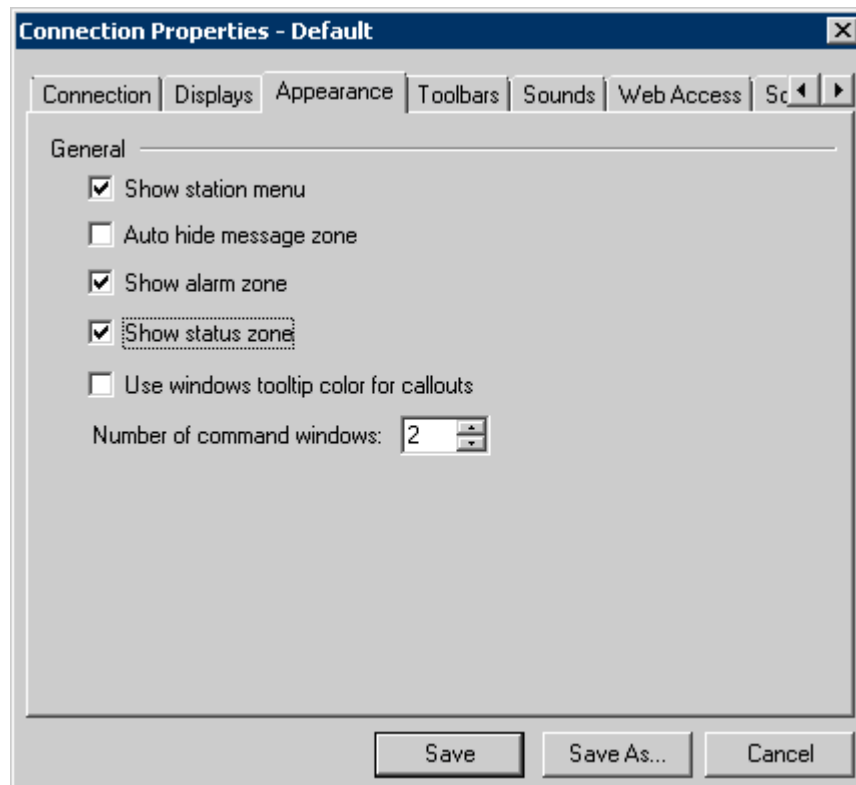


Figure 19: Display Properties Settings and Resulting Station Command Windows



41.3.1 Configuring Station for multiple command windows

- 1 In Experion Station, open the **Connection Properties** dialog box.
- 2 Click the **Appearance** tab to open the following page:



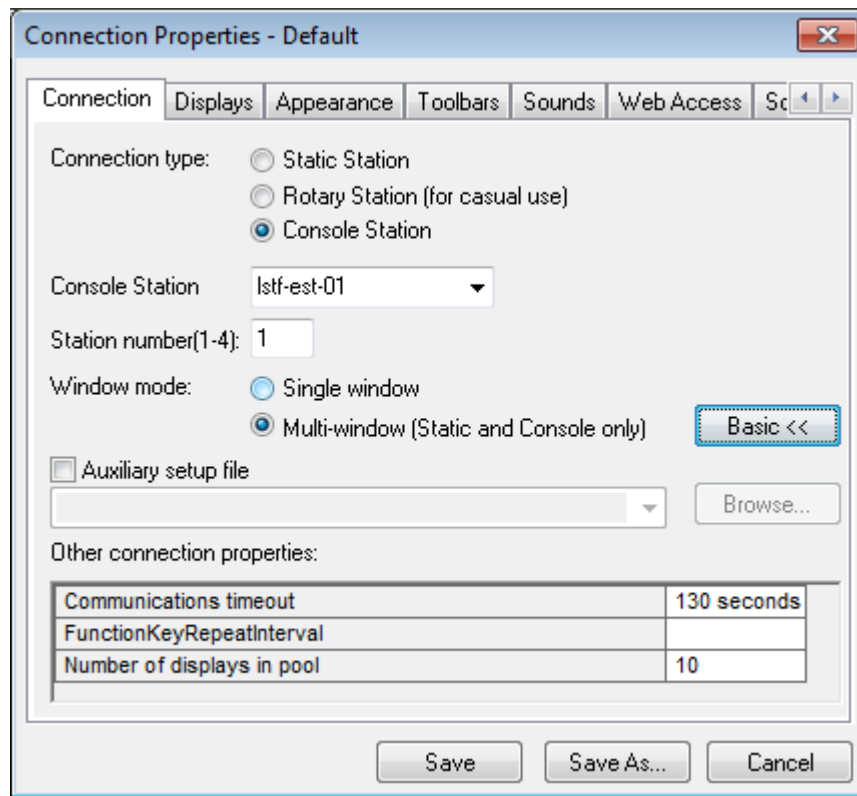
- 3 In the **Number of command windows** spin box, type or select the appropriate number.
- 4 Click **Save** to implement the change.
- 5 Close the **Connection Properties** dialog box.

41.4 Supporting additional displays in multi-window Station

Several “Safeview workspace examples” on page 153 given herein can support up to twelve trends and faceplates. Since these typically use fewer resources than dense custom graphics, you may find that supporting this many displays simultaneously does not excessively burden your system. However, to support this many displays (along with main and catch-all displays), you must adjust the number of Station processes available simultaneously; otherwise, Station will display an error when you try to invoke more than the available number of displays. To make this adjustment, do the following procedure.

41.4.1 Configuring Station to support additional displays

- 1 In Experion Station, open the **Connection Properties** dialog box.
- 2 Click the **Connection** tab to open the following page:



- 3 In the **Number of displays in pool** text box, type the appropriate number.



Tip

For the extreme workspace (twelve trends, twelve faceplates, four main displays), you will need twenty-eight processes, but always add one or two extra (as a process must be used to invoke a new display before SafeView will close/replace an existing display, releasing its process back to the pool).

- 4 Click **Save** to implement the change.
- 5 Close the **Connection Properties** dialog box.

41.5 Station's configured startup page vs. Station startup commands

When using Station in single-window mode, SafeView is not required. However, using SafeView may be suitable in single-window mode when also using other applications. In single-window mode, Station launches the display in the server-wide settings page as shown in the following figure. In multi-window mode, you should use SafeView's startup commands to automatically launch displays at Station startup as discussed here.

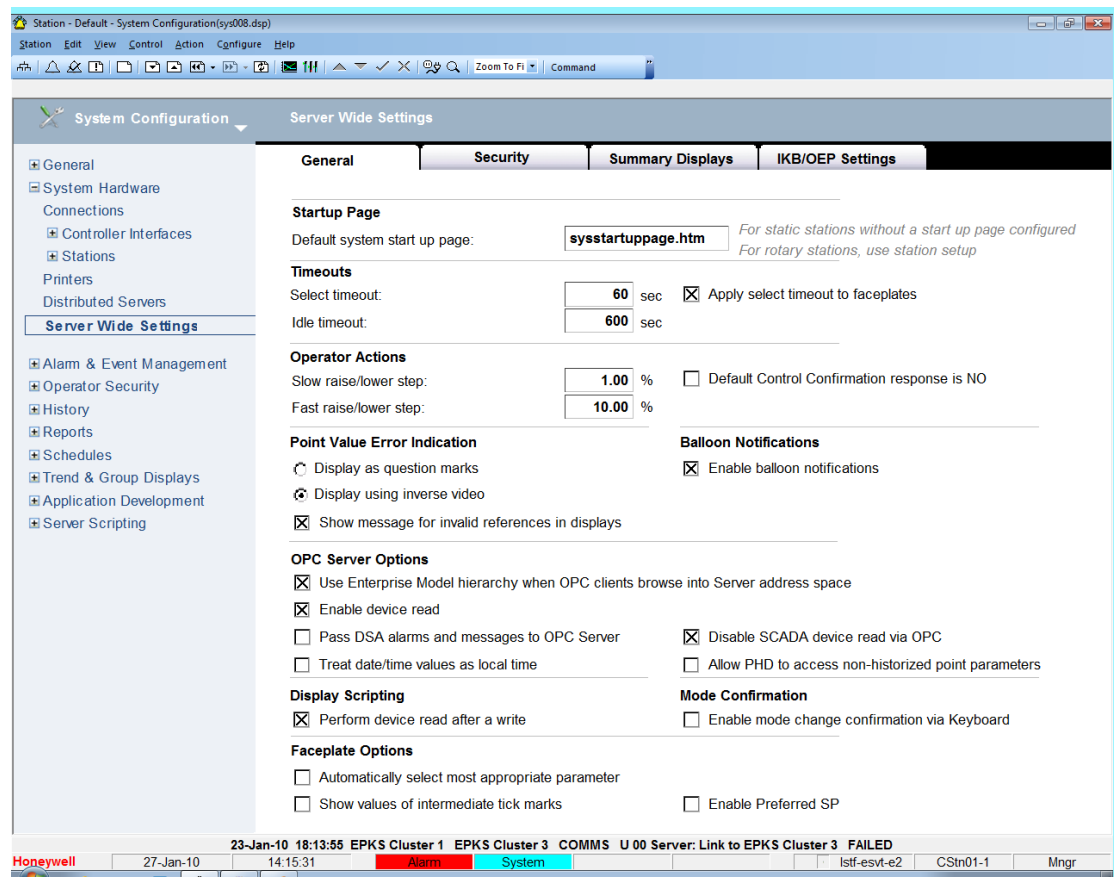


Figure 20: Server-Wide Station Settings Page

41.6 SafeView's graphical editor vs. text-based editor

SafeView workspaces may be viewed, edited, and saved using its Graphical Workspace Editor (GWE) or Text Editor (TextEdit) programs. The GWE allows you to visualize the workspace layout, graphically layout windows, move and resize windows, and so on. The Text Editor is similar to *notepad.exe* but also offers a very useful syntax check capability.

**Tip**

Older SafeView workspace examples provided with the Experion product contain comments that are lost when saving them from the GWE. To retain your comments across both the GWE and TextEdit, use the **Description** field. This is done for the workspace examples described herein so they can be viewed, edited, and/or saved in either environment without losing content.

**Tip**

The TextEdit syntax check is identical to the run-time workspace loader. So, if TextEdit validates the workspace, the workspace will load. The GWE check offers some heuristics for improving the workspace, but it is *not* identical to the TextEdit syntax check. Thus, before deploying a workspace, always use TextEdit to check syntax.

The GWE does not “write out” default values, such as ALWAYS ON TOP = *no*; or PLACEHOLDER = *no*. If you want to have these in your workspace files, *do not* use the GWE to save them.

41.7 The focus-based, programmatic, multi-display launch problem

One scenario where focus-based window management presents a problem is when SafeView is used to automatically populate a set of displays across multiple screens at startup. Normally, the operator selects an area of focus in which to invoke displays at startup, at which time multiple displays are immediately invoked. The problem occurs when the focus-based search mode continuously replaces one display with the next, leaving just one window populated with the last automatically-invoked display.

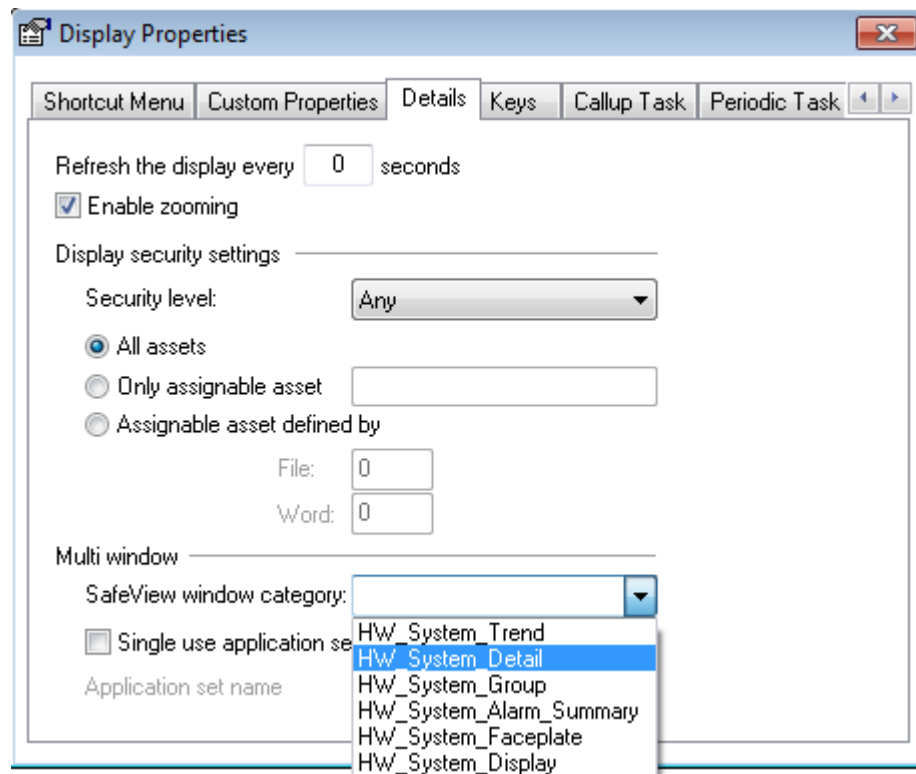
This problem is more generically characterized as the “programmatically multi-display launch problem” in which each display replaces the previous one that had focus. This may cause a problem for yoked display invocations, in which the invocation of a single display is always programmatically accompanied by the invocation of other displays.

Then again, depending on the types of yoked displays, focus-based matching may serve quite well, as when the yoked displays are associated faceplates and other displays that would not be confused with the main invoked display. If yoked displays are similar to the main invoked displays, you may want to consider the normal search mode with round robin behavior for main displays.

41.7.1 Workaround

The workaround for this problem is to make a custom copy of the few initial displays, each with highly unique category values. The procedure is as follows:

- 1 Determine which displays to launch at SafeView startup (or programmatically yoke).
- 2 Create a copy of the second, third, and fourth displays (such as *Tankfarm1b.htm* is copied to *Tankfarm1b.htm*).
- 3 Edit the copy of each display and modify its category to a unique value. See the following figure and the list of standard built-in Experion category values.



- 4 Modify your workspace such that only one main window matches the category value of each copied display. In the following example MATCH expression, “TankFarm1b” is excluded from a main display that typically accepts everything but trends, faceplates, and command/status displays.

```
match = module("?*") and not category("TankFarm1b") and not category("?*faceplate?*" ) and not  
category("?*trend?*") and not category("?*HSC_Station?*");
```

- 5 Include the copied displays in your startup commands. At startup (or when yoking displays), all windows will now be properly populated.

**Tip**

The copied displays are only used at startup (or yoking). The original displays can/will be used thereafter, since no other displays/applications launch them. So, next time you invoke “TankFarm1,” you just get your original “TankFarm1.”

41.8 SafeView's command line options

You can start SafeView with command line options to provide certain features as described in the following table. (Also see ““Command line options” on page 99.”)

| Command line option | Action |
|---------------------|--|
| /f<filepath> | Loads a workspace at startup. Note: Do not put spaces between “/f” and the workspace file pathname. |
| /cs | Makes SafeView's MATCH expressions case-sensitive. |
| /mp | Causes SafeView to attempt management of pre-existing displays when SafeView loads a workspace. |
| /nw | Disables the ability to invoke (CTL+ALT+W) the SafeView control panel. |
| /hp | Hides placeholders when SafeView loads a workspace. Note: This option is available only in SafeView R310 and newer versions. |

41.9 SafeView's facilitation of Station client tracing

SafeView supports automatic launching of applications at SafeView workspace load time. Of course, this includes Station and associated command options. Optionally, this can include other Honeywell and/or third party displays and applications. Thus, you can use SafeView to launch Station with a minimal level of debug-tracing by including the “Station client tracing” (“-T”) option, along with an output path filename such as “*station -T c:\temp\clienttrace.txt*.” SafeView can facilitate this tracing to be continuous, thus making traces available if Honeywell ever needs to investigate an issue (without perhaps the need for assistance in restarting Station with this option, and subsequent reproduction of the same problem). This option is provided in some “Safeview workspace examples” on page 153 herein and can be included in any SafeView workspace.

**Tip**

Recall that in the workspace file, using double backslashes (\\) as shown in the following example demonstrates launching Station with the “-T” option enabled and launching two initial displays:

```
"c:\\program files\\honeywell\\experion pks\\client\\station\\station.exe -T c:\\temp\\stationtrace.txt /windows  
sysAlarmSummary sysEventSummary"
```

41.10 Two MATCH-expression options for managing Experion faceplates

In multi-screen workspaces that are not focus-based, you can direct faceplates invoked from a particular display to the same physical window as that display. To do this, use the “_Monitor=<n>” category adjustment made by Station at run time. If using HMIWeb Display Builder to look at the display properties of Experion faceplates, the category is “HW_System_Faceplate.” However, the category is modified at run time, as Station appends the screen number associated with the display from which a faceplate is invoked. As a result, the faceplate category is (for example) “category(“HW_System_Faceplate_Monitor=1”).” This mechanism provides simple and effective focus-based matching capability for faceplates (and only faceplates.)

With focus-based workspaces, you can be more generic and leave each of the various screens' faceplate MATCH expressions as (for example) “category(“?*faceplate?*”).” This is because SafeView will look first in the local group when in focus-based mode as described above. This capability is generic and therefore not limited to faceplates, and thus may be applied to custom pop-up displays, trend displays, and so on.

41.11 Inclusive/exclusive 'everything except' MATCH expressions

Main display windows, particularly in focus-based workspaces, support most displays and offer choices of where to put various graphics. Consequently, MATCH expressions become more “homogenized” across main displays, trend/faceplate groups, and so on. That is, although faceplates can go to any screen, each is guided by the currently active window and thus displayed in the same window from which it was invoked.

For the main and other generic display windows, instead of enumerating or explicitly including a wide range of windows/categories in your MATCH expressions, consider including “everything except the few others.” This approach applies to most “Safeview workspace examples” on page 153 herein. So, for example, if you have trends and faceplates aside from main displays, you can say this for your main display MATCH expressions:

```
Match = module("?*") and not category("?*trend?*") and not category("?*faceplate?*") and not
category("HSC_Station?*")
```

Note: In this MATCH expression, the “and not category(“HSC_Station?*)” clause generically excludes all Station command/status windows.

You can use this same approach to direct specific displays to a particular screen. For example, in a dual-screen display, you can exclude all system displays from the left screen and display them only on the right screen by including “and not category(“HW_System?*)” to your MATCH expression as follows:

```
Match = module("?*") and not category("HSC_Station?*") and not category("HW_System?*")
```

Note: Since trend and faceplate categories also begin with “HW_System,” this MATCH expression excludes them as well.

Similarly, you can use this same approach to manage your own custom-categorized (or third-party) displays in specific main windows by simply excluding them from the MATCH expressions of the other main windows.



Tip

Avoid situations where you have a pop-up display that appears to SafeView as a main display. This is because SafeView will automatically close the old display to make room for the new display. In such cases, your pop-up display (intended to float-on-top) will end up closing its parent display. If you cannot implement such displays as true child displays, you may need to modify the MATCH expressions for your main displays to exclude these particular displays such that they are not managed by SafeView at all, but managed in their own specific SafeView windows.

41.12 Yoking faceplates to main displays

SafeView supports “display yoking” in which the invocation of a display is accompanied by simultaneous programmatic invocation of related displays. This can be tricky when yoking several main displays in a focus-based workspace. But, for other uses such as yoking trends, faceplates, or other custom displays, this can be done easily in any type of workspace. The typical example is a workspace that uses either focus-based management or the built-in “HW_System_Faceplate_Monitor=n” style matching. Simply put, when a new custom process graphic is invoked, it is accompanied by the invocation of one or more associated faceplates. The result has several benefits:

- Automatic invocation of related detailed (faceplate) information with no additional operator actions.
- Automatic closure of former faceplates that are likely unrelated to the newly invoked display.
- Consistent presentation layout, in which the screen is typically used in a similar manner (generally, leaving the right side of custom graphics with room for faceplates).

To yoke a faceplate to a given display, you must identify the specific faceplate display used for the point of interest and then make a script call that invokes that display for that point. For TPS points, the display type is determined by the *DISPTYPE* parameter value for the given point. So, for example, if the point name is A100, then the *DISPTYPE* is “A100.DISPTYPE” (this value can be retrieved through the Native Window's “datachg” display). If the *DISPTYPE* for A100 is “PID,” then the faceplate display for that point is “sysdtltpsPID_fp.htm” (similar for any other *DISPTYPE*, such as “sysdtltps<DISPTYPE>_fp.htm”). So, the script to yoke A100's faceplate to a given main custom graphic would be placed in the graphic's *Page OnPageComplete* event. The following example demonstrates yoking two explicitly selected points for a given graphic for faceplate display.

```
Sub Page_onpagecomplete
'Yoke A100's faceplate to this displays' invocation, A100.DISPTYPE="PID"
window.external.invokepopup "sysdtltpspid_fp.htm?currentpoint=A100", 0,0,1
'Also yoke A101's faceplate similarly, A101.DISPTYPE= "3INPUT"
window.external.invokepopup "sysdtltps3input_fp.htm?currentpoint=A101", 0,0,1
End Sub
```

41.13 Enhancing consistency and safety with region constraints

SafeView offers the capability to constrain a window or group of windows to a particular display region, thus enabling you to protect critical displays from being occluded by non-critical displays. It also protects operators from the potential for unintended/unexpected manual or programmatic display movements. Region constraints offer an extra level of consistency and safety, particularly for main displays that should always be in a “locked-down” (fixed) position. The following is an example of a region constraint line:

```
Region = 0, 0, xresolution, yresolution;
```

This region constraint line precludes a given window from being moved anywhere but the primary monitor. For locked-down main displays, set the REGION values (*left, top, width, height*) to match the original POSITION and SIZE as specified in the SafeView workspace for the given window. Also, to protect against sizing changes within the allowable region, fully locked-down displays can be constrained from being sized, dragged, and having their minimum and maximum sizes explicitly set (to original size). Doing this prevents that window from being altered for any reason.

Similarly, faceplates should be region-constrained at a minimum to the (parent) screen that owns them. It is a good practice to constrain them from being resized, maximized, and so on. You may even want to lock them down as well, but be careful if you have a workspace where the faceplate is allowed to occlude a portion of your main display. You want the operator to view an entire display, which may require moving a faceplate.



Tip

Faceplates and trends should always be configured as “always on top” when allowed to float over main displays. This prevents them from being hidden behind main displays and using system resources unnecessarily.



Tip

Region constraints need not be bound to screen resolutions; you can region-constrain windows to a particular area of a screen, as is the case for trends and faceplates. In many workspace examples provided herein, faceplates can be dragged around and even maximized, but are region-constrained to an area that limits them to double-size on the right side. Similarly, trends can be maximized; but even when maximized, their region constraints keep them from occluding (for example) command and status windows.

41.14 Closing faceplates automatically on main display close

In Station's multi-window mode, it may be best to manually close certain faceplates after the main display on the same screen has been replaced. To do this automatically, you can insert a script into each custom main display in your library that leverages SafeView API calls. Since this scripting approach is not necessarily limited to faceplates, you can use it to also automatically close trends and other supporting displays in the same manner.

41.14.1 Example scenario

You have three tanks, Tank1, Tank2, and Tank3. Each has its own graphic and two associated faceplates for various points, but all three graphics are nearly identical. While monitoring the Tank1 graphic and its two faceplates, you notice that the Tank1 level is getting low. So, to compare Tank1 with Tank2, you invoke the Tank2 display. *However, you neglect to manually close the two Tank1 faceplates and then invoke the two Tank2 faceplates.* Consequently, the Tank2 display replaces the Tank1 display, but the two Tank1 faceplates remain open.

After leaving the station for a period, you return to see that the Tank2 level is getting too high. Quickly, you attempt to decrease the output of Tank2 by adjusting (what you perceive to be) the associated point on its faceplate. Mistakenly, you used point *T201* to further decrease the already low level in *Tank1* instead of using point *T202* for *Tank2*. This mistake causes Tank1 to empty, thereby triggering an alarm and prompting a visit from the supervisor.

To address this situation, you should auto-close faceplates for custom graphics whenever those graphics are replaced or closed. You can do this easily by using scripted SafeView API calls as demonstrated in the following example script. Inserting code such as this into each of your custom graphics that fit this scenario will automatically close the displays associated with the former display.



Tip

Automatically closing faceplates would not be necessary if you also yoke faceplates as described in this section. If doing both activities, you should consider using a scripted timer to delay invocation of a yoked faceplate to avoid having it closed by the script.

41.14.2 Example script

The following script example closes a single faceplate per screen. Although this example supports dual or quad ICON workspaces, you may edit it as necessary for your scenario. Insert this script into the Page_OnUnload event on your main display page.



Tip

Be sure to change the names of the faceplate window specifications to match your workspace.

```
Sub Page_onunload
dim sv
dim FPLL1
dim FPLL2
dim FPLR1
dim FPLR2
dim FPUL1
dim FPUL2
dim FPUR1
dim FPUR2
set sv = CreateObject("Honeywell.workspace.client")
FPLL1 = sv.getwindowhandle("FPUpperLeft")
FPLL2 = sv.getwindowhandle("FPLowerLeft2")
FPLR1 = sv.getwindowhandle("FPLowerRight")
FPLR2 = sv.getwindowhandle("FPLowerRight2")
FPUL1 = sv.getwindowhandle("FPUpperLeft")
FPUL2 = sv.getwindowhandle("FPUpperLeft2")
FPUR1 = sv.getwindowhandle("FPUpperRight")
FPUR2 = sv.getwindowhandle("FPUpperRight2")
'dual-screen side-by-side
```

```

if window.external.Appwindow.Left < 500 then
sv.closewindow(FPLL1)
sv.closewindow(FPLL2)
elseif window.external.Appwindow.Left > 500 then
sv.closewindow(FPLR1)
sv.closewindow(FPLR2)
end if
'Use the following code for the faceplate closing business on site when 4-screen safeview is
installed
if window.external.Appwindow.Left < 500 and window.external.Appwindow.Top < 0 then
sv.closewindow(FPUL1)
sv.closewindow(FPUL2)
elseif window.external.Appwindow.Left > 500 and window.external.Appwindow.Top < 0 then
sv.closewindow(FPUR1)
sv.closewindow(FPUR2)
elseif window.external.Appwindow.Left < 500 and window.external.Appwindow.Top > 0 then
sv.closewindow(FPLL1)
sv.closewindow(FPLL2)
elseif window.external.Appwindow.Left > 500 and window.external.Appwindow.Top > 0 then
sv.closewindow(FPLR1)
sv.closewindow(FPLR2)
end if
End Sub

```

41.15 Managing cross-screen, focus-based displays

Because SafeView's focus-based mode is oriented around window groups instead of physical screens, you have the flexibility to “yoke” displays independent of physical screen locations. One benefit of yoking is the ability to manage windows in a particular screen, across halves of a quad-screen ICON, or to “cross screen invoke” displays on the same node. Several “Safeview workspace examples” on page 153 demonstrate invocation of trends to opposite (or upper) screens.

This yoking ability can allow you to, for example, view a main display on the left screen and invoke a trend for a point from that display. As a result, the trend appears on the right screen, thereby not obscuring the display of interest on the left screen. This is done by placing the right screen Trend (or Trend group) into the window group that also includes the left screen's main display. See the following figure of SafeView's Graphical Editor tree view of a workspace that demonstrates this concept. (In the figure, note that the “LeftScreen” group contains a trend group with “TrendRight1” and “TrendRight2.”) Similarly, the ICON workspaces in ““Safeview workspace examples” on page 153” manage trends from upper and lower screens of either side of the ICON to a trend group on the upper screen of whichever side invokes the trend.

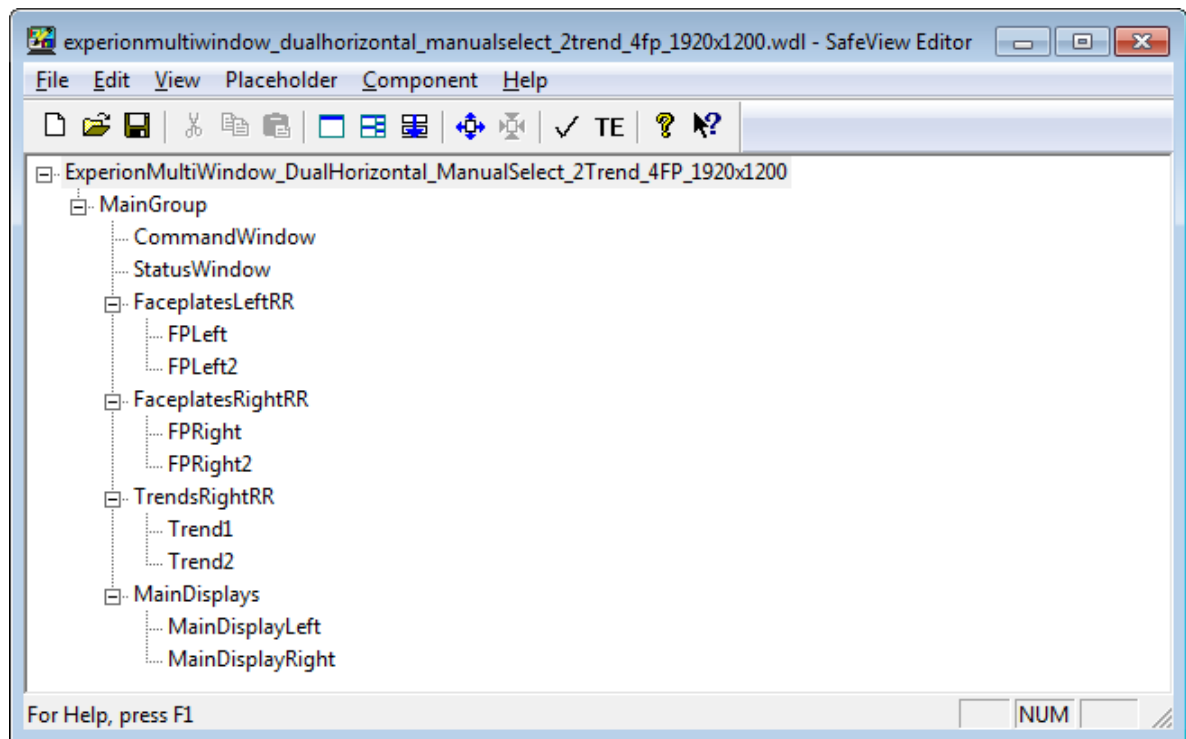


Figure 21: Example: SafeView's Graphical Editor Tree View



Tip

Be aware of the consequences of having overlapping displays. You should avoid situations where multiple displays can partially or completely overlap one another, as this will unnecessarily consume system resources. For example, if you offer multiple trends to your operators, but they typically only use one trend but invoke several, and then maximize one trend for extended use, the underlying trends (and the main display at the bottom) will continue to burden the system, using data access and computing resources.

41.16 Supporting multiple catch-all windows

SafeView workspaces typically support a “catch-all” window as the match of last resort. This window's MATCH expression is so wide open that it can catch virtually everything. For that reason, it should be placed at the bottom of the workspace in an outer first-match group, and outside any interior groupings that support focus-based searching.

Typically, only a single catch-all window is used, meaning that only one “other” display is supported at a time. However, you may need to support more than one other such application display simultaneously. You can do this using a few methods. First, the following example demonstrates proper positioning for a catch-all window, thus guaranteeing it is examined “always last,” even in a focus-based workspace:

```
Group Main group("first match")
Group LeftSide("first match")
Group LowerLeft("first match")
Window commandwindow...
Window MainLL
Group LowerRight("first match")
Etc..
End group
Group RightSide("first match")
Group LowerRight("first match")
Group LRFaceplates("round robin")
Etc.
Group LowerLeft("first match")
Etc.
End group
Window catch-all
End group
```

To support multiple catch-all windows, consider the following:

- Provide a catch-all window group, such as a round-robin group, that supports whatever number of displays you need to support. Note that SafeView will close the oldest of such displays once the limit of displays is reached.
- Use a single catch-all window, but provide the windows specification option, MAXWINDOWS = *infinite*, for the catch-all window. This informs SafeView to use this same window specification for any number of displays simultaneously, leaving the closure of such displays to the operator. Note that with this approach, you have the possibility of having displays hidden beneath other displays or under your main display. To prevent loss of catch-all displays, you can make them “always on top.”



Tip

Use the MAXWINDOWS = *infinite* option with caution because it allows you to open an unlimited number of such displays. This may prevent you from invoking additional Station displays (until existing displays are closed).

41.17 Using wildcard character-matching

SafeView MATCH expressions support “wildcards” to simplify expressions. Wildcard expressions are helpful if you are unsure of how a particular set of values begins or ends, or there is a certain amount of variation. The usual method is to “wildcard” the beginning or end of a given string to match a set of strings, instead of explicitly calling out each one. For example, the following two MATCH expressions accomplish similar ends:

```
Match = category("overview1") or category("overview2") or category("overview3");
Match = category("overview?*");
```

Wildcards work well for faceplates, as they are dynamically modified by Station to include “_Monitor=*n*” where *n* is the current screen. So, instead of “HW_System_Faceplate,” the actual category at run time may be “HW_System_Faceplate_Monitor=1.” A convenient MATCH expression for faceplates would be:

```
Match = category("HW_System_Faceplate?*");
Match = category("?*faceplate?*"); (this could also match GUS faceplates)
```

Similarly, in focus-based matching, you can exclude all Station command windows (one or more may be configured via Station connection properties) as well as the Station status bar with a single wildcard clause:

```
Match = category("?*") and not category("?*faceplate?*") and not category("HW_Station?*");
```

Using wildcards for MATCH expressions may be used for window titles and application modules (file paths) as well. For example, if you have GUS displays and wish to manage them in a given window or window group, you could use the following MATCH expression:

```
Match = module("?*gpb?*");
```

Or, you could be explicit:

```
Match = module("c:\program files\honeywell\tps\gus\gpb.exe")
```

Similarly, you can wildcard window title matches, for example:

```
Match = title("?*Microsoft?*");
```



Tip

SafeView supports full regular-expression wildcard pattern matching.

41.18 Constant-based vs. literal-based coordinates

The most useful feature of SafeView's GWE is perhaps its ability to provide a build-time visualization of a workspace, allowing you to verify window locations, sizes, and general layout. You can do this using the constant-based method or the graphical layout (literal) method. (It is best to choose your preferred method and be consistent in using it.)

- In the constant-based approach, SafeView allows you to define constants to use when specifying numbers such as window sizes, regions, and positions. This is convenient for creating a workspace that can easily accommodate changes to screen resolution, or adjusting other windows in response to resizing another window.
- In the graphical (literal) layout approach, SafeView allows you to graphically layout and “snapshot” window positions. This is a convenient method because you can simply manipulate window sizes and positions, snapshot, and be finished.

Each of the “Safeview workspace examples” on page 153 herein uses the constant-based approach with a consistent naming convention. Using constants to specify your numbers allows you to modify the entire layout by simply changing the appropriate constants. For example, to modify a workspace designed for 1280x1024 resolution screens to work on 1600x1200 (or other) resolution screens, simply change the *xresolution* and *yresolution* constants.



Tip

Newer versions of SafeView's GWE support viewing multi-screen workspaces on a single screen by offering 15%, 25%, 50%, and normal 100% views. This capability was used to create several screen shots for this document. Since these are actual windows presented by the GWE, their title bars were not “shrunk” accordingly when shown in less than normal 100% view. So, in some cases, when shown in 15% or 25% view, the titles become occluded. In other than the 100% view, editing workspace data in the GWE is disabled, though you can open other workspace files without reverting to 100% view.



Tip

Use the GWE to view the workspace layout and to select specific check box options, but use TextEdit to make constant and/or layout-oriented changes.

Unless you do not mind losing the flexibility of tweaking a workspace by changing constants, avoid graphically editing the window positions such as moving placeholder windows and selecting **Update from placeholder** to update the workspace position data. Doing so will (1) remove the constants from that placeholder position data, and (2) update its position instead with the literal values of its present location.

42 Notices

Trademarks

Experion®, PlantScape®, SafeBrowse®, TotalPlant®, and TDC 3000® are registered trademarks of Honeywell International, Inc.

OneWireless™ is a trademark of Honeywell International, Inc.

Other trademarks

Microsoft and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Trademarks that appear in this document are used only to the benefit of the trademark owner, with no intention of trademark infringement.

Third-party licenses

This product may contain or be derived from materials, including software, of third parties. The third party materials may be subject to licenses, notices, restrictions and obligations imposed by the licensor. The licenses, notices, restrictions and obligations, if any, may be found in the materials accompanying the product, in the documents or files accompanying such third party materials, in a file named third_party_licenses on the media containing the product, or at <http://www.honeywell.com/ps/thirdpartylicenses>.

42.1 Documentation feedback

You can find the most up-to-date documents on the Honeywell Process Solutions support website at:

<http://www.honeywellprocess.com/support>

If you have comments about Honeywell Process Solutions documentation, send your feedback to:

hpsdocs@honeywell.com

Use this email address to provide feedback, or to report errors and omissions in the documentation. For immediate help with a technical problem, contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

42.2 How to report a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, please follow the instructions at:

<https://honeywell.com/pages/vulnerabilityreporting.aspx>

Submit the requested information to Honeywell using one of the following methods:

- Send an email to security@honeywell.com.
- or
- Contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

42.3 Support

For support, contact your local Honeywell Process Solutions Customer Contact Center (CCC). To find your local CCC visit the website, <https://www.honeywellprocess.com/en-US/contact-us/customer-support-contacts/Pages/default.aspx>.

42.4 Training classes

Honeywell holds technical training classes on Experion PKS. These classes are taught by experts in the field of process control systems. For more information about these classes, contact your Honeywell representative, or see <http://www.automationcollege.com>.

