JIS Interface Server:

Basic User's Guide

Version 9.3

September 2026
(originally released January 2005)

This document applies to JIS Interface Server Version 9.3 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1992–2026 Software GmbH, Darmstadt, Germany and/or their suppliers. All rights reserved.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s). Document ID: JIS-BASIC-UG-93-20260923

Table of Contents

About this Guide 25
Documentation Set 26
Document Conventions 27
Viewing the Documentation Online 28

Chapter 1. Creating Screen Images 29
Creating Screen Images Using Screen Definition Files 29
Screen Definition File Types 30
AS/400 Applications 31
Mainframe Applications 32
AS/400 Applications and Screen Definition Files 32
The Recommended Workflow 32
Screen Model Type 33
Screen Model Type 2 33
Screen Model Type 5 34
Using Libraries to Contain Different Screen Model Types. 34
Creating Screen Images 35
The Create Screen Images From Compiled DDS Dialog Box 35
Features of the Dialog Box 36
Handling Protected and Hidden Fields 38
Mainframe Applications and Screen Definition Files. 38
The Recommended Workflow 39
Screen Model Type 39
Screen Model Type 2 39
Screen Model Types 3, 4, and 5 39
Using Libraries to Contain Different Screen Model Types. 40
Creating Screen Images 40
Screen Image Name Source 41
Capturing Your Host Application 41
Creating Screen Images From Screen Captures 41
Deleting Captured Screen Images 42
Combining a Screen Capture and a BMS or MFS File. 42
Maintaining Screen Images. 42
Editing Screen Images. 43
Accessing the Screen Image Editor 43
Via the Host Menu 44
Via the Utility Menu 44
Using the Screen Image Editor 44
Moving the Cursor 45
RMB Functionality 46
Changing the Default Cursor Location. 46
Changing colors 47
Adding or Removing Extended Attributes 48

Adding, Editing, and Deleting Attributes 48
Adding and Deleting Text 50

Chapter 2. Comparing the Host Screen and the GUI Window 51
Host View 51
What You See in Host View 52
What you can do in Host View 52
Edit an Incorrect Screen Image 52
Display or Hide Attributes in Screen Views 53
Test View 53
What You See in Test View 54
What You Can Do in Test View 54
View Attached Functionality 54
Change Table Appearance 54

Chapter 3. How Pattern Definitions Work 55
The Idea Behind Pattern Definitions 55
What is a Pattern Definition? 56
The Parts of a Pattern Definition 56
Examples Using String Type Pattern Definitions 57
How ACE Applies Pattern Definitions to the Host Screen 58
How ACE Chooses Which Pattern Definition to Apply 59
Sections in Brief. 60
Primary Patterns in Brief 60
The Location Parameter 60
Compound Pattern Definitions 62
Horizontal Groups: A First Look 62
Scanning and Child Patterns 64
OneOf Type Pattern Definitions 64
Horizontal Groups and OneOfs: Putting it Together 65
Combining OneOf Type Pattern Definitions 65
Recognizing Real Screens: Character Sets and Iterations 66
Strings Have Their Uses. 67
Character Set Type Pattern Definitions 67
Why Have Character Sets?. 68
Using Character Sets 68
Flexibility Versus Specificity 69
Iteration Type Pattern Definitions 69
Properties of Iterations 70
Using Iterations 70
Recognizing Words 71
Recognizing Combinations of Words 71
Other Compound Pattern Definition Types 73
Dynamic Pattern Definitions 73
How ACE Uses Regular Pattern Definitions 73
Regular Pattern Definitions and Dynamic Screens 74
How Dynamic Pattern Definitions Work 74

Why Use Regular Pattern Definitions? 75
Dynamic Pattern Definitions in Detail 75
When Does ACE Use Dynamic Pattern Definitions? 76

Chapter 4. Sections and Layouts 77
Introduction to Sections and Layouts 77
What Are Sections? 78
What Are Layouts? 79
What is Layout View? 79
How Sections Improve Host Screen Analysis 80
Why Use Sections? 80
List Sections Applied by the ACE Wizards 82
The Scanning Order for Pattern Definitions 82
About Ordering Pattern Definitions 82
How Sections are Ordered 83
Filter Sections 83
How Filter Sections Work 84
Why Regular Sections are Necessary 86
Using Layouts 86

Chapter 5. Layout View Operations 89
What You See in Layout View 90
Host Session Window 91
Sections To Drag Panel 92
Section Explanation Panel 92
Show/Hide Sections to Drag and Section Explanation 93
Operations with Sections 93
Moving a Section on the Screen Image 93
Resizing a Section on the Screen Image 94
Applying a Section to the Screen Image 94
Removing Sections from the Screen Image 96
Replacing One Section with Another on the Screen Image 96
Using Wizards to Mark Menus or Lists on the Screen Image 97
Editing Sections 97
Creating New Sections 99
Deleting Sections 100
Sections and Pattern Definition Search Order 100
Pattern Definition Priority Within a Section 100
Establishing the Order of Pattern Definitions 101
Operations with Layouts 102
Removing a Layout from the Screen Image 102
Applying a New Layout to the Screen Image 102
Editing Existing Layouts 103
Creating New Layouts 103
Writing Layouts to INI files 104
Filter Section Operations 104
Displaying Filter Sections 105

Working with Filter Sections 105
Removing Filter Sections 106
BMS Filter Sections for Mainframes 106
DDS Filter Sections for AS/400s 107
Menus Specific to Layout View 108
Layout Menu 108
Primary/Filter Section Submenu 109

Chapter 6. Operations Performed in Analysis View 111
The Analyzed Screen 112
The Coloring Scheme 113
Tasks You Can Perform in Analysis View 114
Moving the Focus Through the Hierarchy of a Pattern Definition 114
Ignoring a Pattern Definition 117
Modifying the Location or Dimension of a Pattern Definition 117
Modifying a Pattern Definition 118
Setting an Example 118
Copying Text to the Clipboard 119
Editing a List 120

Chapter 7. Pattern Definition Modifications Through the KnowledgeBase 121
About Working with the KnowledgeBase 121
Overview of the KnowledgeBase Definitions Window 122
The Parts of the KnowledgeBase Definitions Window 123
Pane Operations 124
Viewing Pattern Definitions 125
Navigating the Pattern Panes 128
Filtering a Pattern Pane 129
Viewing Message Definitions 134
The Lower Message Pane 134
Viewing Pattern Definitions that are Message Definitions 135
Querying the Use of Pattern Definitions 135
Editing Pattern Definition Structure 136
Saving the KnowledgeBase 136
KnowledgeBase Validation 136
About the Structural Aspects of Editing Pattern Definitions 138
Top Level and Child Pattern Definitions 138
Cutting a Pattern Definition 139
Replacing a Child Pattern. 139
Adding a New Child Pattern to a Parent Pattern 140
Duplicating a Pattern Definition 143
Creating a New Pattern Definition 144
Designating a Pattern Definition as a Message Definition 147
Editing Pattern Definition Properties 147
The Properties Pane 147
Pattern Definition Types 148
Character Set 148

Dynamic Group 149
Dynamic Iteration 149
Horizontal Group 150
Horizontal Iteration 151
List Column. 152
List with Parameters 152
One Of. 153
Popup Border 153
Scattered Group. 154
String 155
Vertical Group 155
Vertical Iteration 156
The Extended Info Tab 157
Primary Patterns 157
The Set/Change Location Tab 158
Setting Location 158
Using the Set/Change Location Tab 158

Chapter 8. Subapplication-Specific Modifications of the Design 161
Design View 162
Control Appearance vs. Control Functionality 163
Representation Definitions 163
Other Representations 164
GUI Control Types 165
Working in Design View 166
Design View Toolbar 167
Design View Palettes 167
View Menu 168
Design View Grid 168
Highlighting Overlapping Controls 170
The Design Menu 171
Control Editing 173
Selecting Controls in the Window 173
Selecting Controls Individually 173
Selecting Groups of Controls 174
Leading Controls. 176
Arranging Controls on the Window 176
Control Editing Palette 177
Adding Controls to the Window 181
Definitions Palette. 181
User-Added Controls 183
Modifying Control Properties 184
Window Components Palette 184
Component Properties Pane 185
Deleting Components 188
Renaming Components. 189
Control Component Dialog Box 190
Positioning Controls 191

Sizing Controls 192
Undo Repositioning or Resizing 193
Design View Shortcut Keys 193
Undoing Changes 193
Undo/Redo Behavior 194
Performing an Undo Operation 194
Performing a Redo Operation 195
Undo Limitations 196
Attaching Functionality to Controls 197
Attaching a Method to a Control 197

Chapter 9. Representation Definitions 199
Representation Definitions and How They Work. 199
Components and Controls 200
Representation Definitions With More Than One Component 200
Representation Definitions for Lower Level Pattern Definitions 200
Components and the Subapplication 201
Multiple Components of the Same Type 202
Editing a Representation Definition 203
Accessing the Representation Definitions 203
Deleting a Representation Definition. 205
Creating Representation Definitions and Components 205
Creating Floating Representation Definitions and Components 206
Deleting a Component 207
Renaming Floating Representations 207
Relative Placement of Floating Representation Controls 208
Representation Component Properties 211
The Screen Tab 212
Pattern Definition Tree. 212
Screen Options 214
The Manager Tab 215
Runtime Data Flow 216
Variable Names 218
Read Data From INI File 220
The Style Tab 221
Modifying Component Styles 221
Validity Checks 221
Internal Validity Checks. 222
External Validity Checks 224
Control Types and Styles 225
Accelerator 225
Button. 226
Check Box 231
Combo Box 233
Date Field 236
Dynamic Group 237
Frame 238
GeneralUTMethod 240

Group Box 240
HostBasedFormatValues 242
Line 246
Link 247
MenuItem 249
Prompt 250
Radio Group 252
Spin 254
Special Considerations for iSeries Host Applications 256
Static 257
SubWindow 260
Table 262
Tabs 267
Variable 268
Window 268
TextBox 270
The Format Tab 277
Modifying Component Formats 277
Check Box 278
Combo Box, Radio Group or List Box 281
Text Formatting 282

Chapter 10. Editing Menus in Design View 287
The Menu Bar 287
The Structure of a Pull-down Menu 288
Selecting the Menu Type 289
Subapplication Menu Editor 289
Editing Options 290
The Menu Items List Box 290
The Properties Button 291
Working with the Menu Editor 291
Creating a New Top Level Menu 292
Adding a Menu Item to a Top Level Menu 292
Adding Sub-menus 292
Reordering Menu Items 293
Changing the Name of a Menu Item 293
Attaching Functionality to Window Menu Items 293
Adding a Separator to a Menu 294
Creating an Accelerator for a Menu Item 294
Creating a Hotkey for a Menu Item 295
Floating Menus 297
Attaching Functionality to a Floating Menu 299
Changing the Default Menu for All Subapplications 303
Creating Menus Automatically with Representation Definitions 303
Creating an Automatic Menu Item 304

Chapter 11. Color and Font Design 307
About Control Colors 308
Color Specification Mechanisms 308
The Scope of a Color Mechanism 309
Specifying the Color Mechanism Globally 309
Specifying the Color Mechanism Locally 309
Setting a Control's Color in the Font and Color Dialog Box 310
Working with Colors 310
Fixed Colors 311
Colors Depending on the User's Windows Setup 312
Table of Windows System Elements 312
Remarks about Window System Colors. 314
Color Tables 314
Selecting an Existing Color Table 315
Modifying an Existing Color Table 315
Creating a New Color Table. 315
Color Table Settings. 316
Combining Host Dependent and Host Independent Colors 317
Selecting the Font. 318
Defining Fonts to Produce a Consistent GUI 318
Measuring Changes in Control and Font Sizes 319
Combining Items Measured in Pixels and Dialog Units 320
How This Affects the Appearance of Your GUI in ACE. 320
Suggested Fonts 321

Chapter 12. Tabbing Order Modifications 323
Setting Tabbing Order 323
The Tabbing Order 323
How ACE Sets the Tabbing Order. 323
Automatic Recalculation of the Tabbing Order 324
How to Specify the Tabbing Order Mechanism that ACE Uses 324
Viewing the Tabbing Order 325
Modifying Tabbing Order 325
Entering Modify Tabbing Order Mode 326
Local Tabbing Order Modifications. 328
Returning to Design View 331
Tabbing Order in Subwindows and Tab Controls 331
Limitations 332

Chapter 13. Methods: Attaching Functionality to the GUI Window 335
What are Methods?. 335
Method Types and Method Levels 336
User-Triggered Methods 337
Examples of User-Triggered Methods 338
System-Triggered Methods 339
Example of a System-Triggered Method 340
Events and System-Triggered Methods 341

General Methods 345
General User-Triggered Methods 346
General System-Triggered Methods 347
Current Subapplication Methods and Current Library Methods 347
Accessing Methods 348
User-Triggered Methods Manager 348
System-Triggered Methods Manager 350
UserRMB Methods 352
User-Triggered Methods vs. System-Triggered Methods 354
Linking User-Triggered Methods to Controls 355
Linking Methods to Controls in a Subapplication 356
Detaching Methods from Controls 357
Linking Methods to Controls in the KnowledgeBase 357
Writing Methods 359
The Define Method Dialog Box 359
Working with Method Lines 360
Method Operations: User-Triggered Methods 363
Modifying User-Triggered Methods 363
Writing New User-Triggered Methods 364
Renaming User-Triggered Methods 364
Deleting User-Triggered Methods 364
Method Operations: System-Triggered Methods 365
Modifying System-Triggered Methods 365
Emptying System-Triggered Methods 365
Method Line Types 366
Writing Method Lines 368
DoMethod Method Line Type 383
Enter Method: an Example 393
Referencing a Method Line from Another Method Line 394
Method Examples 397
Working with Variables 397
Working with Messages 398
Using Conditions 398
Reading Values from *.ini Files 399
Pressing a Host Key 400

Chapter 14. Methods: System-Triggered Methods List 407
Return Values 408
AfterPageUpDown 409
GetToBottomOfList 409
GetToTopOfList 410
PageDown 410
PageUp 411
TableChangedSelection 411
UserAcceptScreen 412
UserAfterRefreshSubApplication 413
UserAfterTabFolderChanged 414
UserAfterTableAction 415

UserAttentionRequest 415
UserBack 416
UserBeforeTabFolderChanged 416
UserBeforeTableAction 417
UserCloseSubApplWindow 417
UserDestroyApplication 418
UserDestroySubApplication 419
UserHostHelpRequest 419
UserHostMessageHelpRequest 420
UserInitApplication 421
UserInitBeforeFirstSubAppl 421
UserInitSubApplication 422
UserIsRealMessage 422
UserMoveToDependentScreen 423
UserMWIRequest 424
UserPreUpdateInitSubAppl 424
UserRefreshSubApplication 425
UserServerDataReady 426
UserShouldCloseSubApplWindow 426
UserShouldWindowBeBuilt 427
UserSkipSubApplication 427

Chapter 15. Methods: DoMethods List 429

Chapter 16. Runtime Screen Identification View 465
How Runtime Screen Identification Works 465
The Runtime Screen Identification Menu 466
The Runtime Screen Identification Screen. 467
The Color-coding Scheme 467
Fixed and Variable Characters in a Screen. 468
Message Areas in a Screen. 470
The Screen Fingerprint 471
The Location of the Fingerprint 472
Using a Pattern Definition as a Fingerprint 473
Undoing Your Changes 473
The Behavior of Undo Changes 473
Performing an Undo 474
From the Toolbar 474
From the Edit Menu. 474
From the Keyboard 474
Performing a Redo 475
From the Toolbar 475
From the Edit Menu. 475
From the Keyboard 475
Automatic Identification Definitions 475
Comparing a Captured Screen Image and a Screen in Runtime Screen Identification View. 479
Capturing a Host Screen Image in Runtime 480

Comparing Two Screen Images 480

Chapter 17. Runtime Field Information View 483
How Runtime Field Information Works 484
Variable Patterns 485
The Runtime Field Information Screen. 485
The Color-Coding Scheme 487
The Runtime Field Information Menu 487
The Decomposition Definition Properties Dialog Box 488
Fixed Data 488
Lock Selection. 489
Location 489
Dimensions 490
List Options. 490
Modifying the Range of a Pattern Definition's Location 490
Modifying the Range of a Pattern Definition's Dimensions 492

Chapter 18. Multiple Subapplication Features 497
Using Batch Processing. 497
Using Subsets 498
Accessing Batch Mode 498
Implementing Batch Processing 498
Processing an Application or a library 499
The Batch Processing Dialog Box 499
The Log Files 500
The Batch Changes Log 500
The Subapplication Log 502
Defining Logging Parameters 502
The Log Options Tabs. 503
Query. 506
The Query Dialog Box 507
Selecting Subapplications 507
Query by Conditions 507
Performing a Query. 508
Creating a Subapplication Subset 509

Chapter 19. The Runtime Application 511
Generating the Runtime. 511
How Does the Compiler Work? 511
Configuring the Generate Runtime Process 512
Generate Runtime Including all the Subapplications 513
Generating Runtime Using Subsets 513
Creating a Test Subset. 513
Configuring Your Test Subset 514
Subsets in the Converter INI File 515
Producing the Executable File 515
The Generate Runtime Command. 516

Suppressing Error Messages While Compiling 517
Running the Generated Application 518
Running Your Application from the Windows Program Manager 518
About the Runtime Application 518
Creating the Runtime Installation File 518
Runtime License. 519
System Setup 519
Types of Emulators 519
Demos and Testing 520
Setting Up PANELS.INI 521

List of Figures

Screen image sources 29
DDS compiler workflow from iSeries to PC 32
DDS compiler workflow in ACE 33
Setting the screen model type. 34
Create Screen Images from Compiled DDS step in wizard 35
Displaying extra fields and protected fields 38
Choices for setting the model type 40
Edit Screen Images wizard 44
Screen Image Editor wizard via the Utility menu 44
Properties panel 45
Editable place on the screen is where cursor is blinking 45
RMB host session options 46
Set cursor location 47
Selecting the desired color 47
Paint frame 48
Editing attributes 49
The screen image in host view 51
The GUI representation of the host screen 53
WindowCaption section 78
A layout 79
Top region of a host screen 80
Top region of a GUI window derived from the last host screen 81
Members of separate sections. 81
FixedFontHeader section 82
Filter sections 84
Filter sections in the top section of the host screen 85
Filter sections sit between the attributes. 85
Layout view 90
The colored rectangles in Layout view 91
Sections to drag panel. 92
Section explanation panel. 92
Section definitions manager 95
Primary section definition dialog box 98
A screen image in analysis view 112
The coloring scheme 113
Dynamic analyzed screen 115
Frozen analyzed screen 115
Copying text via the RMB 119
KnowledgeBase Definitions window 123
Pattern Definition View in the KnowledgeBase Definitions Window. 125
Find dialog box. 128
Display Criteria Setup window 131
Display Criteria Setup dialog box 133

Display Criteria Setup dialog box in Modify mode 133
Message Definition view in the KnowledgeBase Definitions window. 134
Duplicate Pattern dialog box 144
New Pattern dialog box 145
Character Set Parameters tab. 149
Dynamic Iteration Parameters tab 150
Horizontal Iteration Parameters tab. 151
List with Parameters' Parameters tab 152
Popup border Parameters tab. 154
String Parameters tab. 155
Vertical Iteration Parameters tab 156
Extended Info tab 157
Set/Change Location tab 158
Host screen transformation into GUI window. 162
Common GUI controls 165
Design View 166
Design View toolbar 167
Always on Top option 167
Toggle viewing of Design View Palettes 168
Toggle viewing of Design View grid 168
:Setting grid properties in the Window Options dialog box 169
Design View menu options 171
Selecting controls individually 173
Selecting groups of controls 174
Select menu options. 174
Local editing options in Design View 177
Control Editing Palette options 177
Advanced Editing dialog box 180
Definitions Palette 181
Filter definitions in Definitions palette 182
Display criteria added in RD View in the KnowledgeBase 182
Finding a specific definition in the Definitions Palette 183
Window Components Palette 185
Undo from the toolbar 194
Undo from the Edit menu 195
Redo from the toolbar 195
Redo from the Edit menu 196
Subapplication components. 201
Multiple components of the same type 202
Representation Definition view in KnowledgeBase Definitions window. 204
New Representation dialog box. 206
Component Arrangement tab in Representation Definitions view 208
Defining offset and size. 209
Pattern definition tree in the Screen tab 212
Expanded pattern definition tree in the Screen tab 213
Manager tab. 215
CompareValue Parameters dialog box 223
Accelerator Style tab 226

Key section 226
Button Style tab 227
Button associated images 229
Button placement relative to text 229
Button special effects 230
CheckBox Style tab 231
ListBox control Style tab 233
Date control Style tab 236
Runtime control ordering section 237
Frame control 238
Frame control Style tab 239
Methods list in Style tab 240
Group Box control Style tab 241
Pattern definition including EntryPair definition 242
Pattern definition including Entry definition 243
HostBasedFormatValues Style tab 243
Correct usage for three of the HostBasedFormatValues styles 244
Updating formats created using HostBasedFormatValues 245
Line control styles 246
Line control Style tab 246
Link control 247
Link control Style tab 247
MenuItem component Style tab 249
Prompt control 250
Prompt control Style tab 251
RadioGroup control Style tab 252
Spin control 254
Spin control Style tab 254
Static control Style tab 258
SubWindow control Style tab 260
Table control Style tab 262
Row and Column Properties dialog 265
Tabs control Style tab 267
Window control Style tab 268
TextBox control 270
TextBox control Style tab 271
Masking dialog box 273
Format dialog box for Check Box 278
Adding new values 279
Format dialog box for Combo Box and Radio Group 281
Text Format Definition dialog box 282
List of Dictionaries dialog box 283
Word Dictionary dialog box 284
A menu bar 288
Structure of a pull-down menu 288
Subapplication Menu Editor dialog box 289
MenuItem Component dialog box 291
Floating Menus manager 298

Floating Menu Editor dialog box 299
Assign Floating Menu Triggers dialog box 301
Font and Color dialog box 310
Text/Background color combo box 311
Color dialog box 311
Use color table check box 314
Color Table dialog box 315
Font and Color dialog box 318
Absolute pixels 319
Dialog units 320
Smart tabbing order mechanism 324
Window Algorithms tab 325
Setting Tabbing order in design view 326
Tabable controls 326
Not tabable controls 327
Tab stop checkbox in the Style tab 327
Tabbing order in subwindows 332
Enter method 338
SelectMenuOption method 339
Flow of events and system-triggered methods 341
User-Triggered Methods dialog box 349
System-Triggered Methods dialog box 350
UserRMB methods in System-Triggered Methods dialog box 352
Events tab of a component dialog box 356
Linking methods to controls in the KnowledgeBase 358
New Representation Component dialog box 358
Define Method dialog box 359
Comment method line type 369
Expression method line type dialog box 371
Variable Lookup dialog box 372
HostType method line type dialog box 373
MsgBox method line type dialog box 376
Return method line type dialog box 379
Return: Move Return method line type dialog box 380
Update method line type dialog box 381
Add Variable Name to List dialog box 382
DoMethod: Method Activation dialog box 385
What happens when the receiver is a variable 388
Inserting a reference to another method line 389
Assigning DoMethod parameters 390
Method Parameters dialog box 390
Define Commonly Used Methods dialog box 392
Enter method in the Define Method dialog box 393
UserAcceptScreen method 409
Runtime Screen Identification view 465
Runtime Screen Identification view menu options 466
Schematic illustration of how a screen may look in Runtime ID view 468
RT Identification dialog box 472

Fingerprint Pattern Definition for Runtime dialog box 473
Undo from the toolbar 474
Undo from the edit menu 474
Redo from the toolbar. 475
Redo from the Edit menu 475
Automatic Identification Definition dialog box 476
To capture a host screen image during runtime 480
Select Next Screen dialog box 481
Runtime Field Information view 484
Runtime Field Information menu options 487
Decomposition Definition Properties dialog box 488
Batch process dialog box 501
Extract from a batch file log. 501
Extract from a Subapplication log file 502
Log Options dialog box 503
Query dialog box 507
Query by option 507
List of queried conditions. 508
Save Subset As dialog box 509
Open Subset dialog box 509
Diagram of the compilation process. 512
Subapplications to Process option 513
Subapplication Subsets dialog box. 514
Generating the Runtime dialog box 516
Diagram of connection between host, emulator and GUI 520

List of Tables

JIS Interface Server documentation set 26
Documentation conventions 27
Screen definition file types 30
Filter sections derived from BMS files 106
Filter sections derived from the iSeries 107
The Layout menu 108
Primary/Filter Section menu options 109
Shortcut keys in Analysis View 116
Display Criteria Setup window fields 131
Shortcut keys available in Design View 193
Example of spin control values from DDS 257
Field token place holders for masking option 275
Field token place holders for textbox controls 276
Table of windows system elements 312
Activated in every application and subapplication 342
Activated under specific conditions 344
General user-triggered methods 346
General UserRMB method implementation 353
User- versus System-triggered methods 354
Method editing options 361
Method Line types 366
Return values 377
DoMethod: Method Activation dialog box elements 386
Parameter types 391
Table of host keys 401
Categories for system-triggered methods in this section 408
DoMethods 430

List of Examples

View attached functionality . . . 54
String type pattern definitions . . . 57
How ACE applies pattern definitions to the host screen . . . 59
The location parameter . . . 61
Iteration type pattern definitions . . . 69
Regular pattern definitions and dynamic screens . . . 74
Using layouts . . . 86
Pattern definition priority within a section . . . 100
Establishing the order of pattern definitions . . . 101
Filter section operations . . . 104
The coloring scheme . . . 113
Ignoring a pattern definition . . . 117
Setting an example . . . 119
Displaying a pattern definition's parent patterns . . . 128
Selecting a pattern definition by name . . . 129
Using display criteria to filter the pattern definition pane . . . 130
Display criteria . . . 132
Top level and child pattern definitions . . . 138
Editing child patterns . . . 139
Drag and drop replacement of pattern definitions . . . 140
Drag and drop a pattern definition as a sibling . . . 142
Drag and drop a pattern definition as a child pattern . . . 143
Adding child patterns to a new pattern definition . . . 146
Character Set parameters tab . . . 149
Horizontal group . . . 150
Vertical group . . . 155
Single representation definitions . . . 163
Multiple representation definitions . . . 164
Highlighting overlapping controls . . . 170
Leading controls . . . 176
Adjusting a control's size to it's text . . . 179
Component properties pane . . . 186
Table component properties . . . 188
Representation definitions . . . 199
Defining representations for lower level pattern definitions . . . 201
Components of the subapplication . . . 202
Relative placement . . . 209
Offset and size . . . 210
Expanding the tree . . . 214
Runtime data flow . . . 216
Explicit variable name . . . 220
Special effects . . . 230
Automatically assigning hotkeys . . . 295

Creating menus automatically 304
Creating automatic menu items 305
Color table settings 316
Methods 336
Method types and method levels 337
General system-triggered methods 347
Current subapplication and current library methods 348
DoMethod parameters 385
Referencing one method line from another 394
Fixed and variable characters 469
The screen fingerprint 471
Automatic identification definitions 477
Comparing two screen images 481
Variable patterns 485
The runtime field information screen 486
The color-coding scheme 487
Modifying the range of a pattern definitions' location I. 490
Modifying the range of a pattern definitions' location II 491
Modifying the range of a pattern definition's dimensions I 492
Modifying the range of a pattern definitions' dimensions II 492
Modifying the range of a pattern definitions' dimensions III. 493
Modifying a decomposition 494
Using subsets 498

About this Guide

ACE, the Automated Conversion Environment, automatically generates a Graphical User Interface (GUI) for legacy applications run on AS/400, S/3X and IBM mainframes.

ACE converts host screens using its KnowledgeBase. JIS 9.3 stores application and library metadata in dbm.sqlite.db. Eligible BDE databases are migrated on access or unpacking, and source BDE data is retained after successful conversion. Back up applications and libraries before upgrading. Install JDK 21 and a signed XML license, then regenerate the runtime and recompile extensions as described in the JIS 9.3 Release Notes.

The manual is divided into chapters. Blocks of chapters deal with the following issues:

- *Creating Screen Images*

  You begin the conversion by creating screen images from each screen of your legacy application. Where screen definition information is unavailable, you can create the panels by capturing host screens during an on-line session. Here you will find a discussion about the way to generate Screen Images from either Screen Captures or Screen Definition Files (for AS/400 and mainframe applications).
- *Fine Tuning the GUI in the ACE Views*

  You fine tune the automatic GUI using the tools available in the different ACE Views. Some Views show the host screen and some Views show the generated GUI. The order in which you work is dependent on the information you wish to gather or the operations you wish to perform.

ACE is a Windows-based product. This manual assumes that the reader has some basic knowledge of the Windows conventions, and the way Windows operates.

Documentation Set

JIS Interface Server is supplied with the manuals shown below. The documentation is delivered in Adobe Acrobat Reader Portable Document Format (PDF). No hardcopy documentation is provided, but you can print the PDF files on your local printer.

Table 1. JIS Interface Server documentation set

This book...	Contains...
JIS Interface Server: Getting Started with the Automated Conversion Environment	Startup information and an introduction to the Automated Conversion Environment (ACE).
JIS Interface Server: Basic User's Guide	Full explanations of the ACE Views and how to use them
JIS Interface Server: Advanced Topics	Explanations of advanced features that give your application extra functionality.
JIS Interface Server: KnowledgeBase User's Guide	In-depth information about the way the ACE KnowledgeBase is designed and how to work with it.
JIS Interface Server: Java Client User's Guide	Information for migrating your host application to Java.
JIS Interface Server: XHTML Client User's Guide	Information for migrating your host application to an XHTML web application.

Document Conventions

The following conventions are used throughout this manual.

Table 2. Documentation conventions

Convention	Description
Click	Position the mouse pointer on the control and quickly press and release the left mouse button **once**. (Unless the right mouse button is explicitly specified, you should click the left mouse button.)
Double-click	Position the mouse pointer on the control and quickly press and release the left mouse button **twice**. (Unless the right mouse button is explicitly specified, you should double-click the left mouse button.)
UPPERCASE	Uppercase letters are used for the names of files. For example, a panel file with the name Menu, will be expressed as MENU.PNL**.**
italics	Names of applications, programs, menus, dialog boxes, and libraries.
Bold	Menu options, and items, dialog boxes and items to be selected from a dialog box. The names of pull-down menus.
Bold Italics	Pattern definitions, representation definitions, message definitions, method names, layout names, section names, selection definitions, function definitions.
BOLD + UPPERCASE	Keyboard shortcuts: Press the **SHIFT** key. Press **CTRL + Z**.

Viewing the Documentation Online

You can also access the latest version of the documentation for Software GmbH products at http://documentation.softwareag.com/. As new versions become available, the documentation on this web site will be updated and the previous versions will be migrated to the Software GmbHdocumentation web site at http://servline24.softwareag.com/public/. If you have a maintenance contract, you can view all versions of documentation on this web site. You will find instructions for registering and obtaining a userid and password on the documentation web site.

Chapter 1. Creating Screen Images

In order to convert your host application you must first create screen images for each of the host application screens. Screen images (.pnl files) can be created by using one or both of two basic means: *Screen definition files* or *screen captures*.

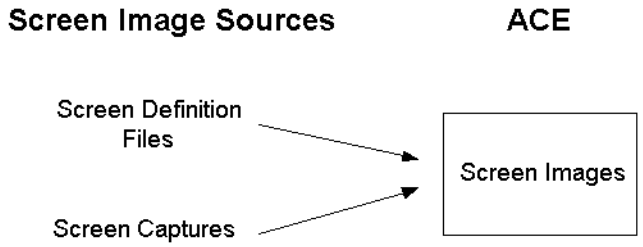


Figure 1. Screen image sources

ACE can successfully exploit information contained in screen definition files to create screen images that include information about the position and content of screen elements, the communication between the host screens and the host, as well as transitions from one screen to another. The depth of information provided by screen definition files makes their use the preferred means of creating screen images.

This chapter describes:

- Creating Screen Images Using Screen Definition Files
- AS/400 Applications and Screen Definition Files
- Mainframe Applications and Screen Definition Files
- Capturing Your Host Application
- Combining a Screen Capture and a BMS or MFS File
- Maintaining Screen Images
- Editing Screen Images

Creating Screen Images Using Screen Definition Files

ACE can convert AS/400 and mainframe applications that were developed using screen definition files. Screen Definition File is a generic term for a group of file formats that store information which defines and describes host application screens. BMS (Basic Mapping Support), MFS (Message Format Service) and DDS (Data Description Specification) are examples of screen definition file formats.

These files usually contain more information than a simple screen capture and therefore can be used to ease and enhance the creation of a GUI interface for your application.

Working with screen definition files has the following advantages:

- In many cases there is no need to obtain screen captures before starting the conversion.
- Screen definition files generate screen images with all the attribute characters needed to facilitate the use of Pattern Definitions.
- Screen definition files display fields that were not visible at the time a screen capture was taken.
- The use of filters greatly improve analysis.
- Screen definition files use meaningful field names.
- Screen definition files include information useful for identifying screens in runtime.
- Screen definition files generate separate filter sections for consecutive output fields even when the output attribute between the two fields is missing (mainframe only).
- Screen definition files use filter sections to determine the exact length of a field, when the output attribute that terminates the field is missing (mainframe only).

Screen Definition File Types

Table 3 lists the describes the different screen definition file types.

Table 3. Screen definition file types (Sheet 1 of 2)

File Type	Description
MFS	These files contain the source code that defines the screen as transferred from the mainframe. The source code comprises hierarchically arranged units called Mapsets, Maps and Fields.
BMS	These files contain the source code that defines the screen as transferred from the mainframe. The source code comprises hierarchically arranged units called Mapsets, Maps and Fields.

Table 3. Screen definition file types (Sheet 2 of 2)

File Type	Description
SDF	Processed BMS or MFS files. They contain all the useful information that was extracted from the source screen definition files. ACE uses the information in this form to create SDI and PNL files.
SDI	These files contain all of the BMS information to be used by ACE in the conversion process for a single screen image.
IND	These files provide ACE with the ability to find screen images that were created using a given Map or Mapset in the **Create Screen Images** wizard. They contain references from each map to the screen images that were created using the Map.
DDS	These files contain the source code that defines the screen as transferred from the AS/400.
DDO	DDS files converted into a format that can be used by ACE. ACE uses this format to create DDI and PNL files.
DDI	These files contain all of the DDO information to be used by ACE for creating a single screen image.
PNL (Panel)	The image of a particular screen. PNL files are either screen captures combined with BMS screen images, files generated by BMS support, or files generated by DDO support.

Note: For each Mapset one BMS/MFS, SDF, and IND file is produced. For each screen image one SDI and one PNL file is produced.

AS/400 Applications

If your application was developed with a screen definition file format such as DDS then refer to "AS/400 Applications and Screen Definition Files" on page 32 for further details on how to work with screen definition files.

Mainframe Applications

If your application was developed with a screen definition file format such as BMS or MFS then you should refer to “Mainframe Applications and Screen Definition Files” on page 38 for further details on how to work with screen definition files.

AS/400 Applications and Screen Definition Files

DDS (Data Description Specification) files are AS/400 files that describe the screens of the application. AS/400 applications that have been developed with DDS have special files that contain the DDS screen information. Code is added to the program that loads the DDS object in runtime.

The DDS information is the precise information needed for understanding the screens. When this information is retrieved it can be used effectively by ACE to create a GUI interface for your application. Thus the DDS files can be an important tool for migrating AS/400 applications to GUI.

ACE can convert screens with or without using DDS. This section describes the ACE conversion process with DDS.

The Recommended Workflow

The ACE DDS Compiler processes the DDS files and creates DDO files that can be used by ACE. The first stage in using ACE with DDS is to install the Compiler and then compile the DDS files. Then you copy the compiled files to your PC, and process them in ACE.

The following diagram illustrates the flow of the DDS files from the AS/400 to the PC. The ACE DDS Compiler processes the DDS source files into DDO files and places them in a shared folder on the AS/400. You then copy the DDO files to a temporary directory on the PC for processing in ACE.

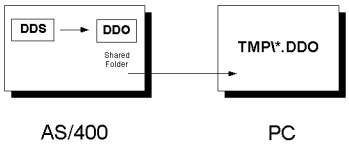


Figure 2. DDS compiler workflow from iSeries to PC

Within ACE, you invoke the *Create Screen Images* wizard to convert the DDO files into screen images—files of DDI and PNL format. ACE generates a GUI from each screen image and then combines the GUIs into an executable file. The PNL files are used with the File emulator to test your executable.

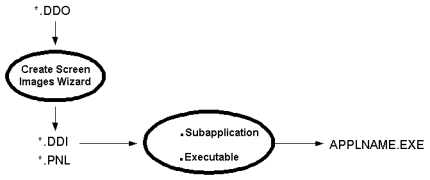


Figure 3. DDS compiler workflow in ACE

The following is the workflow for converting DDS files:

1. Install the ACE DDS Compiler on your AS/400.
2. Use the DDS Compiler to compile the DDS members on the AS/400. The Compiler generates DDO files and saves them in a shared folder.
3. Copy the DDO files from the AS/400 to a temporary directory on the PC.
4. Open ACE and create a new Application.
5. Create screen images. The next section details this process.

> **Note:** This process should be performed by a person well-versed in the host application.

6. Create a Subapplication for each screen image.

For more details, see *JIS Interface Server: Getting Started with the Automated Conversion Environment*.

At this point you have a GUI for each screen image. See the following chapters on ACE's Views for a description of how you can fine-tune the GUIs.

Screen Model Type

The AS/400 display terminal (model 5250 or its emulation) can be of one of two sizes or screen types:

- Model 5250 Type 2 (24x80)
- Model 5250 Type 5 (27x132)

It is important that a library designated for one Screen Model Type contains only Subapplications that represent host screens of that one designated type.

Screen Model Type 2

This host screen model type is the most common. Host screens of this type are comprised of 24 columns and 80 rows. Unless you are working with libraries, ACE automatically assumes that you are working with Model Type 2 screens.

Screen Model Type 5

Model 5250 Type 5 host screens have 27 lines and 132 columns. Type 5 host screens are not as common as Type 2. In most cases, a host application containing Type 5 screens also contains Model type 2 screens. In ACE, the screens definitions for each model type must be housed in their own library.

Note: When creating new Model Type 5 Subapplications in the *New Subapplication* wizard, the default selection in the *Select Screen Layout* step is not suitable for this screen type. Select the *Basic* screen layout option for a more suitable layout.

Using Libraries to Contain Different Screen Model Types

When converting a host application that contains Model Type 5 screens, you must create a library specifically for these screens. You create a library using the *New Library wizard*. In the second step of the *New Library wizard*, you are asked to select the Screen Model Type.

The wizard offers two choices:

- Model 2 (24x80)
- Model 5 (27x132)

This is seen in the following image:

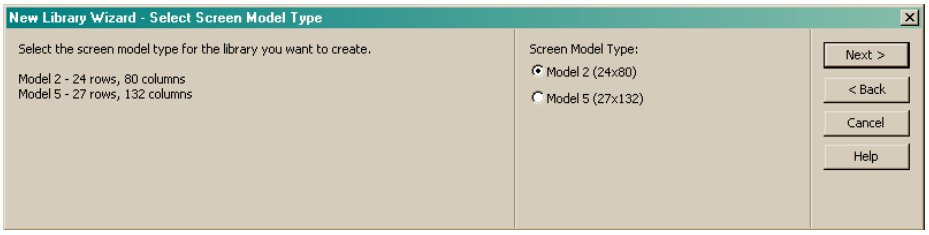


Figure 4. Setting the screen model type

Note: It is important that a library designated for one Screen Model Type will contain only subapplications that represent host screen of that one designated type.

Creating Screen Images

From the *File* menu select *Create Screen Images*. ACE starts the *Create Screen Images* wizard.

The wizard prompts you for the information you need to create screen images from DDO files. You will need to know the name and path to the temporary directory where you stored the DDO files.

The Create Screen Images From Compiled DDS Dialog Box

Within the *Create Screen Images* wizard you will need to build screen images from the records contained in the DDO files. You do this in the *Create Screen Images - From compiled DDS* step:

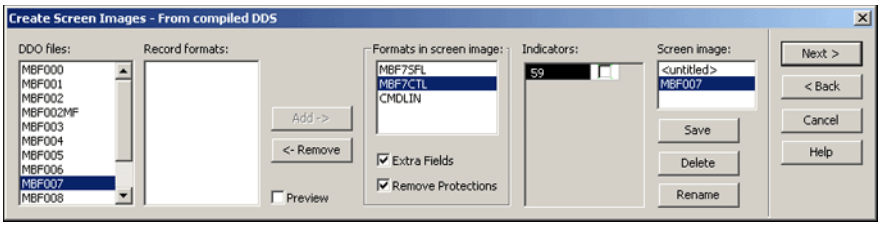


Figure 5. Create Screen Images from Compiled DDS step in wizard

To create a new screen image or edit an existing screen image:

1. Create a new screen image. From the *DDO files* list select the DDO file from which you wish to define a new screen image, and select <untitled> from the *Screen image* list. Each of the DDO file's records appear in either the *Formats in screen image* list or the *Record formats* list.

   -or-

2. Edit an already defined screen image. Select the defined screen image's name from the *Screen image* list. Each of the DDO file's records appear in either the *Formats in screen image* list or the *Record formats* list box.
3. To *include* record formats: From the *Record formats* list select the record formats to be included and click *Add* to transfer them to the *Formats in screen image* list. The record format name is removed from the *Record formats* list and now appears in the *Formats in screen image* list. The screen image displayed in the *Host Session* window is updated to include the added record format.
4. To *remove* record formats: From the *Formats in screen image* list select the record formats to be removed and click *Remove* to return them to the *Record Formats* list. The record format name is removed from the *Formats in screen image* list and now appears in the *Record formats* list. The screen image displayed in the *Host Session* window is updated to exclude the removed record format

5. From the *Formats in Screen Image* list select each format, one at a time, and configure the indicators associated with the record format in the *Indicators* list, where applicable.
6. Click the *Save* button. If you are working on a new screen image, <untitled>, you are prompted to provide the new screen image with a name.

Note: The defined screen images are not actually created until you complete the *Create New Screen Images* wizard.

Features of the Dialog Box

This section describes the Create Screen Images features that are accessed through the Compiled DDS dialog box.

DDO Files

The DDO files list contains the names of all the DDO files that are available for processing. These files form the raw material that you use to create screen images. When you select a DDO file from the *DDO files* list, all the screen images already defined from the selected DDO file, as well as <untitled> for defining new screen images, appear in the *Screen image* list.

Record Formats and Formats in Screen Image

DDO files are composed of smaller units called record formats. A DDO file may contain a single record format or a number of record formats.

- The record formats used in the defined screen image appear in the *Formats in screen image* list. The image created from these records appears in the *Host Session* window.
- The record formats not used in the defined screen image appear in the *Record formats* list.

Note: Selecting <untitled> in the *Screen Image* list begins the definition of a new screen image. At this point you have not yet added any record formats to the new screen image, so all of the selected DDO file's record formats appear in the *Record formats* list.

Indicators

Some record formats have multiple versions, controlled by indicators. When you select such a record from the *Formats in screen image* list, the *Indicators* box lists all the different indicators associated with the record.

Enable or disable an indicator by setting or clearing the check box next to it. The screen image displayed in the *Host Session* window updates to reflect the indicator settings.

Screen Image

The *Screen image* list contains the names of all the screen images defined from the DDO file selected in the *DDO files* list. To change an existing defined image, select the defined image in the *Screen images* list. To define a new image from the selected DDO file select <untitled> in the *Screen images* list.

The selected image appears in the *Host Session* window.

Preview

The *Host Session* window displays the screen image defined by the record formats included in the *Formats in screen image* list and the *Indicator* settings. You can preview the effect of transferring a record from the *Record formats* list to the *Formats in screen images* list by setting the *Preview* check box.

When the *Preview* checkbooks is set, the screen image displayed in the *Host Session* window includes the selected record in the *Record formats* list.

The *Preview* feature allows you to quickly see each individual record format's effect on the screen image, since it is faster to select records in the *Record formats* list than it is to transfer records back and forth between the *Record formats* list and the *Formats in screen images* list.

Save

Click the *Save* button to save the selected configuration of records and indicators as a screen image.

- If you are working on a new screen image—<untitled> is selected in the *Screen image* list—you are prompted to give a name to the new screen image. The new name appears in the *Screen image* list.
- If you are working on a named screen image—the name is selected in the *Screen image* list—then the new image "overwrites" the old image.

Delete

Removes the selected screen image from the Screen image list and deletes it from the Application.

Rename

Click *Rename* to give a new name to the selected screen image. The new name appears in the *Screen image* list.

Handling Protected and Hidden Fields

Certain screen images represent two or more screens. If screens generated from a single screen image are used in an Application, ACE must be presented with all the necessary information regarding these screens in order to convert them properly. This entails displaying all extra fields, as well as protected fields.

Figure 6. Displaying extra fields and protected fields

Extra Fields

The *Extra Fields* check box refers to fields which are either not shown because of indicators or are not initially displayed. These fields are hidden or withheld from a screen. When the *Extra Fields* check box is set, the hidden property of all fields is overridden, and all the fields are displayed.

The display of all fields is necessary for the conversion in ACE, even if not all fields can be displayed when the Application is running. In runtime, ACE refers to the host screen to determine which fields will be presented in the individual screens.

If colors or other keywords are conditioned by indicators that are currently switched off, they are not shown or "forced" by setting the *Extra Fields* check box. The *Extra Fields* check box only causes hidden fields to appear.

Remove Protections

The *Remove Protections* check box refers to input fields which are considered "protected". Protected fields are fields that have received a keyword DSPATR (PR) which indicates that the specified field may not receive input. When the *Remove Protections* check box is set, all protected input fields are displayed as plain input fields, even if this cannot happen when the application is running.

In runtime, ACE refers to the host screen to determine which fields are protected in the individual screens.

Mainframe Applications and Screen Definition Files

BMS (Basic Mapping Support) and MFS (Message Format Service) are two forms of mainframe files that define application screens. They are types of screen definition files, and the information contained in them can be processed and then used by ACE. When this information is retrieved it can be used effectively by ACE to create a GUI interface for your application. SDF information can be an important tool for migrating mainframe applications to GUI.

ACE can convert screens with or without using SDFs. The following describes the process of creating screen images from BMS or MFS files.

The Recommended Workflow

To create screen images from BMS or MFS files:

1. Copy the BMS or MFS files from the mainframe as text files to a temporary directory on the PC. The files should be saved with a BMS or MFS extension, as appropriate.
2. Create a new ACE Application or open an existing Application. For more details, see *JIS Interface Server: Getting Started with the Automated Conversion Environment*.
3. Create the screen images.

Screen Model Type

The S/390-type mainframe display terminal (model 3270 or its emulation) can be of several different sizes or screen types:

- Model 3270 Type 2 (24x80)
- Model 3270 Type 3 (32x80)
- Model 3270 Type 4 (43x80)
- Model 3270 Type 5 (27x132)

It is important that a library designated for one Screen Model Type contains only Subapplications that represent host screens of that one designated type.

Screen Model Type 2

This host screen model type is the most common. Host screens of this type are comprised of 24 columns and 80 rows. Unless you are working with libraries, ACE automatically assumes that you are working with Model Type 2 screens.

Screen Model Types 3, 4, and 5

Model 3270 Type 3 host screens have 32 lines of 80 columns. Type 4 host screens have 43 lines and 80 columns. Type 5 host screens have 27 lines and 132 columns.

Type 3, 4, and 5 host screens are not as common as Type 2. In most cases, a host application containing Type 3, 4, or 5 screens also contains Type 2 screens. In ACE, the screens definitions for each model type must be housed in their own library.

Note: When creating new Model Type 3, 4, or 5 Subapplications in the *New Subapplication* wizard, the default selection in the *Select Screen Layout* step is not suitable for this screen type. Select the *Basic* screen layout option for a more suitable layout.

Using Libraries to Contain Different Screen Model Types

When converting a mainframe host application that contains Model 3270 Type 3, 4, or 5 screens, you must create a library specifically for these screens. You create a library using the *New Library wizard*. In the second step of the *New Library wizard*, you are asked to select the Screen Model Type.

The wizard offers several choices:

- Model 3270 Type 2 (24x80)
- Model 3270 Type 3 (32x80)
- Model 3270 Type 4 (43x80)
- Model 3270 Type 5 (27x132)

This is seen in Figure 7:

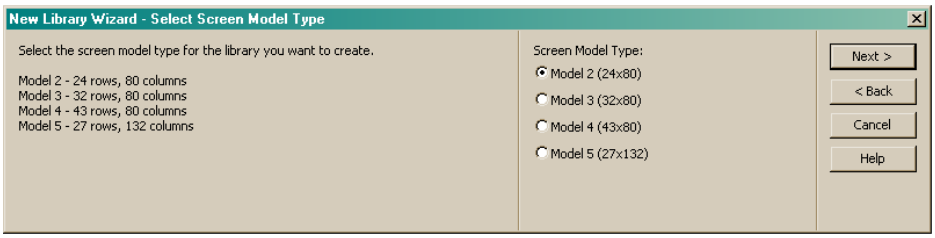


Figure 7. Choices for setting the model type

Creating Screen Images

From the *File* menu select *Create Screen Images*. ACE starts the *Create Screen Images* wizard.

The wizard prompts you for the information you need to create screen images from BMS or MFS files. You will need to know the name and path to the temporary directory where you stored the BMS or MFS files.

Screen Image Name Source

The wizard asks you which parameter you want ACE to use as a screen image name. You can select one of three options:

- Map name
- Mapset name
- Field on host screen

> **Note:** When *Field on host screen* is selected, the content of a specific field is used to name the screen image. Specify the X and Y coordinates of the field to be used. The X and Y coordinates start from zero.

If a second screen image has the same name as a previously created screen image, ACE names the new screen image by adding a character to the name. In cases where the name is too long, a character is removed from the name.

Capturing Your Host Application

Where an application was developed without the use of screen definition files or where individual host screens within an application lack screen definition files, you can create screen images by capturing screens online from the host application using ACE's screen capture facility. You will need to have a live connection to the host, either through TCP/IP, a router, or other channels. You may also need to know the address of your host.

Creating Screen Images From Screen Captures

To create screen images from screen captures:

1. Establish an Application in ACE. For more details, see *JIS Interface Server: Getting Started with the Automated Conversion Environment*.
2. From the *File* menu select *Create Screen Images*. ACE starts the *Create Screen Images* wizard.

The wizard prompts you for the information you need to create screen images from screen captures.

Deleting Captured Screen Images

The Save Images step in the Create Screen Image wizard includes a Delete button. The button is enabled when an image is selected from the list of captured screen images.

Combining a Screen Capture and a BMS or MFS File

The host application has the ability to modify the content of fields and their attributes, in addition to the existing definitions in a BMS or MFS file. In some cases the screen image created by the BMS support in ACE, may lack important information added by the host application.

This situation might occur when an output list is filled and divided into columns at runtime, while the BMS or MFS file just defines empty lines without any division into columns whatsoever. Without the screen capture ACE is missing some important information. Such a situation can be rectified by combining the screen capture with the BMS or MFS derived screen image.

You can combine the screen image with a screen capture by using the *Edit Screen Image* wizard available in the *Host* menu in Host View.

You will need to:

1. Perform the screen capture.
2. Open the BMS or MFS screen image's Subapplication in ACE.
3. Invoke the *Edit Screen Image* wizard to combine the screen capture with the BMS or MFS file. You will need to know the path to the captured screen.

Maintaining Screen Images

When the host screens change as a result of development of the host application, you can update your screen images to reflect these changes. You may have fine-tuned the Subapplication created from the original image. The Maintain Screen Images process preserves fine tuning that is connected to unchanged parts of the host screen.

AS/400 users must recompile the altered DDS files to create new DDO files, using the DDS Compiler. Once the files have been recompiled you can run the *Maintain Screen Images* wizard to update the Subapplications. You will need to know the name and path to the temporary directory where you stored the updated DDOs (for AS/400 users) or updated BMS or MFS files (for mainframe users).

To maintain screen images:

From the *File* menu select *Maintain Screen Images*. ACE starts the *Maintain Screen Images* wizard.

The operation of the wizard is essentially the same as for creating screen images.

Note: For AS/400 users: If new fields were added to the DDS file, then the *Extra Fields* check box should be cleared and then set again in the *Create Screen Images - From compiled DDS* dialog box of the wizard. This re-evaluates and recalculates the fields.

Editing Screen Images

The Screen Image Editor provides a tool for editing or creating PNL (screen image) files that are then used to combine screens or for file testing.

The Screen Image Editor is generally used to solve unique situations when encountering screen identification failures during runtime testing.

Runtime Identification anomalies can also be solved by changes made to INI files. Changes made to the INI files are globally applied to all Subapplications in a given Application. Changes made using the Screen Image Editor are applied only to the edited PNL file and its Subapplication.

The Screen Image Editor is used for the following activities:

- Changing text
- Changing the foreground and/or the background color
- Changing attributes for the attribute properties and/or the field type
- Changing the default cursor location on the host screen

Examples for using the Screen Image Editor:

- Use the Screen Image Editor to create independent screen images and save under new names. For instance, you can create an empty screen for use as a principal Subapplication when working with Many-to-One.
- Change the controls in table columns. For instance, you can change an adjustable edit field to a combo box or a check box by changing the attributes.

Accessing the Screen Image Editor

The *Screen Image Editor Properties* dialog box is used to edit PNL files. You can access the Screen Image Editor via the *Edit Screen Image* wizard from either the *Host* or the *Utility* menu.

Via the Host Menu

The *Edit Screen Image* option is only available via the *Host* menu when the source is a captured screen. The screen must be open in ACE.

To access the Screen Image Editor, from the *Host* menu, select *Edit Screen Image*. The *Edit Screen Images* wizard opens.

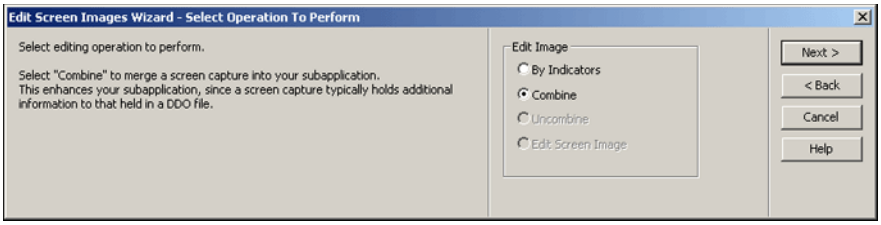


Figure 8. Edit Screen Images wizard

Pressing *Next* in the wizard step shown above opens the Screen Image Editor.

After editing, the PNL file is saved and the corrected image is automatically applied to the Subapplication.

Via the Utility Menu

The Screen Image Editor is also available in the *Utility* menu. From the *Utility* menu, select *Screen Image Editor*. The *Screen Image Editor* wizard opens.

Here the Screen Image Editor can be applied to all pnl files regardless of their source. The *Select Screen Image* step in the Screen Image Editor allows selecting any pnl file appearing in the appropriate directories.

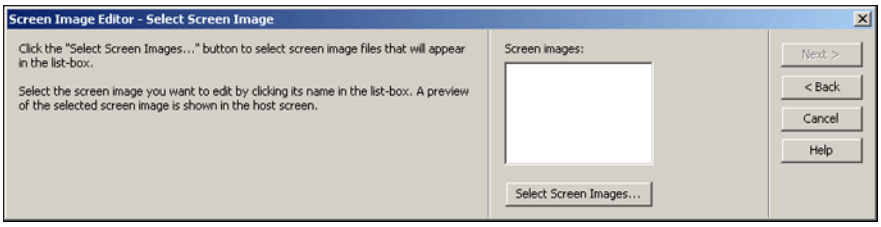


Figure 9. Screen Image Editor wizard via the Utility menu

Since the editing is not carried out on screen images, the modifications are not applied to existing Subapplications.

Using the Screen Image Editor

The *Screen Image Editor Properties* dialog box is automatically displayed after navigating to the *Edit Screen Image* step in the *Screen Image Editor* wizard.

Moving the Cursor

When the Screen Image Editor is open, the cursor blinks in the host screen. The editable place is at the cursor position.

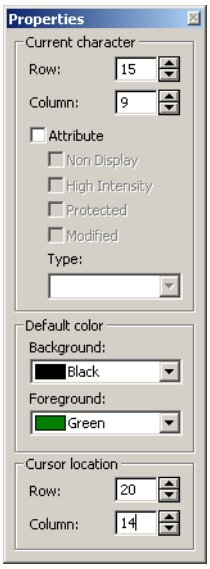


Figure 10. Properties panel

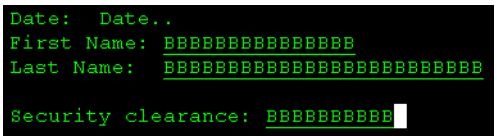


Figure 11. Editable place on the screen is where cursor is blinking

The cursor can be moved from its current position by:

- Pointing and left clicking on a new position
- Using the *Current Character* spin control in the *Properties* dialog box
- Using the keyboard:

Arrow keys	Move the cursor right/left and up/down. When reaching the end of a line the cursor jumps to the beginning of the next line. When reaching the end of the screen the cursor jumps to the first position on the screen.
End/Home	**End** jumps the cursor to the end of the line (last column). **Home** jumps the cursor to the beginning of the line (first column).

PageUp/ PageDown	The cursor jumps to the first or last line.
Backspace	Delete the previous character.
Tab	Jump the cursor to the next editable field.

RMB Functionality

When right clicking on the host session, this shortcut menu is displayed:

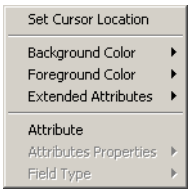


Figure 12. RMB host session options

Changes set using the RMB shortcut menu take effect at the current cursor position.

Changing the Default Cursor Location

Sometimes the cursor is not present in pop-ups captured in runtime. The cursor helps identify the popup border. In this case, it is necessary to set the default cursor location.

Using the *Properties* dialog box:

1. Select the new default location using the *Cursor Location* spin controls.
2. Press *Save* in the *Edit Screen Image* wizard step. The new location is visible only after exiting and re-entering the Screen Image Editor.

Using the RMB shortcut menu:

1. Move the cursor to the new position by using the arrow keys or by pointing and clicking.
2. With the pointer on the cursor, right click.
3. Select *Set Cursor Location* from the shortcut menu. The *Cursor Location* spin control is updated and displays the new location.

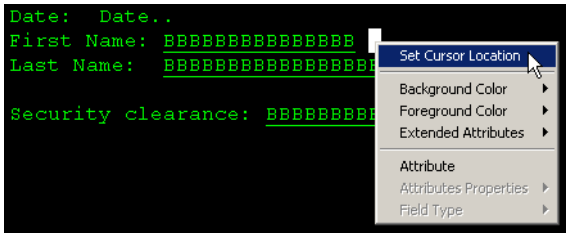


Figure 13. Set cursor location

Changing colors

Changing colors is generally used for messaging. By default, messages appear in white on a yellow background.

Using the *Properties* dialog box:

1. Point and click or use the arrow keys to place the cursor at the target position.
2. Select a color from the *Background/Foreground* color combo boxes. The text typed at the cursor position is displayed in the selected colors.

Using the RMB shortcut menu:

1. Move the cursor to the target position by using the arrow keys or pointing and clicking.
2. With the pointer on the cursor, right click.
3. Select *Background Color* or *Foreground Color* from the shortcut menu. A list of color selections is displayed (see figure below).
4. Click on the desired color. The new color appears on the host screen.

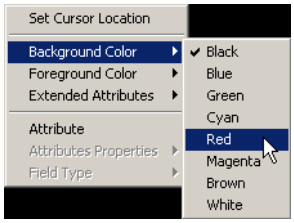


Figure 14. Selecting the desired color

Note: Color options are disabled when the cursor is positioned on an attribute.

Using frames:

1 Position the pointer at the desired position or next to target text.
2 Left click and drag a frame around the area or text.
3 Right click in the frame. Select *Paint Frame* from the shortcut menu.The *Paint Frame* dialog box is displayed:

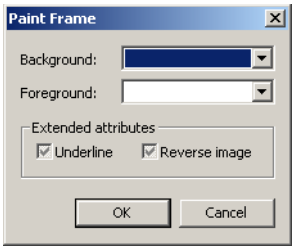


Figure 15. Paint frame

4 Change background/foreground colors using the color combo boxes. Extended attributes are also available in this dialog box.

Adding or Removing Extended Attributes

Extended Attributes include underline and reverse image of text characters. The *Extended Attributes* option is only available from the RMB shortcut menu:

1 Place the cursor on the character or on any position.
2 With the pointer on the cursor, right click and select *Extended Attributes* > *Underline* and/or *Reverse Image*. The underline appears on the host screen. Reverse Image is not visible.

   Extended Attributes are also available in the *Paint Frame* dialog box. See preceding section for details.

Note: Extended Attributes are not available when the cursor is positioned on an attribute.

Adding, Editing, and Deleting Attributes

An example of editing attributes is when the captured panel contains output, but input attributes may be necessary.

Using the *Properties* dialog box to add attributes:

1 Move the cursor to the target position by using the arrow keys or pointing and clicking.

2 Set the *Attribute* check box at the cursor position and select the *Attribute* property. The default is no attribute properties selected.
3 Select the attribute type from the *Type* combo box. The default is alphanumeric.

Using the RMB shortcut menu to add attributes:

1 Move the cursor to the target position by using the arrow keys or pointing and clicking.
2 Put the pointer on the cursor and right click.
3 Click on *Attribute* in the shortcut menu. *Attribute* is checked in the shortcut menu and the check box in the *Properties* dialog box is checked. The default is alphanumeric type with no attribute properties selected.

Using the *Properties* dialog box to edit or delete attributes:

1 Move the cursor and stand on the target attribute.
2 To edit an attribute, click on the *Attribute* property or select the attribute type from the *Type* list.

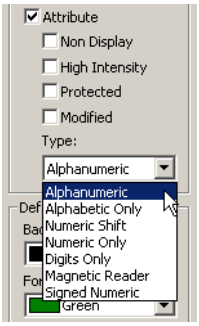


Figure 16. Editing attributes

3 To delete an attribute, stand on the attribute and clear the *Attribute* check box.

Using the RMB shortcut menu to edit or delete an attribute:

1 Place the cursor on the target attribute.
2 To edit and attribute, place the pointer on the cursor and right click. All attribute options are enabled. Selecting *Attribute Properties* or *Field Type* displays the options in cascaded menus. Select the desired option.
3 To delete an attribute, select *Attribute*. The selected attribute is deleted from the host screen and the *Properties* dialog box is updated.

Adding and Deleting Text

Text can be added by typing directly at the cursor position. Typing is always in overwrite mode. The text color and the background color can be altered. See "Changing colors" on page 47 for more details. Adding text might be used to edit dynamic options or leaders on mainframe screens.

To delete text, place the cursor to the right of the character and press *Backspace*. Deleting text might be used to delete a message that appeared on the screen when it was captured during runtime.

Chapter 2. Comparing the Host Screen and the GUI Window

This chapter discusses:

- Host View
- Test View

Host View and Test View allow you to see the host screen and the GUI window respectively.

In Host View you examine the contents of the screen image and verify that they are correct.

In Test View you see the way the window and all of its components will look in runtime. You can also see a description of the functionality of items such as push buttons and menu items. The purpose of Test View is to make sure you are satisfied with the window.

To verify that the window represents the screen image in the most suitable way, it is recommended to compare the window shown in Test View to the screen image displayed in Host View.

Host View

In Host View you see the screen image:

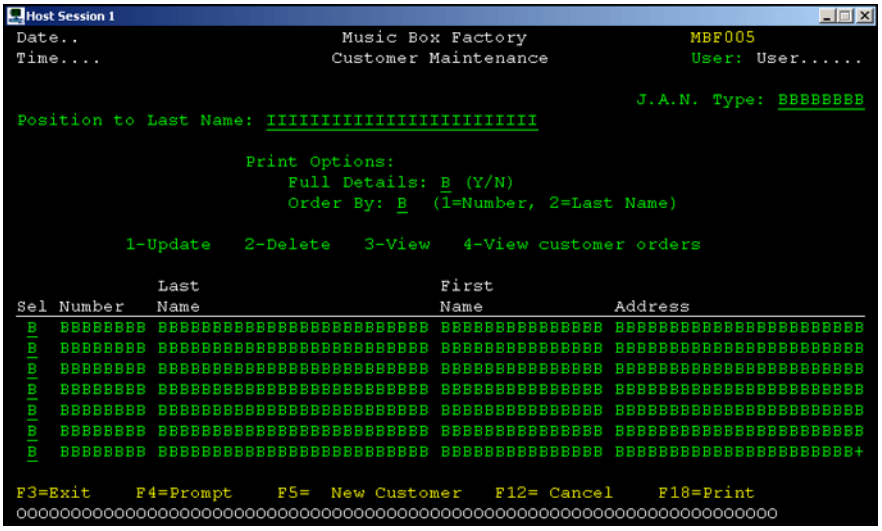


Figure 17. The screen image in host view

What You See in Host View

- *Screen image*

  Host View displays the original screen image.
- *Attributes*

  The screen image is displayed by default with attributes. The attributes help you understand the contents of the screen image, by defining a field's type and delimiting its size.

  OOOOOOs, BBBBBBs, IIIIIIIIIs

  In screen images created from some form of Screen Definition Files (such as DDS for AS/400 and BMS or MFS for mainframe), these characters represent different types of fields in the screen image:

  OOOOOOs represent Output.

  BBBBBBBs represent both Input and Output.

  IIIIIIIIIIIIIIIIs represent Input.

Note: BBBBBBs are more commonly used than IIIIIIIIIIs.

What you can do in Host View

This section describes Host view operations.

Edit an Incorrect Screen Image

The editing options enable you to correct the way the SDF (Screen Definition Files) information is used to create the screen image. You can also combine a screen image created from an SDF with a screen capture. This is necessary when the SDF screen image contains many output fields, and it is difficult to differentiate between the various fields.

For example, a list may be displayed as one block of outputs, in which it is not possible to differentiate between the columns. In a screen capture, however, the list contains real data. By combining the screen capture with the SDF screen image, the information in the resulting screen image is more complete.

To edit an incorrect screen image:

Select *Edit Screen Images* from the *Host* menu. This opens the Edit Screen Images wizard.

Display or Hide Attributes in Screen Views

It is recommended to work with attributes displayed.

To toggle attribute display:

Select *Host Screen Options* from the *Options* menu within any screen view.

Test View

The GUI representation of the screen image is displayed in Test View as follows:

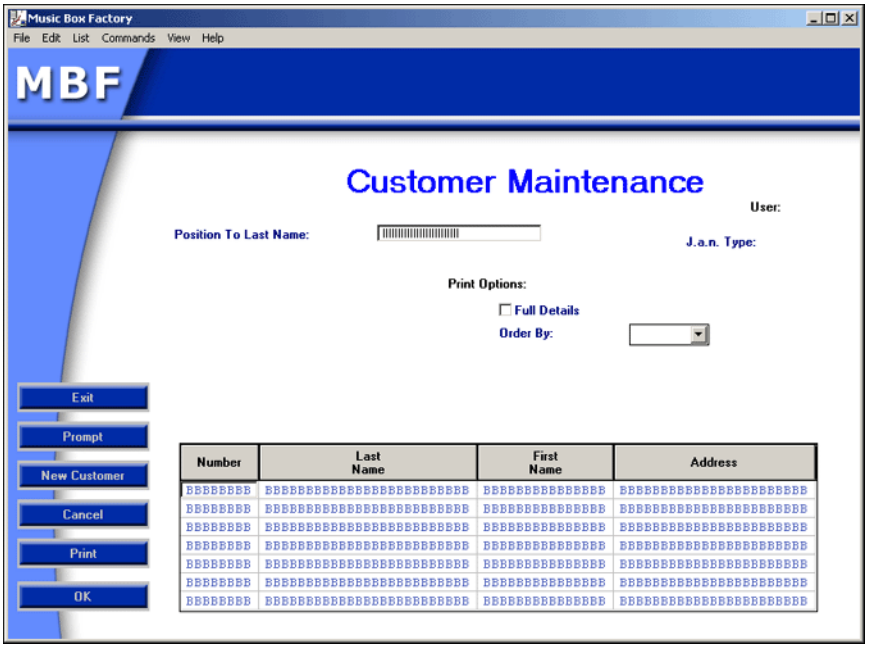


Figure 18. The GUI representation of the host screen

ACE enables you to compare the screen image to the GUI window by simultaneously activating a screen-related view together with a window-related view. Activate Host View and compare all the screen image components to the GUI controls that represent them. This will help you determine whether you need to make any changes to the Subapplication.

What You See in Test View

Test View displays the GUI as it will look in runtime. You can see, for example:

- The values in a combo box.
- The values on tabs.
- A description of the functionality attached to controls.

What You Can Do in Test View

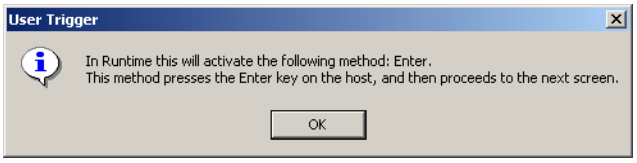
This section describes Test view operations.

View Attached Functionality

Click a push button, click a menu item, or double click a table to display information on the runtime functionality of the control.

Example 1. View attached functionality

Pressing the OK button, which is attached to the *Enter* method, prompts the following popup message:



Change Table Appearance

You can change the way a table looks. You can manipulate the order in which the different columns are placed, and you can also change the width of each column.

Chapter 3. How Pattern Definitions Work

This chapter describes:

- The Idea Behind Pattern Definitions
- Compound Pattern Definitions
- Dynamic Pattern Definitions

This chapter, and the following chapter Sections and Layouts, describe the objects ACE uses to analyze host screens. These are the objects you will modify in order to influence the analysis.

Related information:

- Step-by-step operating instructions for modifying Pattern Definitions are contained in the chapter Pattern Definition Modifications through the KnowledgeBase.
- Step-by-step operating instructions for modifying sections and layouts can be found in the chapter Layout View Operations.
- Conceptual information about how best to modify Pattern Definitions is the subject of *JIS Interface Server: KnowledgeBase User's Guide*.

The Idea Behind Pattern Definitions

Pattern Definitions are automatic rules for extracting meaning from the characters on host application screens. This chapter explains what these rules do and how simple rules combine together to form more sophisticated rules.

Pattern Definitions are the rules that ACE uses to read and understand host screens. Your understanding of Pattern Definitions begins with:

- What a Pattern Definition is
- Examples using string type Pattern Definitions
- How ACE applies Pattern Definitions to the host screen
- How ACE chooses which Pattern Definition to apply

The following sections explain these concepts by means of many examples using a single type of Pattern Definition.

Note: The example Pattern Definitions do not necessarily exist in the ACE KnowledgeBase.

What is a Pattern Definition?

A host screen displays characters—letters, digits, symbols and attributes—that individually have very little meaning. However, these characters are displayed in groups, which we call words, and these groups do contain meaning. Even more meaning is contained when words are combined into phrases, sentences and ever larger objects.

A Pattern Definition is a rule that decides when a group of characters form a word, a phrase or some other object. Each Pattern Definition recognizes some character groups as words or phrases, and ignores all the others. The total of all the Pattern Definitions is designed to recognize all the words and phrases on a host screen.

Each Pattern Definition is associated with a specific GUI element. When a Pattern Definition recognizes a group of host screen characters, ACE generates the corresponding element. Generally, the GUI element also includes in some form the specific characters that the Pattern Definition recognized.

The Parts of a Pattern Definition

A Pattern Definition has four parts. These are:

- *Type*

  The general form of the Pattern Definition.
- *Structure*

  The details of which character groups the Pattern Definition recognizes. The Pattern Definitions have a hierarchical form—large Pattern Definitions built from smaller Pattern Definitions—in analogy with how characters form words and phrases.
- *Name*

  A unique identifier for the Pattern Definition.
- *Parameters*

  Refinements to the Pattern Definition that are not related to host characters.

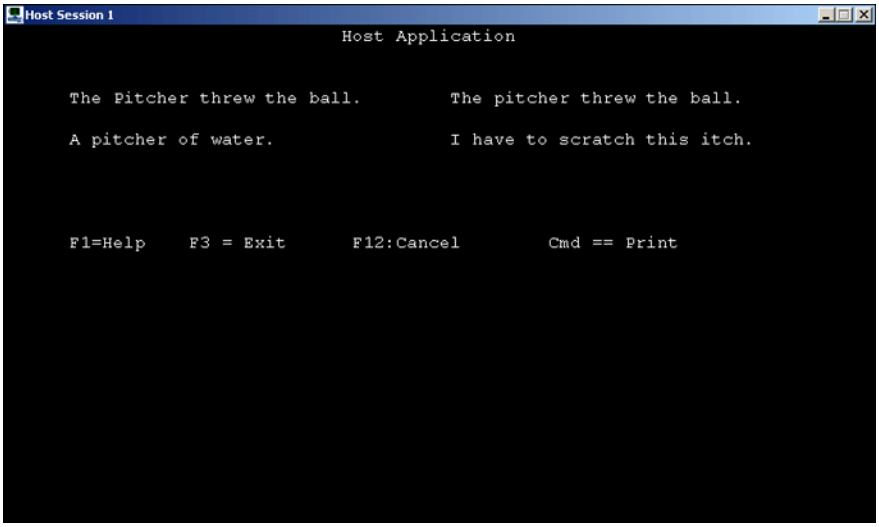
Examples Using String Type Pattern Definitions

A string type Pattern Definition is a rule that recognizes a specific sequence of characters.

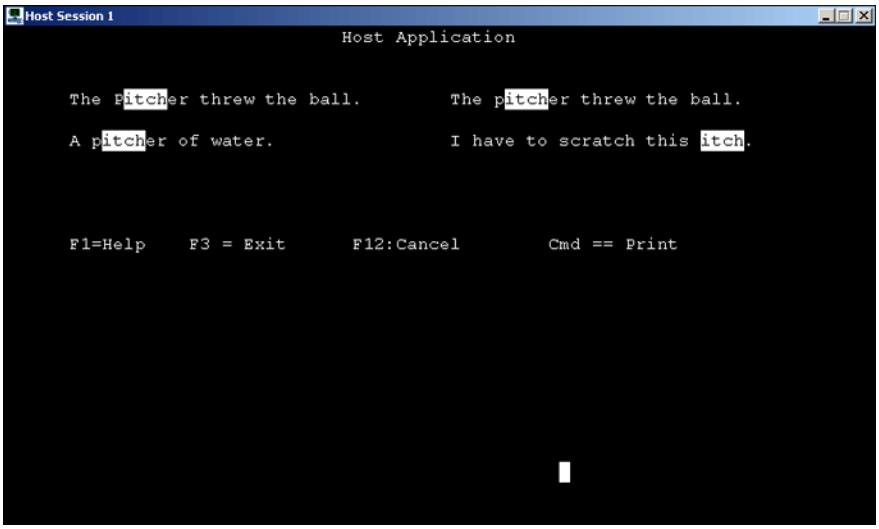
Example 2. String type pattern definitions

Suppose there is a string type Pattern Definition named itchStr that recognizes the character sequence: `itch`

The following figure shows a simulated host screen:

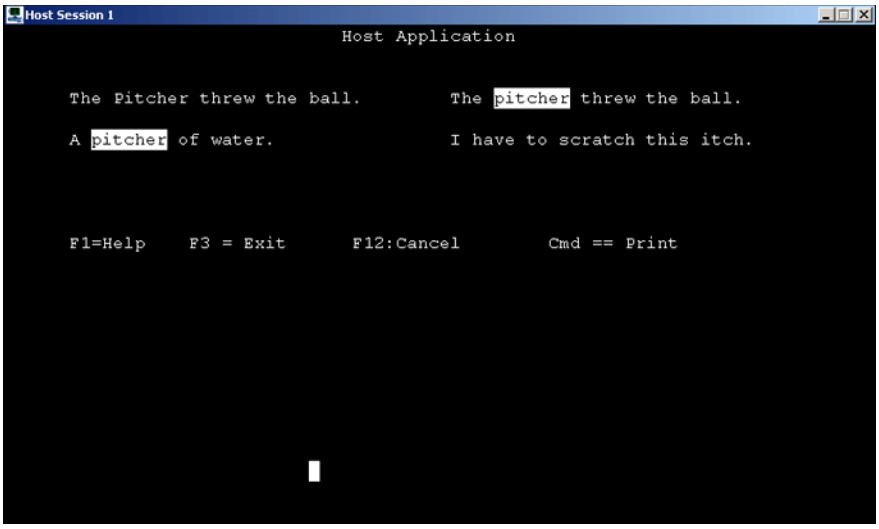


When ACE scans the host screen with the Pattern Definition itchStr it finds four separate character groups that satisfy itchStr as illustrated:



Notice that **`itchstr`** found every occurrence of the character sequence `itch` regardless of preceding or following characters.

Now, suppose there is another string type Pattern Definition named **`pitcherStr`** that recognizes the character sequence: `pitcher`. There are two character groups that satisfy this Pattern Definition:



Here you should notice that the sequence `Pitcher` is not recognized. A string type Pattern Definition generally considers upper and lower case letters to be separate characters. However, this can be relaxed.

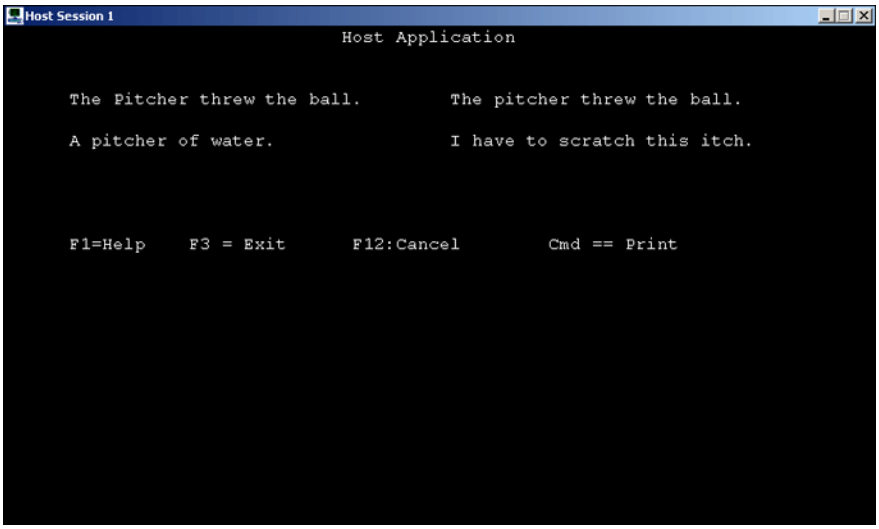
How ACE Applies Pattern Definitions to the Host Screen

ACE applies the Pattern Definitions by taking each one in turn and scanning it over the screen. Each time a character sequence satisfying the Pattern Definition is found ACE marks the characters and does not let any other Pattern Definitions scan those characters.

This means that the results of the Pattern Definition analysis depend on which Pattern Definition ACE takes first and which Pattern Definition ACE takes second.

Example 3. How ACE applies pattern definitions to the host screen

ACE scans the screen with the Pattern Definitions **`itchStr`** and **`pitcherStr`**:



- If ACE scans with itchStr first, then the character sequence `itch` is found four times. When ACE scans with pitcherStr the character sequence `pitcher` is not found.
- If ACE scans with pitcherStr first then the character sequence `pitcher` is found twice. When ACE scans with itchStr the character sequence `itch` is also found twice.

How ACE Chooses Which Pattern Definition to Apply

There are three main ways that ACE chooses which Pattern Definitions to apply to a host screen. These are:

- Sections; priority within sections and priority between sections.
- Scanning with primary patterns only. Primary patterns are usually compound Pattern Definitions.
- The Pattern Definition's location parameter.

Sections in Brief

The ACE KnowledgeBase contains many hundreds of Pattern Definitions. In principle, when ACE reads the host screen it takes each one of these Pattern Definitions in turn and scans the entire screen.

In practice, ACE limits this process in two ways:

- ACE contains many mini-KnowledgeBases, called sections, each of which contains a small number of Pattern Definitions.
- A section is applied to just part of the host screen.

Thus, instead of scanning the entire screen with all the Pattern Definitions, ACE scans parts of the host screen with small numbers of Pattern Definitions.

Of course, ACE must decide which section to apply to which part of the host screen. This is the role of layouts. Moreover, when two sections are applied to overlapping areas of the host screen, ACE must decide which section's Pattern Definitions to scan the overlapped area with first, and which section's Pattern Definitions to scan the overlapped area with next.

Finally, ACE must choose which of a section's Pattern Definitions to take first, and which to take next.

Primary Patterns in Brief

ACE Pattern Definitions are built modularly; smaller Pattern Definitions are combined together into compound Pattern Definitions. This greatly reduces the size of the KnowledgeBase. However, this means that there are many small Pattern Definitions that by themselves have no meaning.

Sections do not include the small Pattern Definitions. Sections contain only primary Pattern Definitions. Primary Pattern Definitions are generally high level compound Pattern Definitions or string type Pattern Definitions.

The effect is that ACE does not scan with individual small Pattern Definitions, but with several small Pattern Definitions in combination.

The Location Parameter

The section mechanism imposes an external limit on which parts of a host screen ACE scans with a given Pattern Definition. Pattern Definitions can also contain internal limits on which screen area they can scan.

A Pattern Definition can be limited to:

- a single location on the screen
- a range of rows
- a range of columns

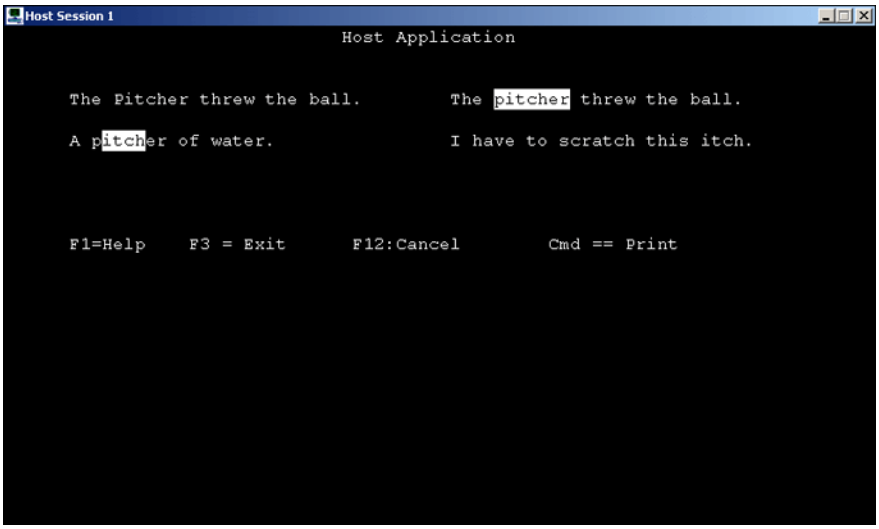
The ranges can be defined relative to the edges of the screen or to the edges of a list contained within the screen.

The ranges limit where ACE starts the scan. For a string type Pattern Definition this means that the first character in the string must lie within the defined range.

Example 4. The location parameter

- Suppose the string type Pattern Definition itchStr, that recognizes the character sequence `itch`, is limited to starting in the six left columns of the screen.
- Suppose the string type Pattern Definition pitcherStr, that recognizes the character sequence `pitcher`, is limited to starting in the four topmost rows of the screen.

Then in the host screen:



- the Pattern Definition itchStr recognizes only `A pitcher of water.`
- the Pattern Definition pitcherStr recognizes only `The pitcher threw the ball.`

Compound Pattern Definitions

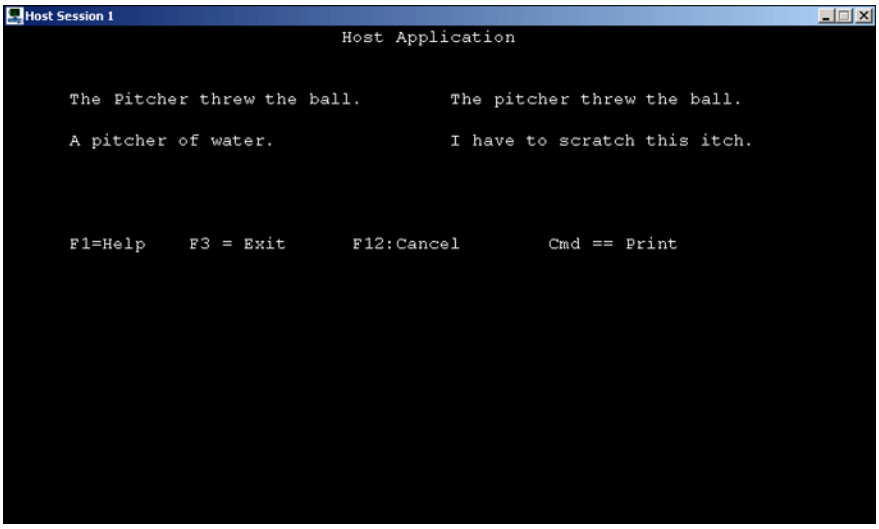
ACE Pattern Definitions are built modularly. There are three main ways that ACE combines Pattern Definitions:

- Groups where one, two or more smaller Pattern Definitions are combined in a fixed spatial relationship.
- OneOfs where one, two or more Pattern Definitions can be used interchangeably.
- Iterations where a single Pattern Definition is combined with itself an arbitrary number of times, in a fixed spatial relationship.

When these Pattern Definition types are combined together, a relatively small number of Pattern Definitions is able to recognize a wide variety of host screen character sequences.

Horizontal Groups: A First Look

The example host screen:



contains the character sequence "`A pitcher of water.`"

ACE could recognize this character sequence using a single string type Pattern Definition called, say, **`PitchWatStr`**. This is efficient, a single Pattern Definition, but not very flexible. **`PitchWatStr`** won't recognize "`A jug of wine.`" or "`A bottle of milk.`"

Instead, ACE could have four smaller string Pattern Definitions:

- **`ABlStr`** which recognizes "`A␣`"
- **`pitcherBlStr`** which recognizes "`pitcher␣`"

- **ofBlStr** which recognizes "`of∨`"
- **waterPerStr** which recognizes "`water.`"

"/" represents the empty character generated by pressing the spacebar.

A horizontal group type Pattern Definition can combine these four Pattern Definitions into a single Pattern Definition that recognizes the entire character sequence "`A pitcher of water.`" This Pattern Definition could be called, say, **APitWatHG**.

APitWatHG would have the structure:

APitWatHG

ABlStr

pitcherBlStr

ofBlStr

waterPerStr

A horizontal group defines a left-to-right fixed sequence of its component Pattern Definitions. In the above diagram, the top Pattern Definition AblStr is the left component Pattern Definition and the bottom Pattern Definition waterPerStr is the right component Pattern Definition.

A horizontal group has the following properties:

- The sequences it can recognize must be *horizontal*. The characters recognized by the component patterns must be in the same row (or rows, for child patterns that recognize blocks of characters.)

  Thus, APitWatHG recognizes "`A pitcher of water.`" But it does not recognize

  "`A pitcher/`

  `of water.`"

  "/" represents the character obtained by pressing the spacebar.
- The horizontal group is a group. Each of the component, or child, patterns must individually recognize a separate character sequence in order for the entire Pattern Definition to recognize a sequence. Thus, APitWatHG recognizes

  "`A pitcher of water.`" But it does not recognize any of

  "`A pitcher water.`" as ofBlStr is not matched

  "`pitcher of water.`" as ABlStr is not matched

  "`A pitcherof water.`" as pitcherBlStr is not matched
- The horizontal group is a sequence. The characters recognized by the child patterns must be in the same order as the order of the child patterns. Changes

in the order, insertions, or duplications are not allowed. Thus, **`APitWatHG`** recognizes

"`A pitcher of water.`" But it does not recognize any of

"`A of pitcher water.`" as ofBlStr and pitcherBlStr are not in order

"A pitcher of water ." as an additional blank character is not recognized by waterPerStr

"`A pitcher of of water.`" as a duplicate occurrence of ofBlStr is not allowed

Scanning and Child Patterns

The Pattern Definition APitWatHG performs correctly as long as other Pattern Definitions such as **`ofBlStr`** and **`pitcherBlStr`** have not already recognized the characters "`of/`" and "`pitcher/`". This is prevented by means of the section mechanism, which ensures that only **`APitWatHG`** is used for scanning the host screen.

OneOf Type Pattern Definitions

The *OneOf* type Pattern Definition is a compound Pattern Definition that recognizes sequences recognized by any one of the OneOf's child patterns. Suppose there are the string Pattern Definitions:

`pitcherBlStr` which recognizes "`pitcher/`"

`jugBlStr` which recognizes "`jug/`"

`bottleBlStr` which recognizes "`bottle/`"

`decanterBlStr` which recognizes "`decanter/`"

"/" represents the character obtained by pressing the spacebar.

These four Pattern Definitions can be combined into a OneOf type Pattern Definition called, say, **`ContainerOneOf`**:

```
ContainerOneOf
pitcherBlStr
jugBlStr
bottleBlStr
decanterBlStr
```

The Pattern Definition **`ContainerOneOf`** will recognize any of "`pitcher/`", "`jug /`", "`bottle /`" or "`decanter /`".

Horizontal Groups and OneOfs: Putting it Together

The example horizontal group **APitWatHG** is composed of four string type Pattern Definitions. Each child string pattern recognizes one fixed sequence, and the entire group also recognizes a single sequence.

If the string type Pattern Definitions are replaced with OneOf type Pattern Definitions then the same horizontal group structure can recognize many more character sequences. Suppose there is the horizontal group Pattern Definition **AContainerWatHG**.

AContainerWatHG has the structure:

- `AContainerWatHG`
- `ABlStr`
- `ContainerOneOf`
- `ofBlStr`
- `waterPerStr`

All the child Pattern Definitions have the same structure as in the preceding example. In this case, **AContainerWatHG** recognizes each of the four character sequences:

`"A pitcher of water."`

`"A jug of water."`

`"A bottle of water."`

`"A decanter of water."`

The **AContainerWatHG** Pattern Definition is not very efficient. It has four immediate child patterns, and one child has four child patterns of its own, for a total of nine Pattern Definitions. The four sequences recognized by **AContainerWatHG** could also be recognized by four separate string type Pattern Definitions.

Combining OneOf Type Pattern Definitions

Suppose now that instead of just "`water.`" the character sequence might include "`wine.`", "`whiskey.`" or "`milk.`". Four string type Pattern Definitions recognize each of these:

waterPerStr which recognizes "`water.`"

winePerStr which recognizes "`wine.`"

whiskeyPerStr which recognizes "`whiskey.`"

milkPerStr which recognizes "`milk.`"

These four string type Pattern Definitions are combined into a OneOf type Pattern Definition **`BeverageOneOf`**:

`BeverageOneOf`

`waterPerStr`

`winePerStr`

`whiskeyPerStr`

`milkPerStr`

Finally, a new horizontal group type Pattern Definition **`AContainerBeverageHG`** can be formed. **`AContainerBeverageHG`** has the structure:

`AContainerBeverageHG`

`ABlStr`

`ContainerOneOf`

`ofBlStr`

`BeverageOneOf`

`AContainerBeverageHG` can recognize 16 different character sequences:

"`A pitcher/jug/bottle/decanter of water/wine/whiskey/milk.`"

`AContainerBeverageHG` has four child patterns and two of the child patterns themselves have four child patterns. This gives a total of 13 Pattern Definitions recognizing 16 different character sequences.

Adding more child patterns to the OneOf type Pattern Definitions further increases the efficiency. If the string type Pattern Definition

`flaskBlStr` which recognizes "`flask/`" is added to ***ContainerOneOf*** and the string type Pattern Definition

`juicePerStr` which recognizes "`juice.`" is added to **`BeverageOneOf`** then the number of character sequences recognized by **`ContainerBeverageHG`** increases to 16 from 25, at the cost of only two Pattern Definitions.

Recognizing Real Screens: Character Sets and Iterations

The preceding sections explained two ways to combine string type Pattern Definitions into larger Pattern Definitions using OneOf type and horizontal group type Pattern Definitions. The next two sections explain the character set and iteration type Pattern Definitions.

The character set and the iteration type Pattern Definitions are at the heart of the ACE KnowledgeBase, because they can be combined together into Pattern Definitions that play the same role as string type Pattern Definitions but are far more flexible. The result is that a few Pattern Definitions are enough to recognize any host screen character sequence that a person would understand as words. It is not necessary to create a new string Pattern Definition for each different word.

Strings Have Their Uses

The preceding is not to say that string type Pattern Definitions are not useful. They are. However, they are not used for recognizing general character sequences. Rather, their role is to make sure that certain specific character sequences are treated differently than they would be if the sequence were to be recognized by a more general Pattern Definition.

The discussion of sections in the following chapters explains how Pattern Definition priority can be used in conjunction with string type Pattern Definitions to ensure that certain character sequences are given special treatment.

Character Set Type Pattern Definitions

A character set type Pattern Definition recognizes host screen character sequences that are only one character long. The allowed character, or characters, that the Pattern Definition can recognize is determined by the character set's structure. The character set can be structured to recognize a single specific character, or it can be structured to recognize any one of a set of characters. Some character set type Pattern Definitions might be:

`Period` which recognizes "."

`Colon` which recognizes ":"

`Blank` which recognizes " ", the character obtained by pressing the spacebar.

`LeftBracket` which recognizes any one of "[" or "{" or "("

`RightBracket` which recognizes any one of "]" or "}" or ")"

Character set Pattern Definitions are equivalent to string type Pattern Definitions where the string is only one character long, or more generally, to OneOf type Pattern Definitions where the child patterns are all length-one string type Pattern Definitions.

Thus:

The character set **`Period`** is equivalent to the string **`PeriodStr`** which recognizes "."

The character set **`LeftBracket`** is equivalent to OneOf:

```
LeftBrackOneOf
   LeftSquareStr which recognizes the string "["
   LeftCurlyStr which recognizes the string "{"
   LeftRoundStr which recognizes the string "("
```

Why Have Character Sets?

As character set Pattern Definitions are equivalent to strings of length-one, or to Oneofs whose child patterns are strings of length-one, it might seem superfluous to include character set Pattern Definitions in the KnowledgeBase. In fact, from a strictly structural standpoint this is true. However, it is very tedious to form OneOf Pattern Definitions composed of, say, ten strings representing each of the digits, or 26 strings representing each letter of the Roman alphabet, or 30 odd strings representing attribute characters.

Character sets are easier to build because they have the "OneOf" aspect and the "string of length-one" aspect built in. Creating a character set type Pattern Definition is simply a matter of designating the individual characters belonging to the Pattern Definition.

Using Character Sets

Character sets are used in the same way that strings are used. In the preceding examples, string type Pattern Definitions that recognized words ended in either blanks or periods:

pitcherBlStr recognizes "pitcher␣"

waterPerStr recognizes "water."

"␣" represents the character obtained by pressing the spacebar.

A more efficient construction puts the characters following words in a character set, and the string type Pattern Definitions contain only a word's letters:

`AStr` recognizes "`A`"

`pitcherStr` recognizes "`pitcher`"

`ofStr` recognizes "`of`"

`waterStr` recognizes "`water`"

The period and spacebar character are placed in a character set type Pattern Definition:

`PeriodOrSpaceCH` recognizes "." or "␣"

Finally, a horizontal group Pattern Definition **`APitcherWaterHG`** with the structure:

```
APitcherWaterHG
AStr
PeriodOrSpaceCH
pitcherStr
PeriodOrSpaceCH
ofStr
PeriodOrSpaceCH
```

waterStr

PeriodOrSpaceCH

The above recognizes the entire sequence "`A pitcher of water.`"

Flexibility Versus Specificity

The advantage of the preceding construction is that each word does not have to appear in two forms, one with a trailing space and the other with a trailing period.

The cost is that the horizontal groups are larger. More importantly, there is a loss of specificity. **APitcherWaterHG** also recognizes:

"A.pitcher.of.water."

and all the other various combinations of periods and spaces.

The KnowledgeBase is designed to strike the correct balance between flexible Pattern Definitions and specifically targeted Pattern Definitions. *JIS Interface Server: KnowledgeBase User's Guide* explains how to create and modify Pattern Definitions while preserving this balance.

Iteration Type Pattern Definitions

A horizontal iteration type Pattern Definition recognizes multiple occurrences, one right after the other, of a single child pattern.

An important part of any iteration type Pattern Definition is the minimum and maximum number of allowed occurrences of the child pattern.

Example 5. Iteration type pattern definitions

Suppose the character set type Pattern Definition **Seven** recognizes "7". You want a Pattern Definition that recognizes from three to six sequential occurrences of "7":

"777"

"7777"

"77777"

"777777"

A Pattern Definition that can do this is a horizontal iteration:

Seven3to6HI

Seven which recognizes the character "7".

The number of allowed iterations is set by two parameters:

- Minimum iteration number, set equal to three in this case
- Maximum iteration number, set equal to six in this case

> **Note:** The name **`Seven3to6HI`** is just a name. It does not set the number of allowed iterations.

Properties of Iterations

Iteration type Pattern Definitions have the following properties:

- The largest possible number of iterations, up to the maximum, is recognized, even if this means that there are "left over" characters. Thus **`Seven3to6HI`** recognizes
  `"7777777"`
- It does not recognize first four "7"s and then three "7"s:
  `"777777"`
- The character groups recognized by the child pattern must be in sequence. Thus **`Seven3to6HI`** recognizes
  `"7777"`
- But it does not recognize
  `"77 77"`
- As with all other Pattern Definitions, horizontal iterations scan from left to right. Thus **`Seven3to6HI`** recognizes
  `"7777777"`
- It does not recognize
  `"777777"`

Using Iterations

Horizontal iterations, in combination with character sets and horizontal groups, can be used to recognize any word and any combination of words. This means that instead of an enormous number of string type Pattern Definitions, the KnowledgeBase need only contain a single, properly constructed, Pattern Definition.

Recognizing Words

The Pattern Definition that recognizes words is based on a character set type Pattern Definition that recognizes letters. This Pattern Definition is called, say, **AnyLetter**. **AnyLetter** recognizes any one of the 26 lower case and 26 upper case letters of the Roman alphabet:

"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

The Pattern Definition that recognizes words is called, say, **WordHI**. **WordHI** is a horizontal iteration of **AnyLetter**:

- The minimum iteration number is set equal to one
- The maximum iteration number is set equal to 9999

WordHI recognizes

- "A"
- "pitcher"
- "Pitcher"
- "pItcHEr"
- "hiPtREc"
- "dfkgkjerodofnOieworeoienoIEIOoioIoIOIjOJOPEFE0pI"

WordHI does not recognize the entire phrase "A pitcher of water." but it does recognize the individual words "A pitcher of water." as four separate instances of **WordHI**.

Recognizing Combinations of Words

A Pattern Definition that can recognize a combination of words is based on an iteration of **WordHI**. However, **WordHI** cannot recognize the space between words. To handle this, a character set Pattern Definition that recognizes the space must be grouped with **WordHI**. This Pattern Definition is called, say, **Blank**.

Blank recognizes the character obtained by pressing the spacebar.

Blank is combined with **WordHI** into a horizontal group called, say, **BlankWordHG**. **BlankWordHG** has the structure:

BlankWordHG

Blank which recognizes the character obtained by pressing the spacebar **WordHI** which recognizes any number of lower or upper case letters

In the character sequence "A pitcher of water." **BlankWordHG** recognizes three separate character sequences:

"A pitcher of water."

Notice that **BlankWordHG** does not recognize the first "word" since it is not preceded by a blank. However, the first word is recognized by **WordHI**. A Pattern Definition that can recognize an entire phrase must be composed of **WordHI** followed by any number of instances of **BlankWordHG**.

A Pattern Definition that can recognize any number of instances of **BlankWordHG** would be a horizontal iteration called, say, **BlankWord0orMore**:

BlankWord0orMore

BlankWordHG

- The minimum iteration number is set equal to zero
- The maximum iteration number is set equal to 9999

Notice that the minimum iteration number is set equal to zero. This is to allow for one word phrases. In such a case no instances of **BlankWordHG** are required.

BlankWord0orMore recognizes `"A pitcher of water."`

The two Pattern Definitions **WordHI** and **BlankWord0orMore** are combined into a horizontal group that recognizes entire phrases. This horizontal group is called, say, **PhraseHG** and it has the structure:

PhraseHG

WordHI recognizes any combination of lower and upper case letters

BlankWord0orMore recognizes any number of blank-word groups

PhraseHG recognizes `"A pitcher of water."`

A final refinement of **PhraseHG** is to allow for the period or other character to end a phrase. The pieces are:

- A character set Pattern Definition called, say, **EndChars** that contains the period and any other desired characters such as the exclamation point "!" and the question mark "?"
- A horizontal iteration Pattern Definition called, say, **EndChars0or1** that contains **EndChars**. **EndChars0or1** has a minimum iteration number of zero and a maximum iteration number of one. Notice that this construction is conceptually the same as making **EndChars** optional.
- **EndChars0or1** is added to the end of **PhraseHG**
- **PhraseHGlb**

  WordHI recognizes any combination of lower and upper case letters

  BlankWord0orMore recognizes any number of blank-word groups

  EndChars0or1 recognizes a period, if present

Other Compound Pattern Definition Types

The ACE KnowledgeBase contains many other types of compound Pattern Definitions. All of these work in the same way:

- The Pattern Definition is a parent of one or more child patterns.
- The Pattern Definition type defines a spatial relationship between the child patterns.
- The Pattern Definition recognizes a character sequence only when all the child patterns individually recognize character sequences, and only when these child-recognized character sequences have the spatial relationship defined by the parent Pattern Definition.
- ACE scans the host screen, or part of the host screen, with the parent Pattern Definition. ACE scans from left to right in each row, and each row is scanned in turn from top to bottom.

Dynamic Pattern Definitions

Dynamic Pattern Definitions are special Pattern Definitions that ACE uses when analyzing dynamic screens. Dynamic Pattern Definitions recognize the same character sequences as regular Pattern Definitions and they generate the same GUI controls as regular Pattern Definitions.

The difference between regular Pattern Definitions and Dynamic Pattern Definitions lies in what ACE does when the Pattern Definition recognizes a character sequence.

How ACE Uses Regular Pattern Definitions

When a regular Pattern Definition recognizes a host screen character sequence ACE does two things:

- ACE places the control associated with the Pattern Definition on the GUI window. The location of the control on the GUI corresponds to the location of the character sequence on the host screen. The existence of the control and the type of control are fixed, even though certain parameters of the control may not be established until runtime.
- ACE uses the host region containing the characters as an identifying characteristic of the screen. In some cases the actual characters found in the region are fixed, in other cases they can be variable. In either case, during runtime the executable checks that characters are present only in the area in which the Pattern Definition found characters.

In runtime, when the executable identifies a particular host screen, the executable displays the corresponding GUI. This GUI contains the controls generated when the screen was analyzed in ACE.

Regular Pattern Definitions and Dynamic Screens

Some host screens display different character sequences depending on the state of the application. A common example is the set of command key prompts that appear at the bottom of the screen.

Example 6. Regular pattern definitions and dynamic screens

For one state of a host application, the screen displays six command key prompts along the bottom of the screen:

```
δF3=Exit  F5 = New Order  F11=Fold   F12=Cancel      Roll Up/Dn   F18=Printà
δ00000000000000000000000000000000000000000000000000000000000000000000000000000à
```

When the application is in a second state, the same screen displays only five command key prompts:

```
δF3=Exit    F4=Prompt    F5=  New Customer   F12= Cancel    F18=Printà
δ00000000000000000000000000000000000000000000000000000000000000000000000000000à
```

If ACE analyzes the first version of this screen using a regular Pattern Definition, the executable will only identify this version of the screen in runtime. This is because the runtime is “looking for” exactly six command key sequences in the second last row of the screen.

> **Note:** For more information about using dynamic sections effectively, see the section entitled “Defining Dynamic Sections in Layout View in *JIS Interface Server: Advanced Topics*.

How Dynamic Pattern Definitions Work

Dynamic Pattern Definitions differ from regular Pattern Definitions in the following way:

- Regular Pattern Definitions identify specific locations of character sequences. The same, or similar, sequences are expected in the same locations during

runtime. If more or fewer sequences are present in runtime, the screen will not be identified.

- Dynamic Pattern Definitions identify an area of the screen in which character sequences may potentially occur in runtime. Any number of character sequences satisfying the dynamic pattern definition may occur in runtime. The screen will still be correctly identified.

Why Use Regular Pattern Definitions?

Regular Pattern Definitions give the host screens distinguishing marks that enable the executable to identify the screen in runtime.

The result of using regular Pattern Definitions is that the executable knows that on a given screen specific character sequences occur in specific locations, while other areas of the screen are blank. This information allows the executable to identify the individual screens.

Dynamic Pattern Definitions do not provide this information. When ACE uses a dynamic Pattern Definition to scan a screen area, all the executable knows is that specific types of character sequences may appear anywhere in a certain area, for example, FKeys. This can cause misidentification between similar screens in runtime.

Dynamic Pattern Definitions in Detail

Dynamic Pattern Definitions have the following general form:

```
DynamicGroup
  DynamicIteration1
  RegularPatternDefinition1
```

- The dynamic group Pattern Definition is a special KnowledgeBase structure used for dynamic Pattern Definitions. It can contain one or more dynamic iterations. The dynamic group recognizes characters that satisfy its individual child dynamic iteration patterns. The characters satisfying the various dynamic iterations do not have to lie in a fixed spatial relationship.
- Each dynamic iteration Pattern Definition recognizes one or more occurrences of its child pattern. The individual sequences recognized by the child pattern do not have to be adjacent to each other, nor do they have to lie in a fixed spatial relationship within the dynamic group area.
- The *regular* Pattern Definitions recognize character sequences containing specified characters in a fixed spatial relationship. These regular Pattern Definitions can be complex, containing horizontal and vertical groups, and horizontal and vertical iterations.

When Does ACE Use Dynamic Pattern Definitions?

ACE scans with dynamic Pattern Definitions when you instruct it to do so. You specify an area of the screen to which you wish to apply a dynamic group type Pattern Definition. ACE scans that area with the dynamic group.

Chapter 4. Sections and Layouts

Sections and layouts are the mechanism ACE uses to select the Pattern Definitions it uses when scanning the host screen.

This chapter describes:

- Introduction to Sections and Layouts
- How Sections Improve Host Screen Analysis
- The Scanning Order for Pattern Definitions
- Filter Sections
- Using Layouts

This chapter and Chapter 3 - "How Pattern Definitions Work" on page 55 describe the objects ACE uses to analyze host screens. These are the objects you will modify in order to influence the analysis.

Related information:

- Step-by-step operating instructions for modifying sections and layouts can be found in Chapter 5 - "Layout View Operations" on page 89.
- Step-by-step operating instructions for modifying Pattern Definitions are contained in Chapter 7 - "Pattern Definition Modifications Through the KnowledgeBase" on page 121.
- Conceptual information about how best to modify Pattern Definitions is the subject of *JIS Interface Server: KnowledgeBase User's Guide*.

Introduction to Sections and Layouts

The heart of the ACE conversion process is the analysis of host screen character groups by the KnowledgeBase Pattern Definitions. Sections and layouts are objects that ACE uses to speed this process. Without sections and layouts, analysis would be very time consuming; the KnowledgeBase contains many hundreds of Pattern Definitions, and a screen image can contain many different types of character groups.

What Are Sections?

A section is a mini KnowledgeBase containing a limited number of Pattern Definitions selected from the entire KnowledgeBase. A section's Pattern Definitions are applied to only part of the screen image and not to the entire screen.

The following are examples of Sections that reside in the default KnowledgeBase.

- The section *WindowCaption* contains the single Pattern Definition *WindowCaption*.
- The section *WebLook_Title* contains two Pattern Definitions: *WebLook_Title* and *WebLook_TitleOutput*.
- The section *ScreenData* contains four Pattern Definitions: *SystemDateViaSDFField*, *SystemTimeViaSDFField*, *UserIDViaSDFField* and *SystemIDViaSDFField*.

Figure 19 shows the section WindowCaption applied to the middle of the first row of a screen image. The section appears as a rectangle with a solid outline:

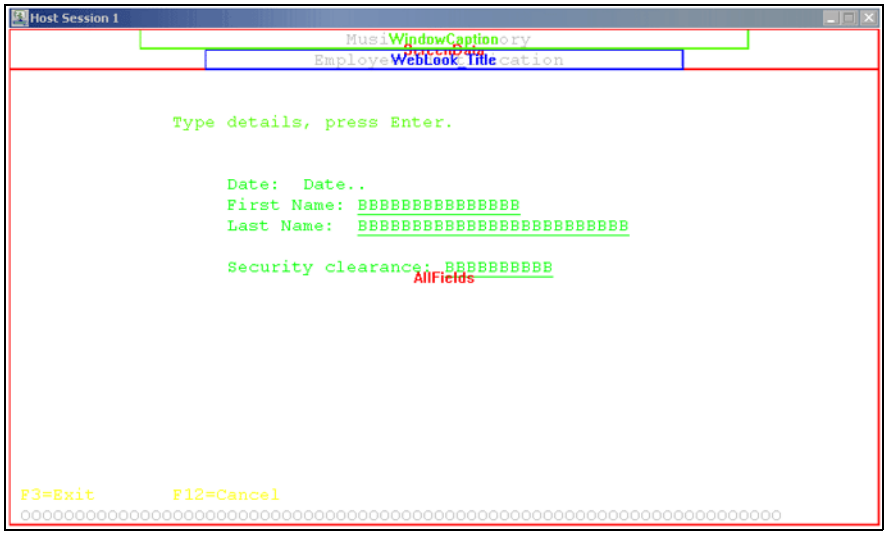


Figure 19. WindowCaption section

ACE applies the Pattern Definition *WindowCaption* only to the middle of the first row of the screen image.

The section ScreenData is applied to the first two rows of the screen image. The four Pattern Definitions in the ScreenData section are applied to only the first two rows of the screen image.

The section *WebLook_Title* is applied to the middle of the second row of the screen image. The Pattern Definitions in that section are applied only to that area.

What Are Layouts?

A layout is an arrangement of sections on a screen image.

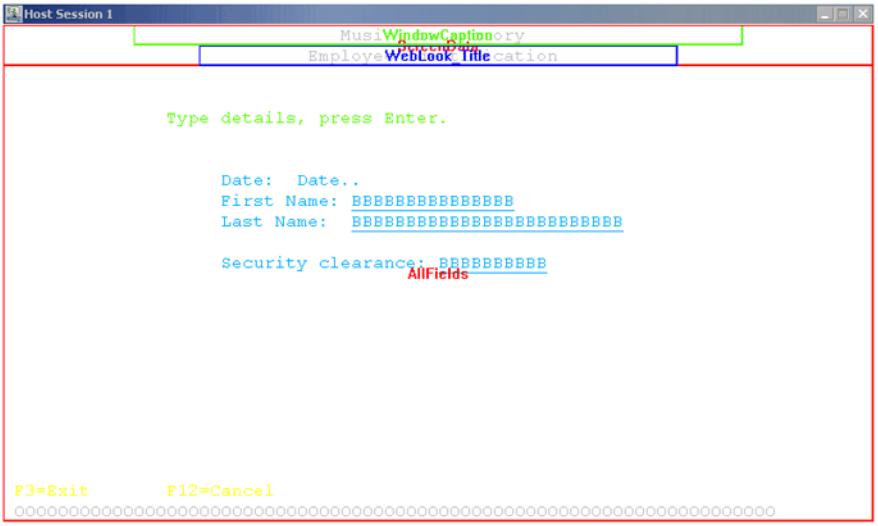


Figure 20. A layout

Figure 20 shows the WebLook layout. The WebLook layout consists of four sections:

- The section WindowCaption applied to the middle of the first row of the screen image.
- The section *WebLook_Title* applied to the middle of the second row of the screen image.
- The section ScreenData applied to the first two rows of the screen image.
- The section AllFields applied to the entire screen image.

What is Layout View?

ACE's Layout View displays the sections that have been applied to the host screen. Layout View is also where you can apply sections, or complete layouts, to the host screen, and where you edit the existing sections and layouts.

Full operating instructions are contained in Chapter 5 - "Layout View Operations" on page 89.

How Sections Improve Host Screen Analysis

Each KnowledgeBase Pattern Definition defines a character sequence or a range of character sequences. ACE analyzes a host screen by selecting a Pattern Definition and scanning the screen to find each character sequence that satisfies the Pattern Definition. ACE then selects another Pattern Definition and scans the screen again, skipping characters that were already picked up by other Pattern Definitions. The process continues until every character on the host screen has been recognized by some Pattern Definition.

In practice, ACE does not use every KnowledgeBase Pattern Definition to scan the entire screen. The scanning process is limited in two ways:

- The Pattern Definitions used are taken from a mini KnowledgeBase called a section.
- Sections are applied to only part of the screen.

Why Use Sections?

Using Sections increases both the speed and accuracy of the host screen analysis.

- The speed increases because fewer Pattern Definitions are scanned over smaller regions.
- The accuracy increases because different Pattern Definitions that recognize the same type of character sequences are prevented from interfering with each other. The following example illustrates this point.

Figure 21 shows the top region of a host screen:

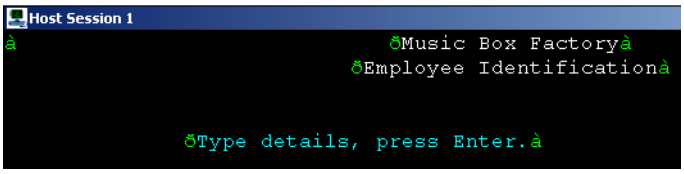


Figure 21. Top region of a host screen

The character sequence `Music Box Factory` in the top line serves to identify the host application. The character sequence `Type details press Enter` is an instruction to the user. What is important to note here is that these two character sequences are similar; both sequences are made up of alphabetic characters formed into common English words.

When ACE creates a GUI for this host screen, it treats these two character sequences very differently:

Figure 22. Top region of a GUI window derived from the last host screen

The host screen characters `Music Box Factory` become the window caption for the GUI, while the host screen characters `Type details, press Enter` are displayed as text in the window client area.

ACE treats these character sequences differently because they are recognized by different Pattern Definitions. One Pattern Definition turns character sequences it recognizes into window captions, while the other Pattern Definition turns character sequences it recognizes into text on the GUI client area.

However, these two different Pattern Definitions actually define the same range of character sequences—either Pattern Definition can recognize both `Music Box Factory` and `Type details press Enter`. ACE prevents these two Pattern Definitions from interfering with each other by making them members of separate sections:

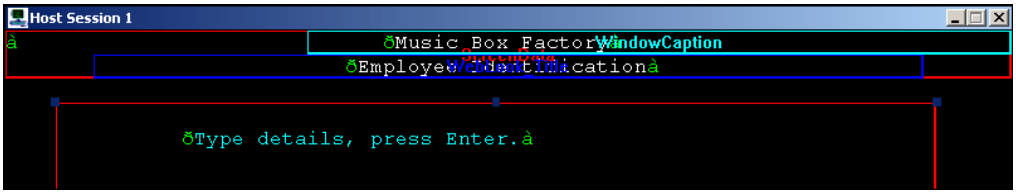


Figure 23. Members of separate sections

The Pattern Definition that generates window captions is a member of a section that is applied to the top of the host screen image.

The Pattern Definition that generates text on the GUI client area is a member of a section that is applied to the middle of the host screen image.

List Sections Applied by the ACE Wizards

There are some sections that are used by the ACE wizards when you mark lists on screen images. Unlike the other ACE sections, you do not have access to these sections. You cannot edit these sections and you cannot manually apply these sections to a screen image.

The Scanning Order for Pattern Definitions

ACE analyzes any particular area of the host screen using all the Pattern Definitions contained in all the sections that are applied to the particular area.

ACE uses all the Pattern Definitions in the sections DefaultText and FixedFontHeader to analyze the character sequence `Security`:

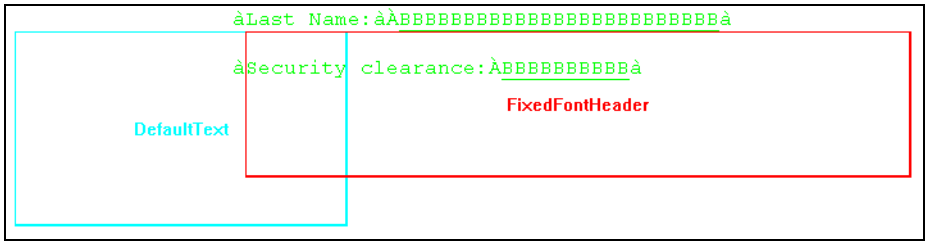


Figure 24. FixedFontHeader section

However, once any character sequence is recognized by a Pattern Definition, no other Pattern Definition can recognize that character sequence. Thus, the order in which ACE selects Pattern Definitions for analyzing the host screen is very important.

About Ordering Pattern Definitions

The Pattern Definitions are ordered on two levels:

- First, ACE orders the sections overlapping a given area. ACE analyzes the overlapped area with all the Pattern Definitions in the first section before going on to the Pattern Definitions in the second section.
- Second, ACE orders the Pattern Definitions within each section. The ordering within the section is the same as the order in which the Pattern Definitions are listed in the section's *Section Definition* dialog box.

The result is that ACE analyzes the overlapped area with Pattern Definitions chosen in the following order:

1 First section, first Pattern Definition

2 First section, second Pattern Definition

3 First section, last Pattern Definition
4 Second section, first Pattern Definition

How Sections are Ordered

When many sections are applied to overlapping areas of the host screen, the section applied to a smaller total area has priority over a section applied to a larger area.

The result is that character sequences contained in an overlapped area are analyzed first by the Pattern Definitions in a "smaller" section and then by the Pattern Definitions in a "larger" section.

In Figure 24, the character sequence `Security` is covered by two sections, *DefaultText* and *FixedFont*. The total host screen area covered by the *DefaultText* section is smaller than the total host screen area covered by the *FixedFontHeader* section.

Thus, *DefaultText* is the "smaller" of the two sections and has priority: ACE will analyze the character sequence `Security` with the Pattern Definitions contained in DefaultText first. If these characters are not recognized by any of DefaultText's Pattern Definitions, ACE will analyze the characters with the Pattern Definitions contained in the FixedFontHeader section.

Filter Sections

ACE can use the field information included in host application screen definition files (SDFs) to increase the accuracy of its analysis.

This information is represented in Layout View by filter sections drawn around areas of the host screen corresponding to SDF fields.

Note: See "Displaying Filter Sections" on page 105 if filter sections are not visible in Layout View.

Filter sections appear as rectangles with a dashed outline:

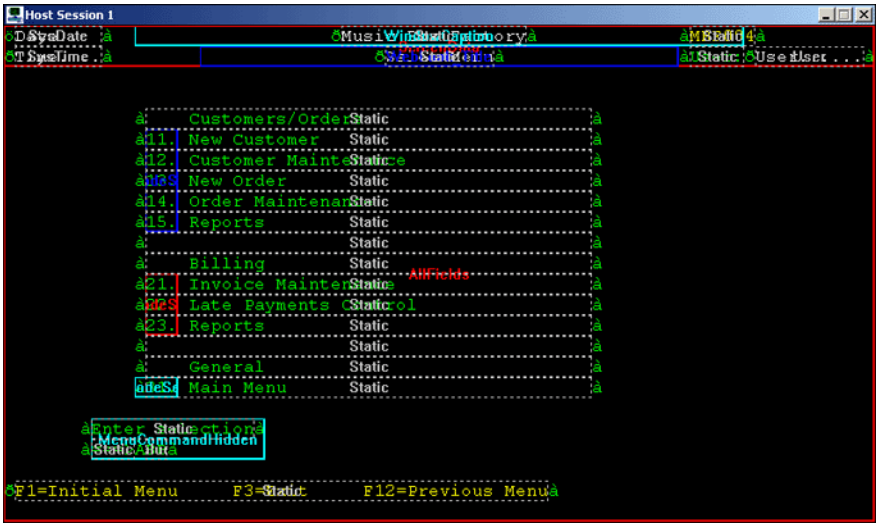


Figure 25. Filter sections

How Filter Sections Work

ACE has a filter section for each type of SDF field. ACE reads an SDF and determines that there is a field of a given type in a particular location. ACE then applies the corresponding filter section to that same area of the host screen.

A filter section contains Pattern Definitions that recognize character sequences typical of the corresponding SDF field. These Pattern Definitions are identical to the other Pattern Definitions in the KnowledgeBase.

A Pattern Definition in a regular section recognizes a character sequence when the following conditions are met:

- The recognizing Pattern Definition is a member of a regular section applied to the area containing the character sequence.
- The recognizing Pattern Definition follows the regular rules for recognizing character sequences; the Pattern Definition must be built from child patterns that together recognize all the characters in the sequence.
- If some of the characters in the sequence are in an SDF field, then the child patterns that recognize those characters must also be in the filter sections applied to those characters.

Generally, the Pattern Definitions in a filter section are themselves compound Pattern Definitions. Their component Pattern Definitions *do not* have to be in the filter section.

> **Note:** When a Pattern Definition in a filter section recognizes a character sequence it does not ensure analysis, it only enables the analysis. In order for the analysis to proceed, the Pattern Definition in the filter section, or its parent, must be in the regular section covering those characters.

Figure 26 shows the top part of a host screen image:

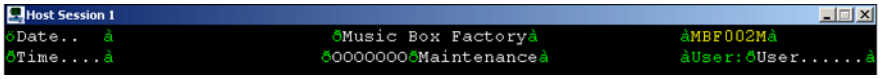


Figure 26. Filter sections in the top section of the host screen

This screen image was generated from an SDF and it has several fields.

Four of these fields are:

- A *date* field in the upper left, in the area between the ð attribute and the à attribute.
- A *time* field in the lower left, in the area between the ð attribute and the à attribute.
- A *user* field in the lower right, in the area between the ð attribute and the à attribute.
- A static field to the left of the user field and between the à attribute and the ð attribute. The field contains the characters "`User: `".

The date, time and user fields all contain the same type of character sequences. However, you may wish to represent their contents in the GUI in different ways. Thus, you require three different Pattern Definitions that all recognize the same character sequences, but that will not interfere with each other *even when all three Pattern Definitions are in the same regular section*.

ACE automatically uses filter sections to distinguish between such structurally identical Pattern Definitions.

Figure 27 shows the filter sections applied to the screen image. Notice that the filter sections sit between the attributes, covering each field precisely:

Figure 27. Filter sections sit between the attributes

- A date filter section covers the date field.
- A time filter section covers the time field.
- A user filter section covers the user field.
- A static filter section covers the static field containing the characters "User:".

> **Note:** The regular sections have been removed for clarity. A complete arrangement of sections would not look like this.

Each of the three filter sections date, time, and user contains a single Pattern Definition: DateViaSDFField, TimeViaSDFField and UserViaSDFField respectively. These three Pattern Definitions have different names and so each can represent the characters it recognizes differently. All three Pattern Definitions are contained in the regular section applied to the top two rows of the screen (not shown.)

The three Pattern Definitions are structurally the same. However, DateViaSDFField, for example, does not recognize the characters in the time field because DateViaSDFField is not contained in the time filter section. The filter sections prevent the structurally identical Pattern Definitions from interfering with each other.

Why Regular Sections are Necessary

This example also illustrates why the analysis does not proceed directly from the filter sections, and the Pattern Definitions the filter sections contain. The characters `User:` in the *static* field together with the contents of the *user* field form a single unit that should be represented together. The only way this can happen is if the Pattern Definition in the *static* field and the Pattern Definition in the *user* field are building blocks of a larger Pattern Definition, and the GUI component is attached to the larger Pattern Definition.

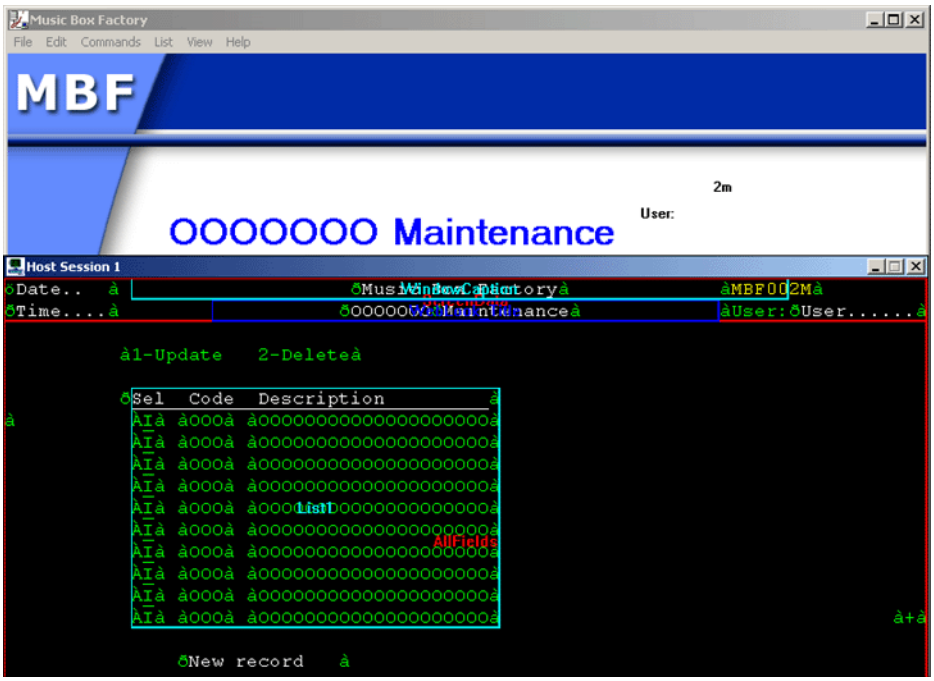
Using Layouts

ACE uses layouts to apply several sections to a screen image in a single step. As screens often have a uniform look across an Application, a single layout can be used for a number of host screens. While the layouts supplied with ACE may not reflect the composition of your host application's screens, ACE provides you with the tools for creating your own layouts.

Example 7. Using layouts

In Example 7 creating a new layout allows you to avoid having to repeatedly edit the sections applied to each screen. The example presents the steps and the reasons behind the steps, but does not give the operating instructions. See Chapter 5 - "Layout View Operations" on page 89 for detailed operating instructions.

The following figure shows the top part of the GUI that ACE generates from the screen image:



The screen image shows the WebLook layout supplied with ACE. This layout includes the WindowCaption and the *WebLook_Title* sections. In addition, the section List1 was added in the New Subapplication Wizard.

Note that in the host screen image the character sequence `MBF002M` is partially covered by the WindowCaption section and part of it extends outside the WindowCaption section. This gives rise to two related problems:

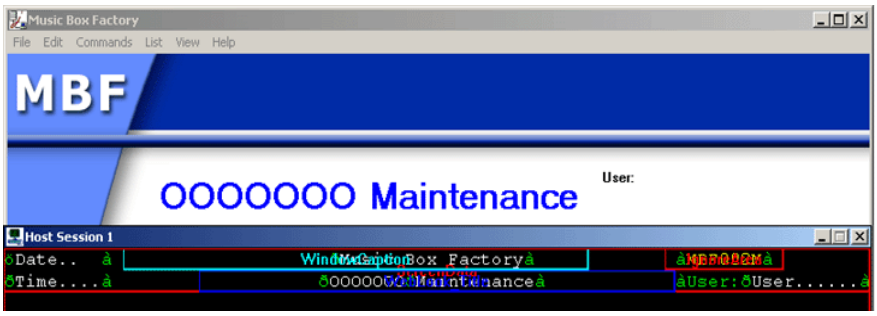
- The WindowCaption section is applied to too large an area, thereby including characters unconnected with the window's caption.
- The screen ID characters `MBF002M` are not covered uniformly by the same sections, so they cannot be analyzed together. In fact, only "2M" appears on the GUI.

There are many possible solutions to these problems. The following solution illustrates the use of Layouts. The first part of the solution is to decrease the area covered by the WindowCaption section:

This changes the GUI to:

The second part of the solution refines how the character sequence `MBF002M` is treated. In the above example, the character sequence `MBF002M` is included in the Screen Data section. Suppose you want hide the screen ID information. In this case, the IgnoreArea is dragged over character sequence `MBF002`.

The result is:



The current arrangement of sections can be saved as a layout named, for example, *IgnoreScreenID*. The *IgnoreScreenID* layout can be applied whenever you want to ignore characters located in the same location that is covered by the *IgnoreArea* section in the IgnoreScreenID layout.

Note: A new layout does not include sections added in a wizard even though the sections are present on the screen image when the new layout is saved. In this example, the section List1 was added by a wizard, but List1 is not included in the new layout IgnoreScreenID. Thus, a new layout is suitable for all three main types of screens: general input/output, menus and lists.

Chapter 5. Layout View Operations

In the New Subapplication wizard you select a layout for your screen image. You may also add sections appropriate for lists or for menus. In most cases, these sections allow ACE to make a complete analysis of the host screen; ACE generates the correct GUI element from each host screen character sequence.

If the arrangement of applied sections is not optimal, you can change the configuration in Layout View.

This chapter describes:

- What You See in Layout View
- Operations with Sections
- Sections and Pattern Definition Search Order
- Operations with Layouts
- Filter Section Operations
- Menus Specific to Layout View

What You See in Layout View

In Layout View your screen appears as follows:

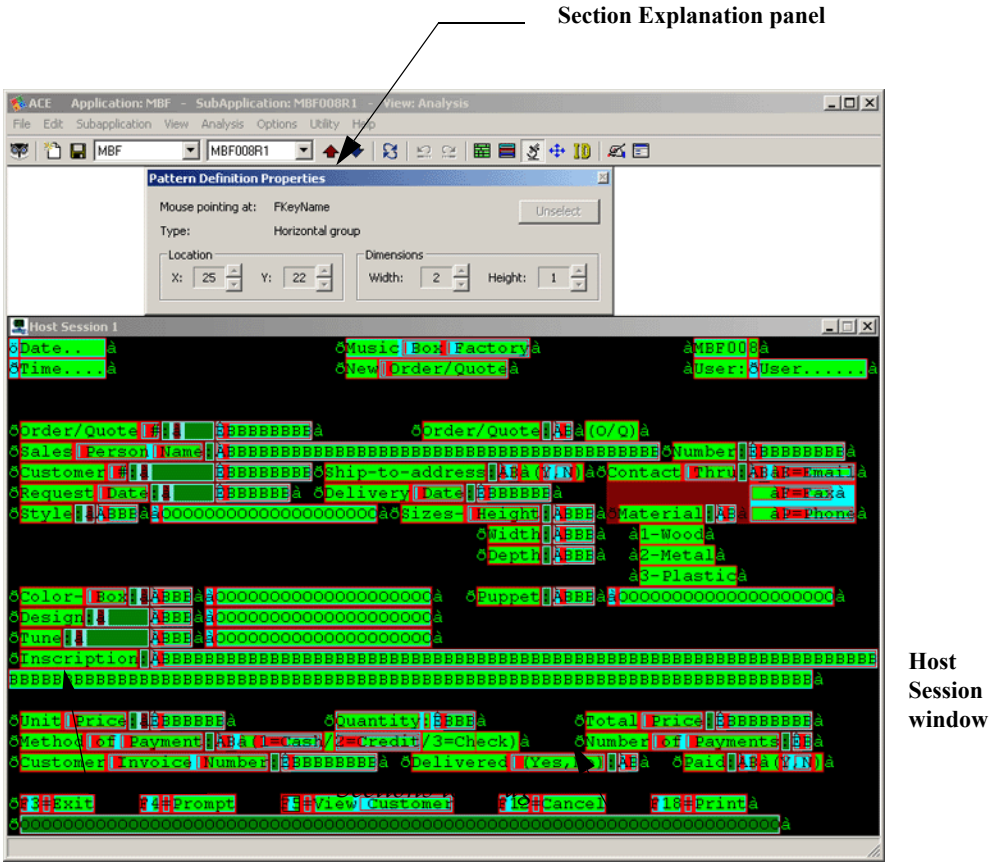


Figure 28. Layout view

The ACE client area pictured above contains three windows:

- The *Host Session* window
- The *Sections to Drag* panel
- The *Section Explanation* panel

Host Session Window

Layout View displays the host screen with colored rectangles marking the areas to which sections have been applied:

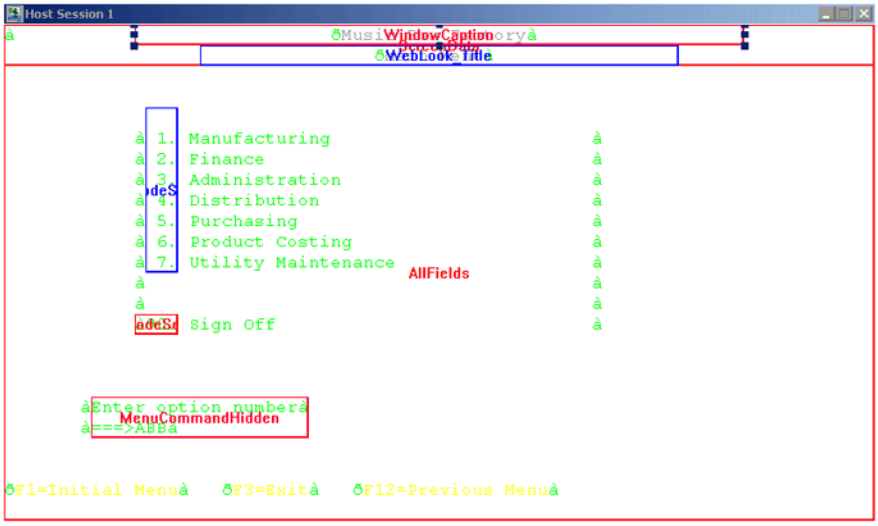


Figure 29. The colored rectangles in Layout view

The colors have no significance other than to make each section visually distinct. The section's name appears in the center of the rectangle.

Sections may overlap on the screen, or be entirely contained in sections applied to larger areas. Clicking within the border of a section selects it for Layout View operations.

If your screen image was generated from a screen definition file, such as DDS, BMS or MFS, the screen image may also display applied filter sections. The rectangles marking applied filter sections are drawn with dashed lines.

Sections To Drag Panel

The Sections To Drag panel lists the available sections, and is your tool for applying them to the screen.

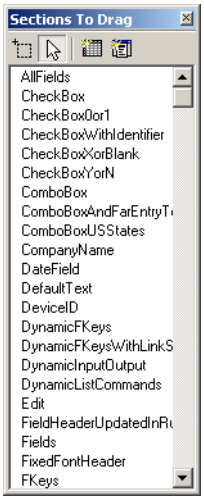


Figure 30. Sections to drag panel

Section Explanation Panel

The Section Explanation panel displays an explanation of the section that is currently selected in the Sections To Drag panel.

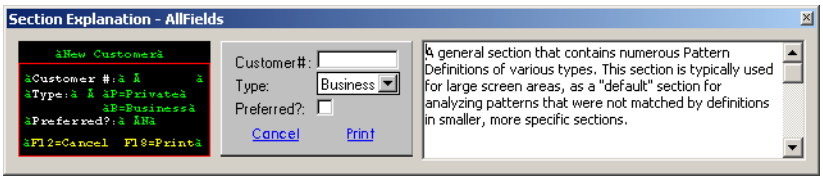


Figure 31. Section explanation panel

The information includes a textual description and two pictures. The left hand picture shows a host screen pattern that would be recognized by the selected section. The left hand picture also indicates where to place the section on such a host screen, including whether or not attributes should be included.

The right hand picture displays the GUI controls that would result from applying the selected section as in the left hand picture.

It is highly recommended that you browse through all the sections and view their explanations in order to have an idea of what options are available.

Show/Hide Sections to Drag and Section Explanation

You can show or hide Layout View's *Sections To Drag* and *Section Explanation* panels.

1. Open the *View* menu.
2. Choose *Palettes*.
3. Enable to show, or disable to hide, the *Sections To Drag* and *Section Explanation* panels.

If *Sections To Drag* and *Section Explanation* have been dragged to a new location, ACE remembers the new location.

Operations with Sections

In most cases, the *New Subapplication* wizard applies the correct sections to the correct regions of the host screen, and you do not need to change the arrangement of sections.

However, you should keep the following points in mind, and use the tools provided by Layout View to refine the arrangement of sections:

- Sections should cover only the characters that are appropriate for the section. When a section is not in the correct place, you can move it or change its size.
- When the wrong sections are on the screen image you can remove sections and add other sections.
- When the available sections do not include the correct mix of Pattern Definitions you can edit the existing sections and create new sections. You can also delete sections you have previously created.

Moving a Section on the Screen Image

To move a section on the screen image:

1. Go to Layout View.
2. Within the section you wish to move, click and hold down the mouse button. The section receives sizing handles and the cursor changes to a four-directional arrow.
3. Drag the section’s rectangle to its new position and release the mouse button.

Note: The point at which you click may be covered by more than one section rectangle. The smallest of the covering rectangles is the one selected. To select one of the larger rectangles, click a region not covered by any smaller rectangles.

Resizing a Section on the Screen Image

To resize a section on the screen image:

1. Go to Layout View.
2. Within the section you wish to resize, click the mouse button. The section receives sizing handles.
3. Place the cursor over one of the sizing handles. The cursor changes to a two headed arrow.
4. Click and drag the section's border to its new position, and release the mouse button.

Note: The point at which you click may be covered by more than one section rectangle. The smallest of the covering rectangles is the one that receives sizing handles. To select one of the larger rectangles, click a region not covered by any smaller rectangles.

Applying a Section to the Screen Image

You apply sections to a screen image either by using the *Sections to Drag* panel, through the *Layout* menu, or by using the *Insert* key.

To apply a section using the *Sections to Drag* panel:

1. From the Layout View *Sections to Drag* panel, select a section to apply to the screen image.
2. Click the button at the top of the *Sections to Drag* panel.

   You are in drag mode.

   The cursor changes to when pointing at the screen image.
3. Hold down the left mouse button and drag a rectangle around the screen area you want the section to cover. When you release the mouse button you see the section applied to the area covered by the rectangle you drew.
4. If necessary, move and resize the section.

To apply a section using the *Layout* menu:

1 Go to Layout View.

2 From the *Layout* menu select *Primary Sections* > *Add Section to Screen*. The *Section Definitions* manager opens.

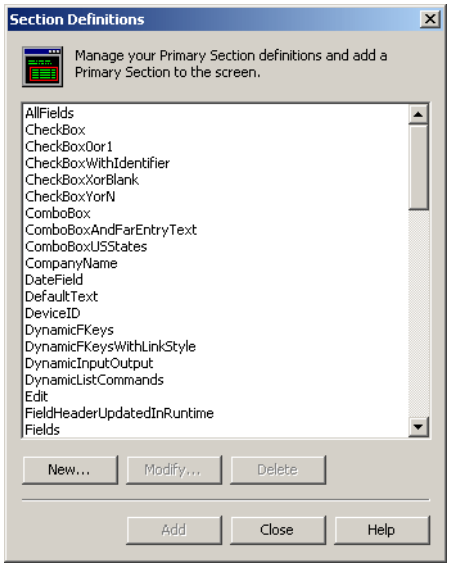


Figure 32. Section definitions manager

3 Select a section and click *Add*. The selected section is applied to the screen image.

4 Move and resize the section to cover the desired area.

To add a section using the *Insert* key:

1 Press the Insert key on the keyboard. The cursor changes to when pointing at the screen image.

2 Hold down the left mouse button and drag a rectangle around the screen area you want the section to cover. When you release the mouse button a shortcut menu opens.

3 From the shortcut menu select *Add primary section*. The Section Definitions manager opens.

4 Select a section and click *Add*. The selected section is applied to the screen image.

5 If necessary, move and resize the section.

Note: The *Section Definitions* manager can also be opened through the *Define* menu in the KnowledgeBase.

Removing Sections from the Screen Image

To remove a section from the screen image:

1. Go to Layout View.
2. Within the section you wish to delete, click the mouse button. The section receives sizing handles.
3. From the *Layout* menu select *Primary Sections* and select *Remove Selected Section From Screen* from the submenu.
   OR
   Press the *Delete* key on the keyboard
   OR
   Right click in the section area and select *Remove Section from Screen* from the submenu.
4. Confirmation is requested. Click *Yes*.

Note: The point at which you click may be covered by more than one section rectangle. The smallest of the covering rectangles is the one that receives sizing handles. To select one of the larger rectangles, click a region not covered by any smaller rectangles.

Replacing One Section with Another on the Screen Image

To replace a section on the screen image:

1. Go to Layout View.
2. Within the section you wish to replace either:
   Double click
   OR
   Right click and select Replace Section With from the popup menu.
   The *Section Definitions* manager opens.
3. In the *Section Definitions* manager click a section and then click *Replace*. The selected section replaces the target section on the screen image.

Using Wizards to Mark Menus or Lists on the Screen Image

You can mark menus and lists either by using wizards or by dragging the appropriate sections. Using wizards is the preferred method.

To mark a menu or list on the screen image using a wizard:

1. Go to Layout View.
2. Remove any menu or list sections that cover the area you wish to mark. Do not remove other sections such as AllFields.
3. In the *Sections to Drag* panel click

   to enter the *Menu* wizard

   -or-

   to enter the *List* wizard.
4. Follow the instructions in the wizard.

Editing Sections

Each section contains one or more Pattern Definitions. You can add new Pattern Definitions to a section, delete Pattern Definitions from a section and change the order of the Pattern Definitions within a section. Note that changing the order of Pattern Definitions within a section determines the order in which ACE selects Pattern Definitions for the analysis of that section.

To edit a section:

1. Go to Layout View
2. On the screen image double click within the section you wish to edit. The *Section Definitions* manager opens. The section you selected is already highlighted.
3. Click *Modify*. The *Primary Section Definition* dialog box opens.

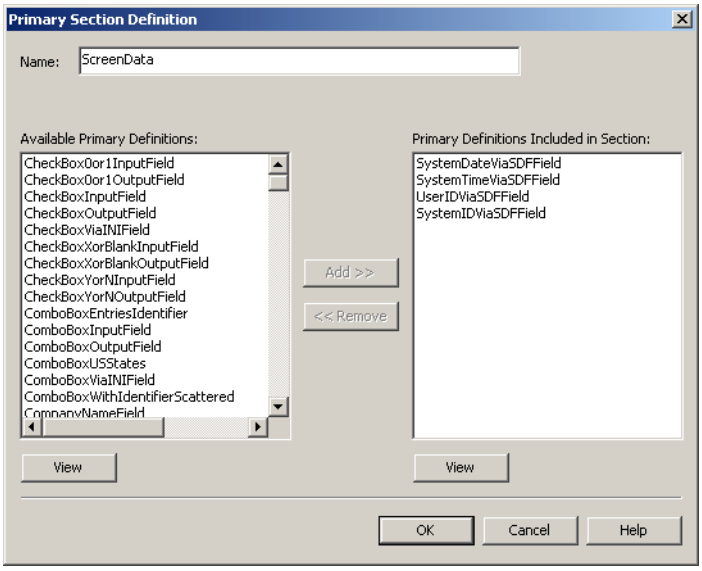


Figure 33. Primary section definition dialog box

The *Primary Definitions Included in Section* pane lists the Pattern Definitions in the section.

The *Available Primary Definitions* pane lists the available Pattern Definitions.

4 Add Pattern Definitions to the section by highlighting the Pattern Definition in the *Available Primary Definitions* pane and clicking the *Add* button.

5 Remove Pattern Definitions from the section by highlighting the Pattern Definition in the *Primary Definitions Included in Section* pane and clicking the *Remove* button.

6 Click *OK*.

Clicking a pane's *View* button opens the KnowledgeBase to the Pattern Definition highlighted in the pane.

Note: In addition to double clicking a section, you can access the *Primary Section Definition* dialog box in the following ways:

On the screen image, right click within the section you wish to edit. From the resulting shortcut menu select *Open Section's Definition*.

-or-

From the *Layout* menu select *Primary Sections* > *Edit Definitions*. The *Section Definitions* manager opens. Click the section you wish to edit and then click *Modify*.

-or-

In the KnowledgeBase *Define* menu, choose *Primary Sections*. The *Section Definitions* manager opens. Click the section you wish to edit and then click *Modify*.

Changing the order of Pattern Definitions is explained in "Sections and Pattern Definition Search Order" on page 100.

Creating New Sections

You can create a new section from the beginning or copy it from an existing section. New sections are added to the Layout View *Sections To Drag* panel.

To create a new section from the beginning:

1. Go to Layout View.
2. From the *Layout* menu select *Primary Sections* > *Edit Definitions* or double click anywhere on the screen image. The *Section Definitions* manager opens.
3. Click *New*. The *Primary Section Definition* dialog box opens.
4. In the *Name* field type a name for the section.
5. Follow the procedure for editing sections on page 97.

To create a new section from an existing section:

1. Follow the procedure for editing existing sections on page 97, but change the name in the *Name* field *before* you click *OK*.
2. Click *OK*. A new section is created and the original section is unchanged.

Deleting Sections

To delete a section from the KnowledgeBase:

1. Go to Layout View.
2. From the *Layout* menu select *Primary Sections* > *Edit Definitions*. The *Section Definitions* manager opens.
3. Highlight a section and click *Delete*. ACE prompts you to confirm the deletion.

   The section is deleted from the KnowledgeBase and is removed from the Layout View *Sections to Drag* panel.

Note: Only modified and user defined sections can be deleted. Sections supplied with ACE are protected and cannot be deleted unless modified. It is strongly recommended that you do not delete an ACE supplied section.

Sections and Pattern Definition Search Order

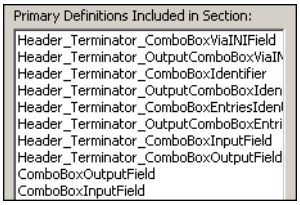
Chapter 4 - "Sections and Layouts" on page 77 explains how ACE establishes priority between sections that overlap on the host screen image. This section explains how to set the Pattern Definition search order within a section.

Pattern Definition Priority Within a Section

ACE searches the host screen for character sequences satisfying Pattern Definitions according to their order within the *Primary Definitions Included in Section* pane of the *Primary Section Definition* dialog box:

Example 8. Pattern definition priority within a section

The following figure shows part of the *Primary Definitions Included in Section* pane of the *Primary Section Definition* dialog box:



ACE searches a screen image area covered by the ComboBox section for the *Header_Terminator_ComboBoxViaIniField* Pattern Definition first. Then ACE searches this area for the *Header_Terminator_OutputComboBoxViaIniField* Pattern Definition, and so on.

Establishing the Order of Pattern Definitions

To change the order of a Pattern Definition already included in a section:

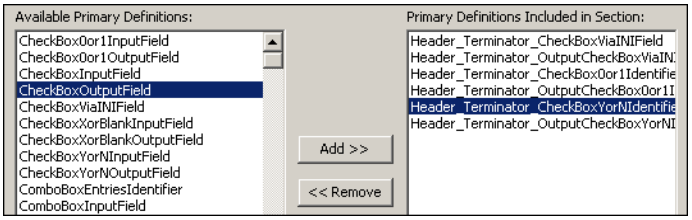
1. Go to Layout View.
2. From the *Layout* menu select *Primary Sections* > *Edit Definitions*.
3. In the *Section Definitions* manager select the desired section and click *Modify*. The *Primary Section Definition* dialog box opens.
4. From the *Primary Definitions Included in Section* pane remove the Pattern Definition you wish to reorder: Highlight that Pattern Definition and click *Remove*. Do not select any other Pattern Definition. Do not click OK.
5. Go to step 4 of the procedure for adding a new Pattern Definition in the correct position (procedure below). The Pattern Definition to add is the one you have just removed.

To add a new Pattern Definition to a section and place it in the correct position:

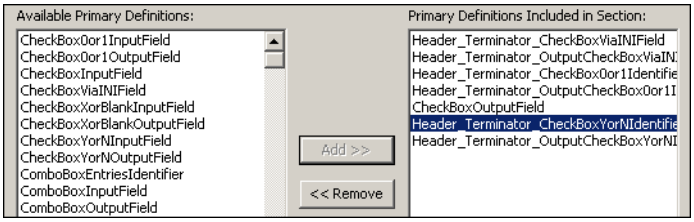
1. Go to Layout View.
2. From the *Layout* menu select *Primary Sections* > *Edit Definitions*.
3. In the *Section Definitions* manager dialog box select the desired section and click *Modify*. The *Primary Section Definition* dialog box opens.
4. In the *Available Primary Definitions* pane highlight the Pattern Definition you wish to add to the section.
5. In the *Primary Definitions Included in Section* pane highlight the Pattern Definition before which you wish to insert the new Pattern Definition. See example below. Skip this step to add the new Pattern Definition to the end of the list.
6. Click *Add*. The new Pattern Definition is added to the correct position in the *Section Pattern Definitions* panel.
7. Click *OK*.

Example 9. Establishing the order of pattern definitions

The following figure shows a section just before step 6. The Pattern Definition *CheckBoxOutputField* is about to be added to the section and placed immediately before the Pattern Definition *HeaderTerminatorCheckBoxYorNIdentifier*.



When the *Add* button is clicked, *CheckBoxOutputField* is added to the section. The result is:



Operations with Layouts

ACE allows you to apply a layout to the screen image or to delete a layout from the screen image. Modifying layouts is performed by applying a layout to the screen image and then using the tools for arranging sections.

Removing a Layout from the Screen Image

To delete a layout from the screen image:

1. Go to Layout View.
2. From the *Layout* menu select *Clear Current Layout*. A confirmation dialog box opens.
3. Click *Yes*. The layout is deleted from the screen image.

Applying a New Layout to the Screen Image

To apply a layout from the screen image:

1. Go to Layout View.
2. From the *Layout* menu select *Change Layout*.

   If there are any sections on the screen image you are asked whether you wish to clear the current layout. Click *Yes*.

   The *Open Layout* dialog box opens.

3 Select a layout from the list and click *OK*.
   OR
   Double click a layout from the list.
4 The selected layout is applied to the screen image.

Editing Existing Layouts

To edit an existing layout and save it to the KnowledgeBase:

1 Go to Layout View.
2 Open a subapplication that uses the particular layout.
   OR
   Apply the layout to an appropriate subapplication. From the *Layout* menu choose *Change Layout*.
3 Change the arrangement of sections on the screen image as desired.
4 From the *Layout* menu choose Update Layout. The layout is saved to system memory and can be used in the current session.
5 To save the layout permanently you must save the Application. From the *File* menu select Save All.

Note: Saving your layout to system memory allows you to try different layouts and examine the analysis after each change without destroying the existing layout. Only when the layout is perfected, and you think it can be used to analyze other screens, should you save it permanently by saving the Application.

Creating New Layouts

To create a new layout and save it to the KnowledgeBase:

1 Go to Layout View.
2 Change the arrangement of sections on the screen image as desired.
3 From the Layout menu choose Save Layout As. The Save Screen Layout As dialog box opens.
4 In the edit field, type a name for the new layout. If you wish to overwrite an existing layout, select the existing layout from the list.
5 Click *OK*. The layout is saved to system memory and can be used in the current session.
6 To save the layout permanently you must save the Application. From the *File* menu select *Save All*.

Writing Layouts to INI files

You can save screen layouts to an INI file, as well as saving them to the database. You can also read screen layouts from an INI file, instead of from the database. This is useful in multi-developer environments where an Application is divided into libraries.

- To write Subapplication layouts to an INI file, in SPECIFIC.INI set:

  ```
  [Converter]
  WriteLayoutsToIni=1
  ```

  (the default is 0—layouts are not written toSPECIFIC.INI)
- To read layouts from an INI file, in SPECIFIC.INI set:

  ```
  [Converter]
  ReadLayoutsFromIni=1
  ```

  (the default is 0—layouts are read from the database).

Filter Section Operations

Filter sections represent information derived from screen definition files (SDF). Therefore, filter sections are available only for host screen images derived from screen definition files. When displayed, filter sections appear on the screen image as dashed rectangles, with the name of the filter section in the center of the rectangle.

Example 10. Filter section operations

The following screen area contains three fields, each composed of a static header followed by an input field:

```
Customer #:    123456789
First Name: John
Last Name: Smith
```

The above screen area is taken from a host screen image derived from a screen capture, and it does not have filter sections.

The example below shows the same screen area taken from a host screen image derived from a screen definition file (SDF), with filter sections displaying:

Each static header is covered by a Static filter section, and each input field is covered by an Input filter section.

A filter section contains only pattern definitions that are suitable to its type. For example, the Static filter section contains pattern definitions that match static patterns such as headers. In the analysis process, filter sections are used in addition to primary sections, and they help 'filter out' incorrect matchings of a pattern definitions to a pattern.

Displaying Filter Sections

Filter sections can be displayed in Layout View. By default, filter sections are not displayed. However, you can toggle the display of filter sections as necessary.

To toggle the display of filter sections:

1. Got to Layout view
2. From the *View* menu select *Customize*. The submenu item *Show Filter Sections* appears.
3. Toggle the display of filter sections:
   If filter sections are currently hidden, *Show Filter Sections* is not checked. Click *Show Filter Sections* to display filter sections.
   If filter sections are currently visible, the item *Show Filter Sections* is checked. Click *Show Filter Sections* to hide filter sections.

Working with Filter Sections

ACE with SDF is supplied with a series of defined filter sections accessed through the *Layout* menu:

To work with filter sections:

1. In Layout View open the *Layout* menu.
2. Select *Filter Sections*. A submenu of filter section tools opens.

Note: Select the appropriate item from the submenu. The choices are functionally identical to the choices for regular sections, and the corresponding tools operate identically. The *Filter Definitions* manager can also be opened through the *Define* menu in the KnowledgeBase.

Removing Filter Sections

To remove a filter section from the screen image:

1. Go to Layout View.
2. Within the section you wish to delete, click the mouse button. The section receives sizing handles.
3. From the *Layout* menu select *Filter Sections* and select *Remove Selected Section From Screen* from the submenu.
   OR
   Press the *Delete* key on the keyboard
   OR
   Right click in the section area and select Remove Section from Screen from the shortcut menu.
4. Confirmation is requested. Click *Yes*.

BMS Filter Sections for Mainframes

Table 4 lists the filter sections that exist in the default KnowledgeBase for BMS files attained from the MainFrame.

Table 4. Filter sections derived from BMS files

Filter Section	Description
Background	Any area on the screen that is not a specific field. The filter section is not visible on the screen image.
Input	Input fields
Output	Output field
Static	A constant
List	A table area
ListColumn_Input	A list column that is an input field
ListColumn_Output	A list column that is an output field
ListColumn_Static	A list column containing a constant

DDS Filter Sections for AS/400s

Table 5 lists the filter sections that exist in the default KnowledgeBase for DDS files attained from the iSeries.

Table 5. Filter sections derived from the iSeries (Sheet 1 of 2)

Filter Section	Description
Background	Any area on the screen that is not a specific field. The filter section is not visible on the screen image.
Input	Input fields
Values	An input field that has a “values” keyword that specifies the values that you can type in the field
Output	Output field
Date	A field containing the date from the AS/400
User	A field containing the name of the AS/400 user
Time	A field containing the time from the AS/400
Static	A constant or a field with a MSGID keyword
List	A subfile area
ListColumn_Input	A list column that is an input field
ListColumn_Values	A list column that has a “values” keyword that specifies the values that you can type in the field
ListColumn_Output	A list column that is an output field
ListColumn_Date	A list column containing the date from the AS/400
ListColumn_User	A list column containing the name of the AS/400 user

Table 5. Filter sections derived from the iSeries (Sheet 2 of 2)

Filter Section	Description
ListColumn_Time	A list column containing the time from the AS/400
ListColumn_Static	A list column containing a constant or a field with a MSGID keyword
SysName	A field containing the system name taken from the AS/400

Menus Specific to Layout View

When you switch to Layout View in ACE, the Layout menu item on the ACE standard toolbar becomes available. This sections deals with ACE features available via the Layout menu.

Layout Menu

The options in Table 6 are available via the *Layout* menu.

Table 6. The Layout menu (Sheet 1 of 2)

Option	Description
Clear Current Layout	Clears the current layout.
Change Layout	Clears the current layout and opens the list of the existing layouts.

Table 6. The Layout menu (Sheet 2 of 2)

Option	Description
Update Layout	Saves the current arrangement of sections as a layout with the same name as the most recently applied layout. If no layout was applied, or if the previously applied layout was cleared from the screen image at some point, you are prompted to supply a name for the layout. The layout is saved to the system memory. You must save the Application to save the layout permanently.
Save Layout As	Saves the current arrangement of sections as a layout. You are prompted to supply a name for the layout. The layout is saved to the system memory. You must save the Application to save the layout permanently.
Primary Sections	Opens a submenu containing tools for arranging sections on the screen image and tools for editing sections.
Filter Sections	Opens a submenu containing tools for arranging filter sections on the screen image and tools for editing filter sections.

Primary/Filter Section Submenu

The options in Table 7 are available via *Layout* > *Primary/Filter Section* menu.

Table 7. Primary/Filter Section menu options (Sheet 1 of 2)

Option	Description
Add (Filter) Section to Screen	Opens the *Section/Filter Definitions* manager where you can choose a section to add to the current layout.
Remove Selected (Filter) Section From Screen	Removes the selected section from the layout.

Table 7. Primary/Filter Section menu options (Sheet 2 of 2)

Option	Description
Replace Selected (Filter) Section With	Removes selected section from the layout and opens the *Section/Filter Definitions* manager where you can choose a section that will replace the selected section in the current layout.
Open Selected Section's Definition	Open the selected section's *Primary/Filter Sections Definition* dialog box.
Edit Definitions	Opens the *Section/Filter Definitions* manager.

Chapter 6. Operations Performed in Analysis View

In Analysis View, ACE analyzes your host screen by comparing the patterns on the screen to the Pattern Definitions in the ACE KnowledgeBase. After it has identified the patterns, ACE displays a color-coded breakdown of the host screen. Most of the patterns that ACE identifies have a GUI Representation Definition attached to them. This defines the look of your GUI window.

A correct analysis is a critical aspect of your conversion. The more accurately your system analyzes your screens through the automatic mechanisms of the KnowledgeBase, the faster and simpler will be your conversion.

In Analysis View you can select a Pattern Definition, identify some of its properties and gain direct access to that Pattern Definition in the KnowledgeBase. In addition, you can perform a variety of operations on lists in this View.

This chapter describes:

- The Analyzed Screen
- Tasks You Can Perform in Analysis View

The Analyzed Screen

Figure 34 shows a screen image in Analysis view.

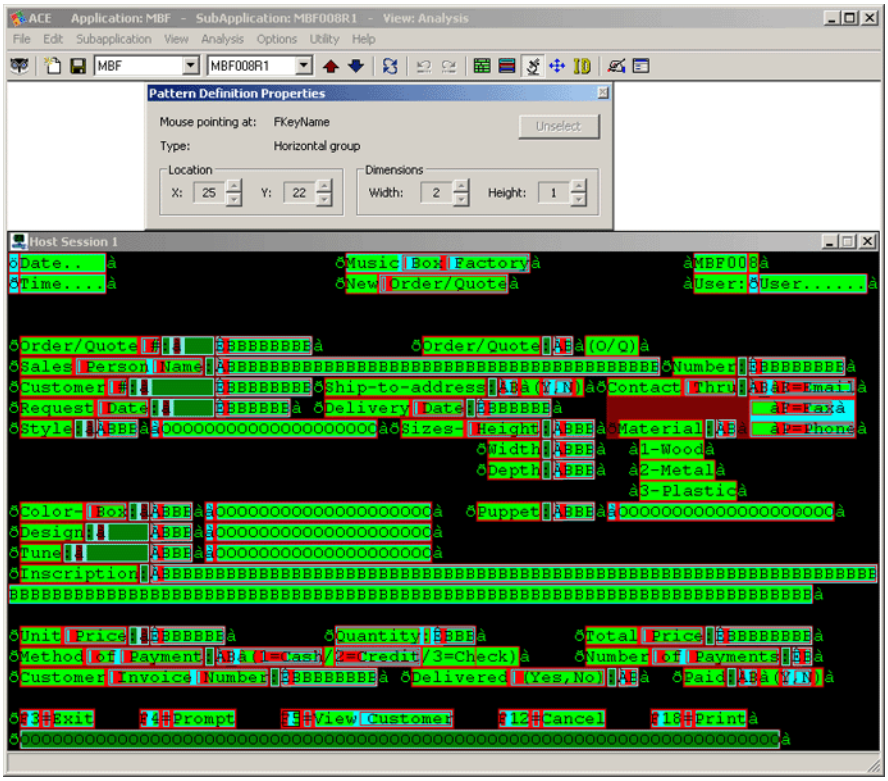


Figure 34. A screen image in analysis view

In Analysis View the *Analysis* menu is enabled on the Menu Bar. The *Pattern Definition Properties* dialog box is displayed at the top of the screen.

ACE analyzes the host screen based on the layout that you have chosen, and breaks down the screen into character groups recognized by Pattern Definitions. The breakdown is displayed in color-coded format. The KnowledgeBase contains several hundred Pattern Definitions, thus most of the character groups on your screen are classified with great precision.

The Coloring Scheme

In Analysis View, ACE color-codes the entire screen. When the analysis is complete, the entire screen is covered with colored rectangles. A function key such as Exit, appears as follows in the analyzed color-coded screen.

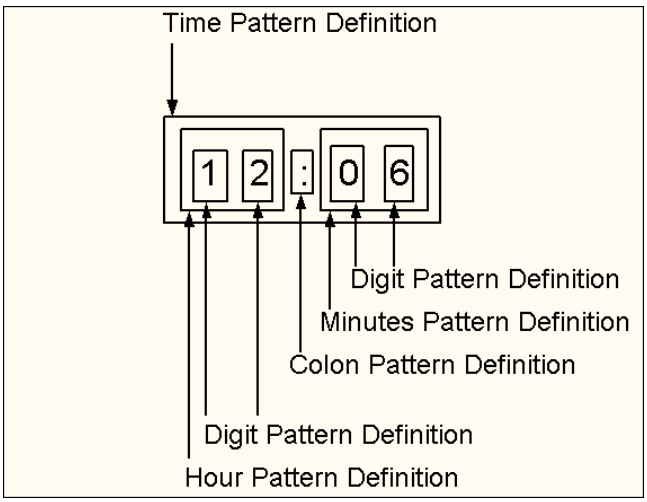
Figure 35. The coloring scheme

The coloring scheme describes the levels of the definitions and their types. Each rectangle in the screen is layered, and contains embedded rectangles, representing lower level definitions. Primary definitions appear as rectangles with multi-colored borders. The innermost rectangles define the lowest level (basic) definitions. The adjacent colored border represents the next higher level definition, and so forth until the outermost border.

Example 11. The coloring scheme

The Time Pattern Definition illustrates the embedded color-coding scheme. When the host screen displays the time: 12:06

ACE displays the analysis in the following manner:



ACE presents each of these elements as a separate color.

Note: The *Time* Pattern Definition used here is merely an example of how a Pattern Definition of this type may be constructed. It does not exist in the KnowledgeBase.

Tasks You Can Perform in Analysis View

The following tasks can be performed in Analysis View:

- Move the focus through the Hierarchy of a Pattern Definition
- Ignore a Pattern Definition
- Modify a Pattern Definition through the KnowledgeBase
- Modify the location or dimension of a Pattern Definition
- Use a Pattern Definition as an example
- Copy a string to the clipboard
- Edit a list (for detailed information refer to *JIS Interface Server: Advanced Topics*)

Moving the Focus Through the Hierarchy of a Pattern Definition

When you move your mouse across the analyzed screen, the pattern that comes into focus is marked by a thin black border.

To view the hierarchy of Pattern Definitions that recognized a character:

- Place the cursor over the center of the character to view the lowest pattern definition in the hierarchy of recognizing Pattern Definitions.
- Move the cursor towards the top or bottom of the character to go up the hierarchy of recognizing Pattern Definitions.
- You can also ascend the hierarchy by pressing the *F5* key.
- You can also descend the hierarchy by pressing the *F6* key.

To select a Pattern Definition, shift the focus to the Pattern Definition and click the left mouse button or press the *F7* key.

Information concerning the Pattern Definition currently in focus is displayed in the *Pattern Definition Properties* dialog box.

The Pattern Definition Properties Dialog Box

The *Pattern Definition Properties* dialog box displays information on the patterns recognized in the screen and provides modifying options for perfecting the analysis. The *Pattern Definition Properties* dialog box is displayed in two modes; Select and Unselect.

Select Pattern Definition

As long as no Pattern Definition has been selected, the analyzed screen is dynamic. This means that you can move the mouse across the analyzed screen and the information displayed in the dialog box fields changes according to the definition level the cursor is pointing at. The name of the Pattern Definition the cursor is standing on appears in the *Mouse pointing at* field. Note that the *Unselect* button is disabled.

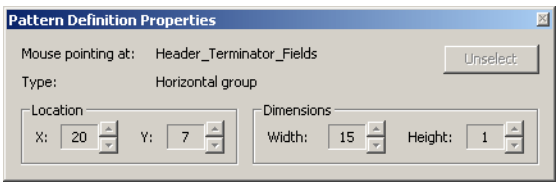


Figure 36. Dynamic analyzed screen

Unselect Pattern Definition

Clicking on a Pattern Definition "freezes" the analyzed screen; moving the mouse across the analyzed screen has no effect. The name of the selected Pattern Definition is displayed in the *Selected pattern* field.

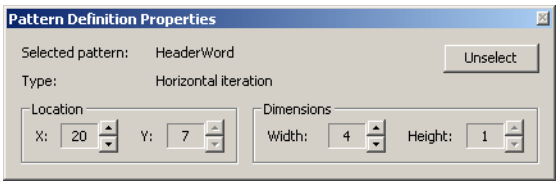


Figure 37. Frozen analyzed screen

When working in this mode you can change a Pattern Definition's location and dimension by using the *Location* and *Dimensions* spin boxes.

To select another Pattern Definition, click on it. Clicking the Unselect button returns the analyzed screen to Select mode.

In both modes, the following information and options are available:

Type	As the Pattern Definition changes, the type changes to reflect the pattern type in focus.

Location and Dimensions

The location and dimension of the Pattern Definition can be modified using the arrow buttons.

The arrow buttons work only when a Pattern Definition has been selected.

Shortcut Keys in Analysis View

The shortcut keys in Table 8 are at your disposal in Analysis View.

Table 8. Shortcut keys in Analysis View

Shortcut	Function
F2	Clear selected Pattern Definition. The analyzed screen returns to its dynamic state.
F5	Mark higher level Pattern Definition in analyzed screen. Moves the current black border to surround the next Pattern Definition. The movement is an *outward* movement, to the next higher level Pattern Definition. The limit is reached when the black border outlines the primary Pattern Definition.
F6	Mark lower level Pattern Definition in analyzed screen. Moves the current black border to surround the previous Pattern Definition. The movement is an *inward* movement, to the previous lower level Pattern Definition. The limit is reached when the black border outlines the lowest level Pattern Definition.
F7	Select Pattern Definition in analyzed screen. Equivalent to clicking the left mouse button; the Pattern Definition the mouse is currently pointing at is selected.
F8	Modify Pattern Definition. Opens the *KnowledgeBase Definitions-Pattern Definition View* displaying the Pattern Definition the mouse is currently pointing at in the analyzed screen.

Ignoring a Pattern Definition

You may find that the system has analyzed a pattern on the screen incorrectly. Use the *Ignore* command to inform the system that this Pattern Definition is incorrect. ACE will then try to find a different Pattern Definition that matches the pattern.

Example 12. Ignoring a pattern definition

Let us assume that the following pattern appears in a screen:

```
The new code is 12:06
```

The Analysis might match a *Time* Pattern Definition with the pattern `12:06`. However, it is clear that a *Time* Pattern Definition is incorrect in this case. Choosing *Ignore* causes the Analysis to reject the *Time* Pattern Definition, and look for another suitable Pattern Definition.

This Pattern Definition will be ignored in any future analysis of this screen section.

To ignore a pattern analysis:

1. Select the Pattern Definition to be rejected.
2. Click the right mouse button and select *Ignore*.
3. The colors of the ignored Pattern Definition and all its lower level definitions disappear. Either new Pattern Definition colors are displayed, or the background colors green on black are displayed if there is no Pattern Definition underneath.
4. From the *Analysis* menu choose *Apply Analysis Changes*. The system analyzes the screen again and displays the updated modifications.
5. To return to the original analysis, from the *Analysis* menu choose *Restart Analysis*.

Modifying the Location or Dimension of a Pattern Definition

When a pattern has been matched with the correct Pattern Definition but you would like to modify the dimension or location, use the following procedure:

1. Select the pattern to be modified.
2. Use the *Location* or *Dimensions* arrow buttons to obtain the desired modification.
3. The colors on the screen change as higher level Pattern Definitions are revealed or concealed.

Note: Modification should be done with caution and good judgment because an imprudent change may result in overwriting another important pattern.

4 To view your modifications, from the *Analysis* menu choose *Apply Analysis Changes*. ACE presents the improved analysis.
5 You can return to the original analysis at any stage by choosing *Restart Analysis*.

Modifying a Pattern Definition

You may wish to view or make changes to a specific Pattern Definition's properties in the KnowledgeBase.

To access the Pattern Definition View in the *KnowledgeBase Definitions* window:

1 Select a Pattern Definition in the analyzed screen.
2 Click the right mouse button and click *Modify*. The *Pattern Definitions View* dialog box opens and displays the selected Pattern Definition. Alternatively, point at a Pattern Definition and press *F8* to access the *Pattern Definition View* dialog box.

Setting an Example

For each Pattern Definition you can create a visual example of the type of pattern that it recognizes on the screen. Use this feature to illustrate the scope of a new Pattern Definition or to understand the function of an existing Pattern Definition.

To set an example:

1 In your analyzed screen, delimit the area you wish to use as an example by dragging a rectangle around it, using the left mouse button.
2 In the rectangle, right-click a Pattern Definition that you wish to illustrate.
3 From the shortcut menu, choose *Set as example*.
4 From the *File* menu choose *Save All*.

To view an example:

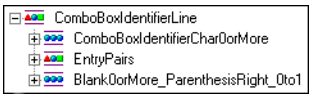
1 Enter the *KnowledgeBase Definitions-Pattern Definitions View* dialog box by double-clicking on the pattern you have chosen or by clicking the KnowledgeBase icon.

2 In the Upper Pattern pane, select the Pattern Definition whose example you wish to view.
3 In the Upper Properties pane, choose the *Extended Info* tab. The example is displayed.

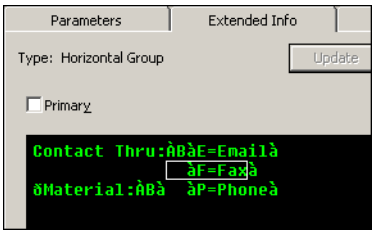
Note: The pattern that satisfies the Pattern Definition you have selected is delimited by a rectangle.

Example 13. Setting an example

In the host screen a pattern satisfying the *ComboBoxIdentifierLine* Pattern Definition has been marked to be used as an example.



The following example illustrates a pattern that satisfies this Pattern Definition.



Copying Text to the Clipboard

ACE enables you to copy strings from the host screen to the clipboard. You can then paste them where necessary during your editing operations.

To copy a string into the clipboard:

1 With the left mouse button, mark a rectangle around the string you wish to copy.
2 Click the right mouse button and select *Copy text to clipboard*.

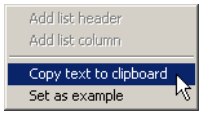


Figure 38. Copying text via the RMB

Editing a List

In addition to the functions described in this chapter, Analysis View is also used to refine a list's analysis. For example, you can manipulate the identification of host screen list headers, list columns, and header-column links. For more information on working with lists in Analysis View, see the section entitled "Arranging List Columns and Headers Manually" in *JIS Interface Server: Advanced Topics*.

Chapter 7. Pattern Definition Modifications Through the KnowledgeBase

The KnowledgeBase contains several hundred Pattern Definitions. These Pattern Definitions are adequate to analyze most host screens correctly. You may find that your Application contains specific situations that are not handled by the existing KnowledgeBase Pattern Definitions. In such a case, you can correct ACE's analysis through modifications made to the KnowledgeBase; by adding new Pattern Definitions, or changing existing ones.

Although it is very important to add and edit Pattern Definitions, exercise caution when you do this. Changes that you make to the KnowledgeBase are global changes that affect the entire conversion process. Changes that you make to the Pattern Definitions affect all the screens in your entire Application. Generally, you should modify the KnowledgeBase when a situation requiring special treatment occurs on many screens.

This chapter describes:

- About Working with the KnowledgeBase
- Editing Pattern Definition Structure
- Editing Pattern Definition Properties

About Working with the KnowledgeBase

ACE allows you to view or edit an existing Pattern Definition, and to add new Pattern Definitions to the KnowledgeBase.

This section explains how to view Pattern Definitions in the KnowledgeBase Definitions window.

To access the Pattern Definitions, do one of the following:

- Click the KnowledgeBase icon on the *Menu* bar to open the *KnowledgeBase Definitions-Pattern Definitions View* dialog box.
- In Analysis View select any analyzed character sequence and either:
  - Double-click, or
  - Press the *F8* key

Overview of the KnowledgeBase Definitions Window

You use the *KnowledgeBase Definitions* window to work with:

Pattern Definitions	View Pattern Definitions and their complete child pattern substructure; create new patterns and modify existing patterns. You can also view and modify Pattern Definition parameters.
Representation Definitions	View Representation Definitions; create new representations, modify existing representations and attach them to Pattern Definitions. You can also view and modify Representation Definition parameters.
Message Definitions	View Message Definitions and designate Pattern Definitions as Message Definitions.

The Parts of the KnowledgeBase Definitions Window

The KnowledgeBase Definitions window contains seven main areas:

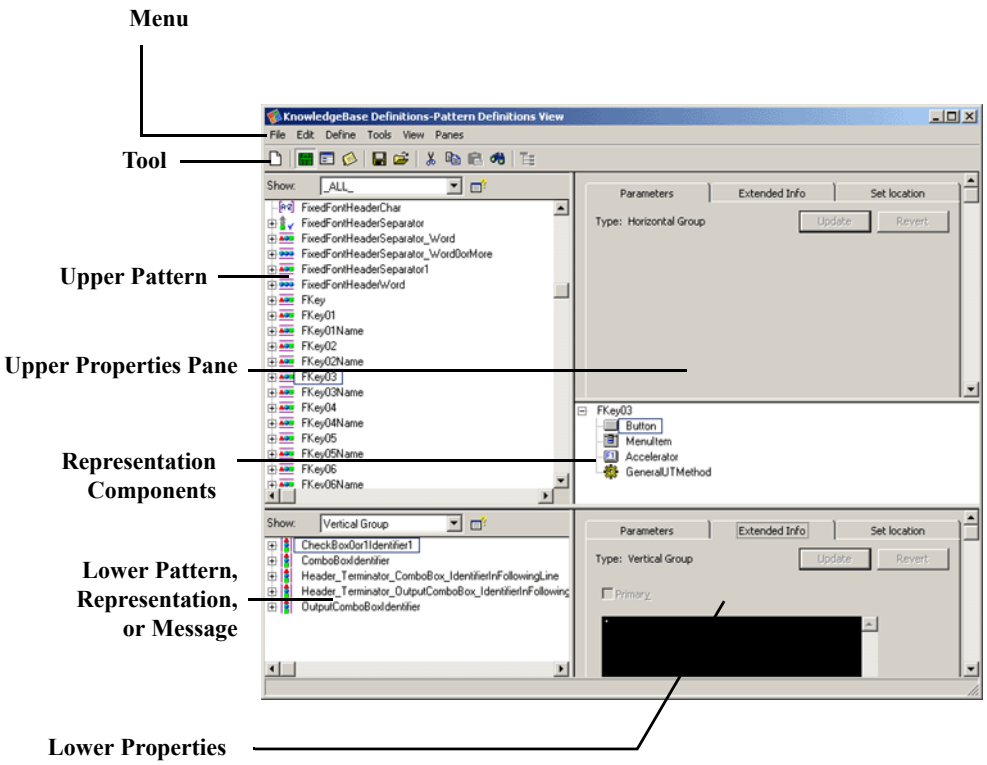


Figure 39. KnowledgeBase Definitions window

Menu Bar

The *Menu* bar includes all the functions you use to edit, view and manage the KnowledgeBase Pattern Definitions and Representation Definitions.

Tool Bar

The *Tool* bar gives you fast access to the most commonly used tools.

Upper Pattern Pane

The *Upper Pattern* pane provides a view of the Pattern Definitions' structure, as well as a work area for "drag and drop" Pattern Definition editing. The properties of a selected Pattern Definition appear in the *Upper Properties* pane.

Upper Properties Pane

The *Upper Properties* pane is where you view and edit the characteristics of selected Pattern Definitions and selected Representation Components.

Representation Components

The *Representation Components* area displays the representation components attached to a Pattern Definition selected from the *Upper Pattern* pane, and is where you select a representation component for editing in the *Upper Properties* pane.

Lower Pattern, Representation, or Message Pane

The *Lower Pattern, Representation, or Message Definition* pane provides a view of the complete structure of the Pattern Definitions, Representation Definitions, or the Pattern Definitions that are designated as Message Definitions.

- In Pattern Definitions View it provides a source reservoir for upper pane "drag and drop" editing.
- In Representation View it is a work area for "drag and drop" representation editing. The properties of representation components selected in the pane appear in the *Lower Properties* pane.
- In Message View it lists those Pattern Definitions that have been designated as Message Definitions. Selecting a Message Definition in the *Message* pane allows you to quickly find it in the *Upper Pattern* pane where you can edit it.

Lower Properties Pane

The *Lower Properties* pane is where you view the characteristics of selected Pattern Definitions and where you view and edit the characteristics of selected representation components.

Pane Operations

- You can set the relative sizes of the upper and lower panes by dragging the horizontal divider.
- Within the upper pane, you can set the relative size of the *Pattern* pane with respect to the *Properties* tabs and the *Representation Components* area by dragging the vertical divider.
- You can set the relative sizes of the upper pane *Properties* tabs area and *Representation Components* area by dragging the horizontal divider.
- Within the lower pane, you can set the relative sizes of the *Pattern, Representation or Message* pane and the *Properties* tabs area by dragging the vertical divider.

- Switch the lower pane to Pattern Definition View by choosing *Pattern View* from the *View* menu, or by clicking the *Pattern Definitions View* button.
- Switch the lower pane to Representation Definition View by choosing *Representation Definitions View* from the *View* menu, or by clicking the *Representation View* button.
- Switch the lower pane to Message Definition View by choosing *Messages View* from the *View* menu, or by clicking the *Message Definitions View* button.

Viewing Pattern Definitions

Access the *KnowledgeBase Definitions* window in Pattern Definition View by clicking the *KnowledgeBase* icon. The Pattern Definitions appear in the *Pattern* pane.

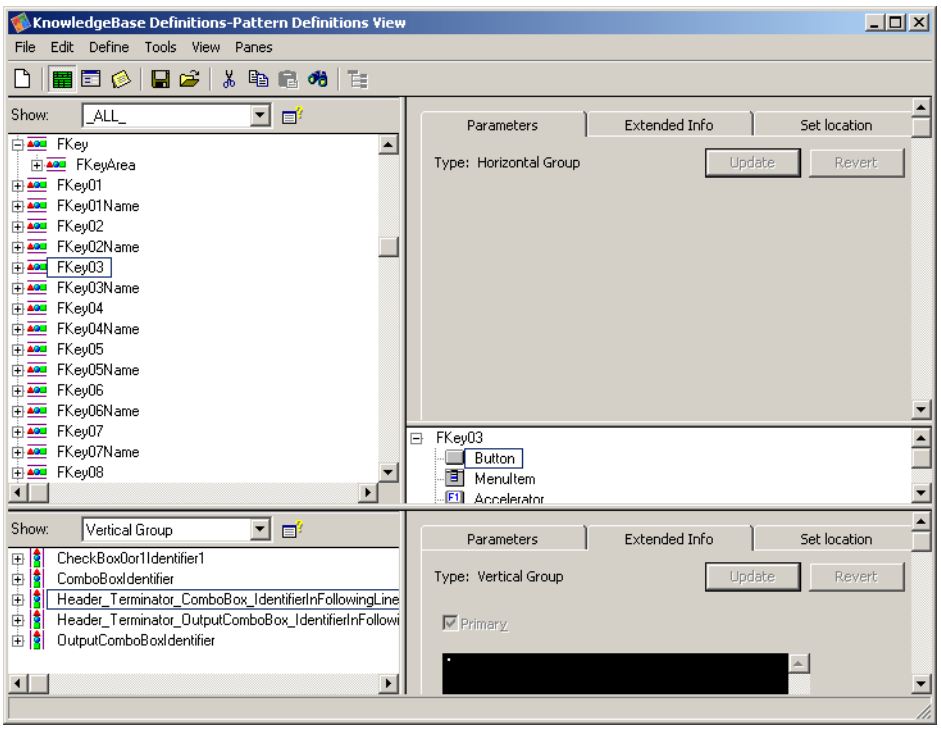


Figure 40. Pattern Definition View in the KnowledgeBase Definitions Window

The *Pattern Pane* displays all the Pattern Definitions in the KnowledgeBase.

Viewing the Pattern Definition Hierarchy

To view the Pattern Definition hierarchy:

- All the Pattern Definitions in the KnowledgeBase appear as top level Pattern Definitions, with the Pattern Definition's identifying icon and the Pattern Definition's name flush against the left side of the *Pattern* pane.

- A Pattern Definition may also appear as a child pattern of another Pattern Definition.
- When ⊞ appears beside the Pattern Definition's icon it means that the Pattern Definition has child patterns, but that these child patterns are not displayed. In the previous figure, the Pattern Definition FKeyArea has child patterns but these Pattern Definitions are not displayed.
- When ⊟ appears beside the Pattern Definition's icon it means that the Pattern Definition has child patterns, and that these child patterns are displayed. In the previous figure, the Pattern Definition FKey has a child pattern and it is displayed.
- The ⊟ symbol refers only to the next level of Pattern Definitions.

Icons

Every Pattern Definition appears with an icon identifying the Pattern Definition's type.

The following list presents the meaning of the icons used in the Pattern Definition tree diagrams.

Symbol	Pattern Type	Symbol	Pattern Type
	Character Set		String
	Dynamic Group		Horizontal Group
	Dynamic Iteration		Horizontal Iteration
	List		One Of
	List Column		Popup Border
	List with Parameters		Scattered Group
	Vertical Group		Vertical Iteration

Expanding and Collapsing the Pattern Definition Hierarchy

You can view all, some, or none of the child pattern levels of a displayed Pattern Definition.

To expand a Pattern Definition by one child level:

Click the ⊞ symbol that is beside the Pattern Definition's icon. The ⊞ symbol changes to ⊟ and the next level of child patterns is displayed.

To expand all child levels of a Pattern Definition:

1. Click the Pattern Definition you wish to expand.
2. From the *View* menu select *Expand All.* All the levels of the selected Pattern Definition are displayed.

   -or-

   Right click the Pattern Definition you wish to expand.
3. From the shortcut menu select *Expand All.* All the levels of the selected Pattern Definition are displayed.

To collapse all child levels of a Pattern Definition:

Click the ⊟ symbol that is beside the Pattern Definition's icon. The ⊟ symbol changes to ⊞ and none of the Pattern Definition's child patterns are displayed.

Displaying a Pattern Definition's Parent Patterns

Use the *Parent Patterns* feature to display all parent patterns of a particular Pattern Definition. This feature is useful for evaluating the effects of modifying a particular Pattern Definition.

To display a Pattern Definition's parent patterns:

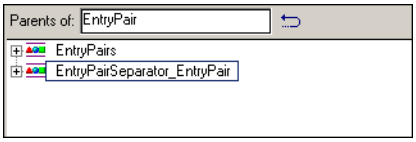
1. In either *Pattern* pane select the Pattern Definition whose parents you wish to display.
2. Click the *Parent Patterns* button on the *Tool* bar.

-or-

3. In either *Pattern* pane right click the Pattern Definition whose parents you wish to display and click *Parent Patterns*.

Example 14. Displaying a pattern definition's parent patterns

The figure shows the result of displaying the parent patterns of Pattern Definition EntryPair.



- The *Pattern* pane shows only the Pattern Definitions that have *EntryPair* as a child pattern. These Pattern Definitions are *EntryPairSeparator_EntryPair* and *EntryPairs*.
- The area above the *Pattern* pane informs you that the displayed Pattern Definitions all contain *EntryPair*.

The *Back* button appears above the *Pattern* pane. Use the *Back* button to return the *Pattern* pane to its previous state.

Navigating the Pattern Panes

You can move through a *Pattern* pane in two ways:

- Use the scroll bars to scroll through a *Pattern* pane. Scrolling does not affect which pattern is selected.
- You can put the focus on the *Pattern* pane and then type a letter to jump to the closest Pattern Definition whose name begins with that letter. The Pattern Definition that is jumped to becomes the selected pattern.

Selecting a Pattern Definition by Name

Use the Find feature to select a Pattern Definition by name, or to select the closest Pattern Definition whose name begins with a particular sequence of letters.

To select a Pattern Definition by name:

1. Put the focus on a *Pattern* pane.
2. From the *Edit* menu choose *Find*.

   -or-

   Click the *Find* button on the Tool Bar.The *Find* window opens.

Figure 41. Find dialog box

3 In the *Find* window type the name of the desired Pattern Definition, or the sequence of letters the Pattern Definition name should begin with. As you type, the focused *Pattern* pane jumps to the nearest Pattern Definition whose name begins with the letters you have already typed.
4 Press *Enter* or *Esc*. The *Find* window closes and the pattern that is jumped to becomes the selected pattern.

Example 15. Selecting a pattern definition by name

Type `Fa` in the *Find* window:

The Pattern Definition *FailPattern* is selected in the focused *Pattern* pane.

Selecting the Same Pattern Definition in Both Pattern Panes

When a Pattern Definition is visible in one *Pattern* pane you can select and display the same Pattern Definition in the other *Pattern* pane.

Use this feature to make a Pattern Definition visible in preparation for drag-and-drop editing.

To jump to the same Pattern Definition in the other Pattern pane:

1 Right click a Pattern Definition visible in either *Pattern* pane.
2 From the pop-up menu choose *Jump to definition in upper/lower pane*. The top level occurrence of the Pattern Definition becomes visible and selected in the other *Pattern* pane.

Filtering a Pattern Pane

You may filter a *Pattern* pane to display only those Pattern Definitions that meet certain criteria. These criteria may include such things as:

- Pattern Definition type.
- Parts of a Pattern Definition's name.
- The Representation Definition attached to the Pattern Definition.

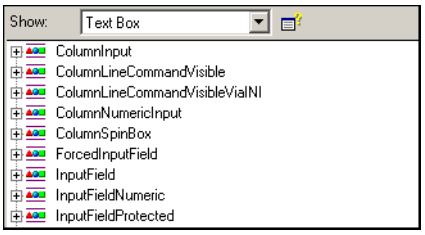
Using Display Criteria to Filter the Pattern Definition Pane

You can filter the Pattern Definitions displayed in the *Pattern* pane by applying a criterion to the *Pattern* pane. When you apply the criterion, the *Pattern* pane displays only the Pattern Definitions that meet the criterion. Select a criterion to apply from the *Show* combo box.

Example 16. Using display criteria to filter the pattern definition pane

Choose Text Box from the *Show* combo box. The *Pattern* pane displays only those Pattern Definitions that have a text box as attached Representation Component.

The result is:



To display all the Pattern Definitions, choose _*ALL*_ from the *Show* combo box.

Displaying Existing Criteria

Existing criteria are displayed in the *Display Criteria Setup* window.

To open the Display Criteria Setup window:

1. Click the button in either of the *Pattern Panes*;
2. The *Display Criteria Setup* window opens.

The Display Criteria Setup Window

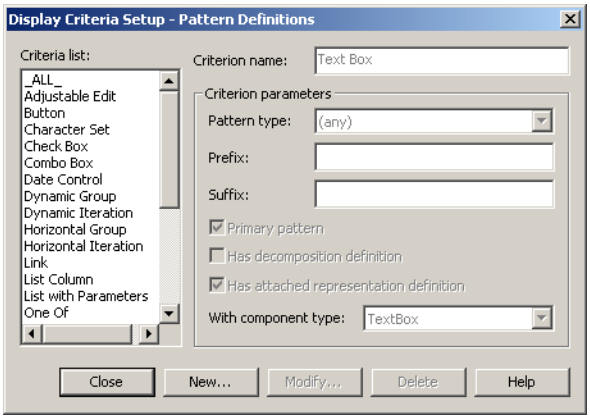


Figure 42. Display Criteria Setup window

The *Display Criteria Setup* window contains the following fields:

Table 9. Display Criteria Setup window fields (Sheet 1 of 2)

Criteria	Description
Criteria List	This list contains the names of all the defined criteria. Selecting a name from this list inserts its values in the other fields.
Criterion Name	The name of the criterion.
Pattern Type	The criterion includes either only the specified Pattern Definition type or any Pattern Definition type.
Prefix	The criterion includes only those Pattern Definitions whose name begins with the specified prefix.
Suffix	The criterion includes only those Pattern Definitions whose name ends with the specified suffix.
Primary pattern	The criterion includes only Primary Pattern Definitions.
Has decomposition definition	When checked, the criterion includes only Pattern Definitions that have attached Decomposition Definitions.

Table 9. Display Criteria Setup window fields (Sheet 2 of 2)

Criteria	Description
Has attached representation definition	When checked, the criterion includes only Pattern Definitions that have the Representation Definition specified in *With component type* attached to them.
With component type	When *Has attached representation definition* is checked, the criterion includes only Pattern Definitions whose attached Representation Definitions include the specified Representation Definition. If *(any)* is specified, the criterion includes all Pattern Definitions that have any attached Representation Definition.

The effects of the *Display Criteria Setup* fields are cumulative.

Example 17. Display criteria

In Figure 42, the *Text Box* criterion is selected, and its values appear in the fields.

This criterion includes Pattern Definitions:

- Of any type.
- That are Primary Patterns.
- Whether or not they have attached Decomposition Definitions.
- That have attached Representation Definitions of the Text Box type.

Note: The words “Text Box” are not necessarily part of the Pattern Definition names, as you can see from the empty prefix and suffix fields.

Creating New Criteria

To create new criteria:

1. Open the *Display Criteria Setup* window.
2. Click the *New* button. The *Display Criteria Setup* dialog box opens:

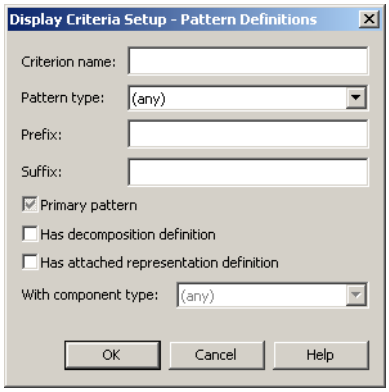


Figure 43. Display Criteria Setup dialog box

3 Type a name for the criterion in the *Name* field.
4 Insert values in the fields as desired.
5 Click *OK*. The *Display Criteria Setup* dialog box closes and the new criterion appears selected in the *Criteria List*.
6 Click *Close* to close the *Display Criteria Setup* window.

Modifying Existing Criteria

To modify existing criteria:

1 Open the *Display Criteria Setup* window.
2 Select a criterion to modify.
3 Click the *Modify* button. The *Display Criteria Setup* dialog box opens with the selected criterion's values inserted in the fields:

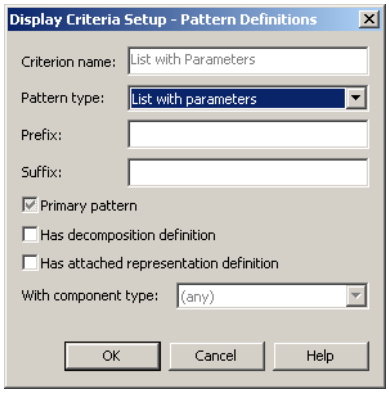


Figure 44. Display Criteria Setup dialog box in Modify mode

4 Edit values in the fields as desired. You cannot edit the criterion's name.
5 Click *OK*. The *Display Criteria Setup* dialog box closes and the modified criterion appears selected in the *Criteria list*.
6 Click *Close* to close the *Display Criteria Setup* window.

Deleting Criteria

To delete criteria:

1. Open the *Display Criteria Setup* window;
2. From the *Criteria list* select the criterion you wish to delete.
3. Click the *Delete* button. The selected criterion is removed from the *Criteria list*.
4. Click *Close* to close the *Display Criteria Setup* window.

Viewing Message Definitions

Message Definitions are regular Pattern Definitions in all respects, except that they play an additional role in runtime.

A Message Definition is used to recognize a host system message and then take whatever action is attached to the Message Definition. Details on the use of Message Definitions can be found in *JIS Interface Server: Advanced Topics*.

Message Definitions are not created. Instead, you designate a Pattern Definition as a Message Definition.

The Lower Message Pane

In Message View the lower left pane becomes the *Message* pane. The *Message* pane displays the names of all existing Message Definitions. Message Definitions exist only for iSeries systems.

The *Message* pane does not display any information about a Pattern Definition that has been designated as a Message Definition.

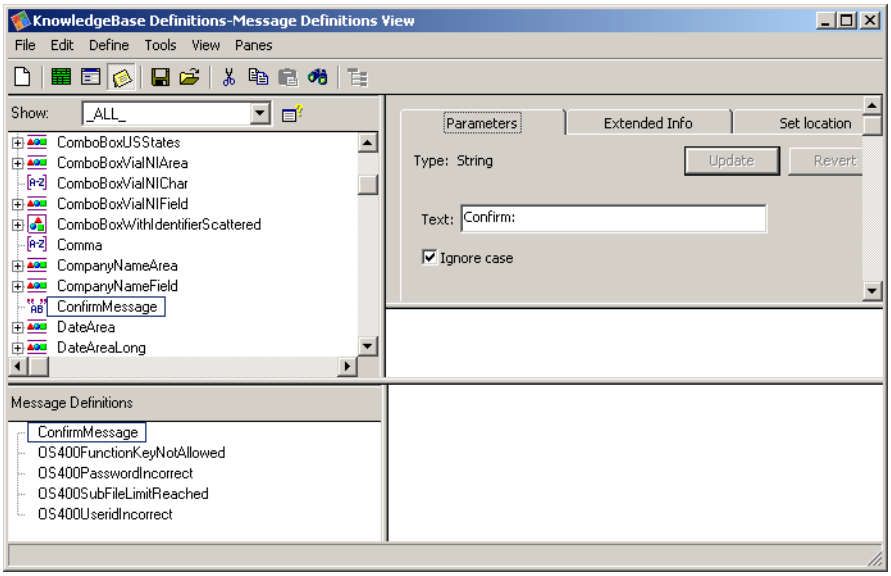


Figure 45. Message Definition view in the KnowledgeBase Definitions window

Viewing Pattern Definitions that are Message Definitions

When you enter Message View the lower pane displays the names of all the Message Definitions. The display state of the upper pane does not change. To view the structure of a Pattern Definition that has been designated as a Message Definition you must scroll the upper pane to the proper place.

To view a Pattern Definition that has been designated as a Message Definition:

1. In the *Lower Message* pane right click the Message Definition whose structure you wish to view.
2. From the shortcut menu select *Jump to definition in upper pane*. The *Upper Pattern* pane scrolls to the Pattern Definition and it is selected.

If the Pattern Definition name is not included in the current display state because of a filter applied to the upper pane, you are prompted to display all Pattern Definitions.

Querying the Use of Pattern Definitions

You can query the total number of times within a library that each Pattern Definition recognized a host screen character pattern.

To query Pattern Definition recognition statistics:

1. From the *Tools* menu select *Query*. The *KnowledgeBase Query* dialog box opens.
2. In the *Report file name* field, type a name for the file in which to write the results of the query.

   The file is written to your installation directory as a text file and can be opened with any ASCII file editor.

The file contains a list of each primary Pattern Definition and how many times it recognized a host pattern as well as a list of all Pattern Definitions and whether or not they recognized a host pattern.

Editing Pattern Definition Structure

Editing Pattern Definition structure is your most powerful tool, allowing you to globally change the functionality and look of your application with just a few operations. Consequently, you must exercise care when you do this, especially when editing the Pattern Definitions supplied with ACE.

Note: The *ListFromWizard* type Pattern Definitions should almost never be edited. These Pattern Definitions are present in special sections to which you do not have access. These special sections are used by ACE when you mark a list on your screen image. It can be very useful to modify such a Pattern Definition if you have many screens with an unusual list structure. However, the fact that you can only modify an existing *ListFromWizard* type Pattern Definition, but not change which Pattern Definitions are in the special list sections, means that the effect of any error, whether conceptual or typographical, is severe.

Saving the KnowledgeBase

Changes to the KnowledgeBase are saved to the file named MODS.GKB. When saving the KnowledgeBase, ACE automatically saves the previous version of the KnowledgeBase to a file called MODS.BAK. This backup file can be used to return to the version of the KnowledgeBase previous to the current version.

Using the KnowledgeBase backup file:

1. Open the ace/Appls folder.
2. Open your Application.
3. Rename MODS.BAK file as MODS.GKB.

Note: The KnowledgeBase cannot be saved if it is designated as a read-only file. For more details, see the chapter that describes the Configuration Management Intrastructure in ACE in *JIS Interface Server: Advanced Topics*.

KnowledgeBase Validation

Any Pattern Definition or Representation Definition you modify or add to the KnowledgeBase must be consistent with the logic of the KnowledgeBase. To make sure no illegal definition is added to the KnowledgeBase, the converter is provided with a validation mechanism that checks the added or modified definition against the logic of the KnowledgeBase.

The validation check is performed before the new or modified definition is written to the KnowledgeBase. Whenever an inconsistency is found between the modified definition and the KnowledgeBase, an error message is prompted displaying the invalid definition name and a detailed message explaining why the definition is invalid.

ListColumn type Pattern Definitions must have attached Representation Definitions including at least one Variable or ListColumn type component. If you attempt to save the KnowledgeBase before attaching one of these Representation components to a *ListColumn* type Pattern Definition the validation mechanism issues the following error message:

Before you can exit the KnowledgeBase window you must either fix the invalid definition/s or reload the KnowledgeBase by selecting *File > Load KnowledgeBase*. This reloads the last saved version of the KnowledgeBase. By reloading the KnowledgeBase you lose all the changes made to the KnowledgeBase since it was last saved.

You must exit the current Subapplication before you can reload the KnowledgeBase.

The KnowledgeBase validation mechanism is activated through an INI file parameter. In the `[Converter]` section of the GUISYS.INI file, you can set the validation mechanism in one of three modes:

- `KnowledgeBaseValidation=0`

  The validation mechanism is not activated.
- `KnowledgeBaseValidation=1`

  The default setting. The validation mechanism is activated. You cannot save any invalid definition to the KnowledgeBase. You cannot exit the KnowledgeBase window without fixing the invalid definitions or discarding the invalid definitions by reloading the KnowledgeBase.
- `KnowledgeBaseValidation=2`

  The validation mechanism is activated. You can save invalid definitions to the KnowledgeBase. You are advised of the invalid definition before exiting the KnowledgeBase window.

About the Structural Aspects of Editing Pattern Definitions

Pattern Definitions have two aspects which you edit in different parts of the *KnowledgeBase Definitions* window:

- You edit a Pattern Definition's hierarchical structure of child patterns in the *Upper Pattern* pane.
- You edit a Pattern Definition's properties in the *Properties* tabs area.

This section describes the structural aspects of editing Pattern Definitions. Editing a Pattern Definition's properties is described in "Editing Pattern Definition Properties" on page 147.

Top Level and Child Pattern Definitions

Every Pattern Definition in the KnowledgeBase appears as a top level Pattern Definition. Some Pattern Definitions also appear as child patterns of other Pattern Definitions.

The editing operations that you can perform on a Pattern Definition differ depending on whether you are editing a Pattern Definition at top level or as the child of another Pattern Definition. The reason is that the top level Pattern Definition controls the *existence* of the Pattern Definition in the KnowledgeBase. The child level controls the *use* of the Pattern Definition.

Example 18. Top level and child pattern definitions

- Cutting a child pattern from its parent pattern only affects how the child pattern is used. The child pattern still exists in the KnowledgeBase.
- Cutting a Pattern Definition at the top level *eliminates* the Pattern Definition from the KnowledgeBase.

Editing Child Patterns

When you edit a pattern, whether at top level or as the child of another Pattern Definition, you affect *every* use of the edited Pattern Definition.

Example 19. Editing child patterns

All of the *FKey* Pattern Definitions have *FKeyTerminator* as a child pattern. If you edit *FKeyTerminator* you affect all the *FKey* Pattern Definitions.

When you edit a Pattern Definition in the KnowledgeBase you are by default warned that you will affect every parent pattern of the edited pattern.

Cutting a Pattern Definition

To cut a Pattern Definition:

1. In the *Upper Pattern* pane select the Pattern Definition you wish to cut.
2. From the *Edit* menu choose *Cut*.

   -or-
3. In the *Upper Pattern Pane* select the Pattern Definition you wish to cut.
4. Click the *Cut* button on the Tool Bar

   -or-
5. In the *Upper Pattern* pane right click the Pattern Definition you wish to cut.
6. From the shortcut menu choose *Cut*.

You cannot cut a top level Pattern Definition if it is in use as a child pattern.

You cannot cut a top level Pattern Definition unless you created it or modified it. When you modify and then cut an ACE supplied Pattern Definition, the cutting is not permanent. Closing and then reopening ACE returns the cut Pattern Definition to the KnowledgeBase.

Replacing a Child Pattern

You can replace a child pattern with another Pattern Definition. The child pattern must be in the *Upper Pattern* pane. The replacement Pattern Definition can be chosen from either pane.

To replace a child pattern:

1. Select the replacement Pattern Definition from either *Pattern* pane.
2. From the *Edit* menu choose *Copy*.

   -or-

   Click the *Copy* button on the Tool bar

   -or-

   Right click the Pattern Definition and choose *Copy* from the shortcut menu.
3. Select the child pattern to be replaced.

4 From the *Edit* menu choose *Paste* > *Paste as Replaced Pattern*.

-or-

Right click the child pattern to be replaced and choose *Paste as Replaced Pattern* from the shortcut menu.

-or-

Click the *Paste* button on the Tool bar and choose *Paste as Replaced Pattern* from the shortcut menu.

A warning that the existing child pattern is about to be replaced appears. Click *OK*. The warning closes and the highlighted child pattern is replaced.

Drag-and-Drop Replacement

To replace a child pattern using drag-and-drop:

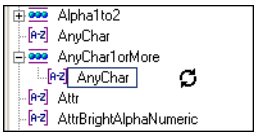
1 Make the child pattern to be replaced visible in the *Upper Pattern* pane.
2 Make the replacement Pattern Definition visible in the *Lower Pattern* pane.
3 Click and hold down the left mouse button while the cursor is positioned on the replacement Pattern Definition.
4 Drag the replacement Pattern Definition from the *Lower Pattern* pane onto the child pattern to be replaced in the *Upper Pattern* pane.

The child pattern is highlighted and the cursor changes to the symbol .

5 Release the left mouse button. A warning that the existing child pattern is about to be replaced appears.
6 Click *OK*. The warning closes and the highlighted child pattern is replaced.

Example 20. Drag and drop replacement of pattern definitions

The figure shows the child pattern *AnyChar* about to be replaced.



Adding a New Child Pattern to a Parent Pattern

You can make a Pattern Definition into a new child pattern of another Pattern Definition. The parent pattern must be in the *Upper Pattern* pane. The Pattern Definition added as the new child pattern can be chosen from either *Pattern* pane.

Positioning the New Child: Adding as a Child vs. Adding as a Sibling

You determine the position of a new child pattern within the list of existing child patterns by choosing between one of two types of procedures:

- To create a new child immediately before a particular existing child, add a Pattern Definition as a *sibling* of the particular existing child pattern.
- To create a new child pattern after the last existing child pattern, add a Pattern Definition as a *child* of the parent pattern.

Positioning a New Child Before an Existing Child: Adding as a Sibling

To make a Pattern Definition into a new child pattern and position the new child pattern immediately before a particular existing child pattern:

1. Select the Pattern Definition to be added as a sibling from a *Pattern* pane.
2. From the *Edit* menu choose *Copy*.

   -or-

   Click the *Copy* button on the Tool Bar

   -or-

   Right click the Pattern Definition and choose *Copy* from the shortcut menu.
3. Select the existing child pattern before which you want to position the new child pattern.
4. From the *Edit* menu choose *Paste* > *Paste Pattern as sibling*.

   -or-

   Right click the existing child pattern and choose *Paste Pattern as sibling* from the shortcut menu.

   -or-

   Click the *Paste* button on the Tool bar and choose *Paste Pattern as sibling* from the shortcut menu.
5. Click *Yes* in the confirmation message box. The Pattern Definition to be added appears as a new child pattern positioned immediately before the existing child pattern.

Drag-and-Drop a Pattern Definition as a Sibling

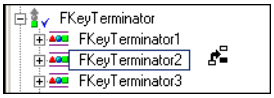
To make a Pattern Definition into a new child pattern, positioned immediately before a particular existing child pattern, using drag-and-drop:

1. Make the existing child pattern visible in the *Upper Pattern* pane.
2. Make the Pattern Definition to be added as a new child visible in the *Lower Pattern* pane.
3. Click and hold down the left mouse button while the cursor is positioned on top of the Pattern Definition to be added as a new child pattern.

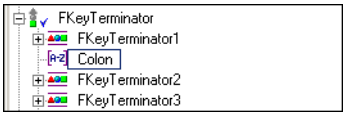
4 Drag the Pattern Definition from the *Lower Pattern* pane to just above the existing child pattern in the *Upper Pattern* pane. The child pattern is highlighted and the cursor changes to the symbol .
5 Release the left mouse button. The *dragged* Pattern Definition appears as a new child pattern positioned immediately before the existing child pattern.

Example 21. Drag and drop a pattern definition as a sibling

The figure shows the parent pattern *FKeyTerminator* about to receive a new child pattern. The new child pattern is being added as a sibling to the existing child pattern *FKeyTerminator2*:



The next figure shows the result of adding *Colon* as a sibling of *FKeyTerminator2*:



Positioning a New Child After All Existing Children: Adding as a Child

To make a Pattern Definition into a new child pattern and position the new child pattern after all existing child patterns:

1 Select the Pattern Definition to be added as a child from a *Pattern* pane.
2 From the *Edit* menu choose *Copy*

   -or-

   Click the *Copy* button

   -or-

   Right click the Pattern Definition and choose *Copy* from the shortcut menu.
3 Select the parent pattern which is to receive a new child pattern.
4 From the *Edit* menu choose *Paste* > *Paste Pattern as child*

   -or-

   Right click the existing child pattern and choose *Paste Pattern as child* from the shortcut menu

   -or-

   Click the *Paste* button.

   The Pattern Definition to be added appears as a new child pattern positioned after all of the existing child patterns.

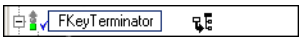
Drag-and-Drop Add as a Child

To make a Pattern Definition into a new child pattern, positioned after all existing child patterns, using drag-and-drop:

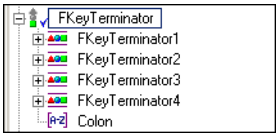
1. Make the parent pattern that will receive the new child visible in the *Upper Pattern* pane.
2. Make the Pattern Definition to be added as a new child visible in the *Lower Pattern* pane.
3. Click and hold down the left mouse button while the cursor is positioned on the Pattern Definition to be added as a new child pattern.
4. Drag the Pattern Definition from the *Lower Pattern* pane to just below the parent pattern in the *Upper Pattern* pane. The parent pattern is highlighted and the cursor changes to the symbol .
5. Release the left mouse button. The *dragged* Pattern Definition appears as a new child pattern positioned after all the existing child patterns.

Example 22. Drag and drop a pattern definition as a child pattern

The figure shows the parent pattern *FKeyTerminator* about to receive a new child pattern. The new child pattern is being added as a child pattern of *FKeyTerminator*:



The next figure shows the result of adding *Colon* as a child of *FKeyTerminator*. Note that *FKeyTerminator* is automatically expanded one level in order to make the new child's positioning visible:



Duplicating a Pattern Definition

Duplicate an existing Pattern Definition to create a new Pattern Definition that has a different name but is otherwise identical to the existing Pattern Definition.

To duplicate a Pattern Definition:

1. In the *Upper Pattern* pane right click the Pattern Definition you wish to duplicate.
2. From the shortcut menu choose *Duplicate*. The *Duplicate Pattern* dialog box opens.

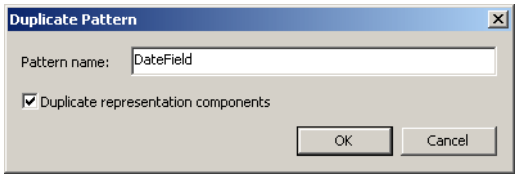


Figure 46. Duplicate Pattern dialog box

3 Type the name of the new Pattern Definition in the *Pattern name* field.
4 If you want to duplicate the attached Representation Definition as well, set the *Duplicate representation components* check box. The check box is disabled when no Representation Definitions are attached to that Pattern Definition.
5 Click *OK*. The *Duplicate Pattern* dialog box closes and the new Pattern Definition and the attached Representation Definitions appear in both *Pattern* panes.

Creating a New Pattern Definition

You can create a new Pattern Definition as just a top level Pattern Definition, or as a top level Pattern Definition and a child pattern simultaneously.

The new Pattern Definition can be positioned as a child pattern using:

Replace Select the *child* pattern to be replaced before creating the new Pattern Definition. The new Pattern Definition will replace the selected child pattern.

Sibling Select a *child* pattern before creating the new Pattern Definition. The new Pattern Definition will be positioned immediately before the selected child pattern.

Child Select a *parent* pattern before creating the new Pattern Definition. The new Pattern Definition will be positioned after all existing child patterns of the selected parent pattern.

To create a new Pattern Definition:

1 If the new Pattern Definition is to be created simultaneously as both a top level and as a child, select the appropriate child or parent pattern from the *Upper Pattern* pane.
2 From the *Define* menu choose *New*, or click the *New* button on the Tool bar, or right click within the *Upper Pattern* pane and select *New* from the shortcut menu. When you create the new Pattern Definition also as a child pattern, right click on the appropriate child or parent pattern. The *New Pattern* dialog box opens:

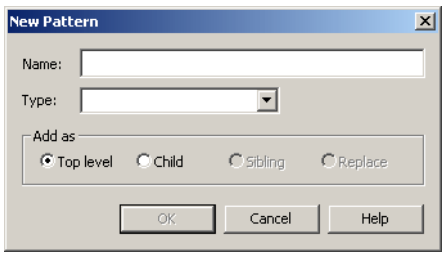


Figure 47. New Pattern dialog box

3 Enter a name for the new Pattern Definition in the *Name* edit box. The *OK* button is enabled.

4 Choose the new Pattern Definition's type from the *Type* combo box.

5 Choose the *Top Level* radio button to create the new Pattern Definition as a top level Pattern Definition only. The individual radio buttons are enabled separately when a valid child/parent pattern is selected at the beginning of this procedure.

6 To create the new Pattern Definition also as a child pattern, choose the *Child* radio button. This creates the new Pattern Definition as *both* a top level and as a child.

7 Click *OK*. The *New Pattern* dialog box closes and the new Pattern Definition appears in its correct alphabetical position in the *Upper Pattern* pane.

When you create the new Pattern Definition as a child pattern also, the new Pattern Definition appears as a child pattern. It will only appear as a top level Pattern Definition when the new Pattern Definition is fully defined.

At this point the new Pattern Definition is not fully defined. The new Pattern Definition may require that you specify:

- Parameter values. For details, see "Editing Pattern Definition Properties" on page 147.
- Child patterns. For details, see "Adding Child Patterns to a New Pattern Definition" on page 146.

You will always have to do at least one of the above because all the Pattern Definitions *except* for Character Set and String types require child patterns, while the following Pattern Definitions require parameters:

- Character Set
- Dynamic Iteration
- Horizontal Iteration
- List with Parameters
- Popup Border
- String
- Vertical Iteration

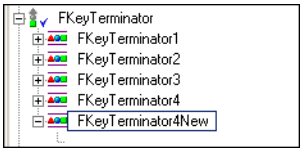
Note: You must make the new Pattern Definition available to the ACE analysis process. For more information, see "The Extended Info Tab" on page 157.

Adding Child Patterns to a New Pattern Definition

When you create a new Pattern Definition that requires child patterns, the new Pattern Definition appears with an empty child position.

Example 23. Adding child patterns to a new pattern definition

You create a new Pattern Definition *FKeyTerminator4New* simultaneously as a top level Pattern Definition and as a child of *FKeyTerminator*. *FKeyTerminator4New* is positioned after all of *FKeyTerminator's* existing children, and *FKeyTerminator4New* has an empty child position:



To complete the structural definition of FKeyTerminator4New you must give it a child pattern. This new child can be an existing Pattern Definition or a new Pattern Definition created as a child pattern.

You must eliminate the blank child. You can eliminate the blank child by:

- Replacing the blank child with a Pattern Definition.

  -or-
- Adding child patterns and then cutting the blank child pattern.

Some Pattern Definitions related to lists and popup windows have more than one blank child, with each child pattern playing a named role in the Pattern Definition. These blank children can be eliminated only by replacing the blank child with a Pattern Definition.

Designating a Pattern Definition as a Message Definition

In Message View you can designate a Pattern Definition as a Message Definition.

To designate a Pattern Definition as a Message Definition:

1. Go to Message View.
2. In the *Upper Pattern* pane right click a Pattern Definition and choose New from the shortcut menu

   -or-

   In the *Upper Pattern* pane select a Pattern Definition and either click the *New* button on the Tool bar.

   -or-

   Select *New* from the *Edit* menu.
3. Confirm your choice. The Pattern Definition is designated as a Message Definition.

Editing Pattern Definition Properties

Pattern Definitions have two aspects which you edit in different parts of the *KnowledgeBase Definitions* window:

- You edit a Pattern Definition's hierarchical structure of child patterns in the *Upper Pattern* pane. For more information, see "Editing Pattern Definition Structure" on page 136.
- You edit a Pattern Definition's properties in the *Properties* pane.

This section explains how to edit Pattern Definition properties in the KnowledgeBase Definitions window.

The Properties Pane

There are three tabs in the *Properties* pane:

Parameters Displays a selected Pattern Definition's type and any additional structural information required by the Pattern Definition. The *Parameters* tab is different for each Pattern Definition type. An explanation of the *Parameters* tab is given for each Pattern Definition type in the following sections.

Extended Info	Displays a selected Pattern Definition's type, whether the selected Pattern Definition is a Primary Pattern, and an example of host screen characters that satisfy the Pattern Definition. The *Extended Info* tab is the same for all Pattern Definition types.
Set/Change Location	Displays a selected Pattern Definition's type and where in the host screen ACE begins to search for this Pattern Definition. The *Set/Change Location* tab is the same for all Pattern Definition types.

Pattern Definition Types

The following sections describe each Pattern Definition type and the changes that can be made in that Pattern Definition type's *Parameters* tab.

Character Set

A Character Set is a sub-set of all the characters.

The list of all the characters includes the following:

- Capital Letters
- Small Letters
- Digits
- Specials
- Input Attributes
- Non Input Attributes
- Null

A Pattern Definition of the Character Set type is satisfied by a host screen character sequence consisting of any single character that is a member of the Character Set.

Character Set Parameters Tab

You can edit the members of a Character Set in the *Parameters* tab. Members of the Character Set are red on a gray background. Non-members of the Character Set are black on a white background.

Each character can be toggled. Click a member character to remove that character from the Character Set. Click a non-member character to add that character to the Character Set.

Click the *Revert* button to undo changes or click the *Update* button to accept changes.

Example 24. Character Set parameters tab

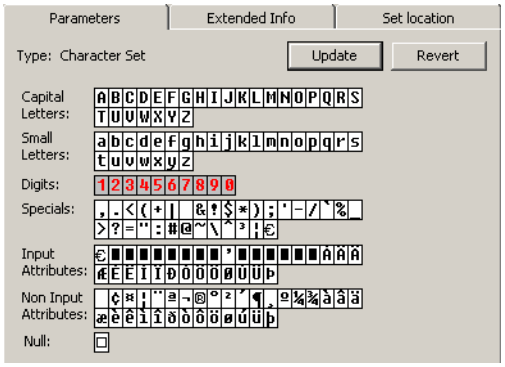


Figure 48. Character Set Parameters tab

The figure shows the *Parameters* tab of the Character Set type Pattern Definition Digit. The ten members of Digit are the characters *1 2 3 4 5 6 7 8 9* and *0*. Digit is satisfied by any one of these ten characters.

Dynamic Group

A Dynamic Group contains one or more Dynamic Iterations as child patterns. Dynamic Groups are used to recognize host screen areas whose contents are determined at runtime.

A Dynamic Group type Pattern Definition's *Parameters* tab displays only the Pattern Definition's type, and the *Parameters* tab cannot be edited.

Dynamic Iteration

A Dynamic Iteration type Pattern Definition contains a single child pattern, repeated at least a minimum number of times and at most a maximum number of times. A Dynamic Iteration type Pattern Definition is satisfied by a sequence of host screen characters that is composed of several smaller sequences. Each

smaller sequence must satisfy the Dynamic Iteration's child pattern separately. The entire sequence must contain at least the minimum number of smaller sequences and at most the maximum number of smaller sequences.

Dynamic Iterations are used to recognize host screen areas whose contents are determined at runtime.

Dynamic Iteration Parameters Tab

A Dynamic Iteration's *Parameters* tab displays the minimum number of times the Dynamic Iteration's child pattern must be repeated and the maximum number of times the child pattern may be repeated:

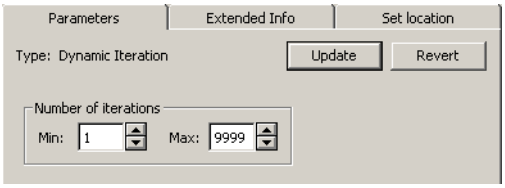


Figure 49. Dynamic Iteration Parameters tab

- In the *Min* box, change the minimum number of child pattern appearances.
- In the *Max* box, change the maximum number of child pattern appearances.
- Click the *Revert* button to undo changes.
- Click the *Update* button to accept changes.

Horizontal Group

A Horizontal Group is a parent pattern to one or more child patterns. A host screen character sequence that satisfies a Horizontal Group type Pattern Definition is a left-to-right grouping of small character sequences. The left small sequence must satisfy the first child pattern, the next sequence must satisfy the second child pattern, through to the right sequence which must satisfy the last child pattern.

Example 25. Horizontal group

The Pattern Definition *Menu* is of Horizontal Group type. *Menu* has three child patterns: *MenuOption*, *MenuTerminator* and *MenuDescription*.

MenuOption	is satisfied by the character sequence:	1

MenuTerminator	is satisfied by the character sequence:	.
MenuDescription	is satisfied by the character sequence:	Manufacturing
Menu	is satisfied by the character sequence:	1.Manufacturing

A Horizontal Group type Pattern Definition's *Parameters* tab displays only the Pattern Definition's type and cannot be edited.

Horizontal Iteration

A Horizontal Iteration type Pattern Definition contains a single child pattern, repeated at least a minimum number of times and at most a maximum number of times. A Horizontal Iteration type Pattern Definition is satisfied by a sequence of host screen characters that is composed of several smaller sequences. The smaller sequences must appear left-to-right across the host screen. Each smaller sequence must separately satisfy the Horizontal Iteration's child pattern. The entire sequence must contain at least the minimum number of smaller sequences and at most the maximum number of smaller sequences.

Horizontal Iteration Parameters Tab

A Horizontal Iteration's *Parameters* tab displays the minimum number of times the Horizontal Iteration's child pattern must be repeated and the maximum number of times the child pattern may be repeated:

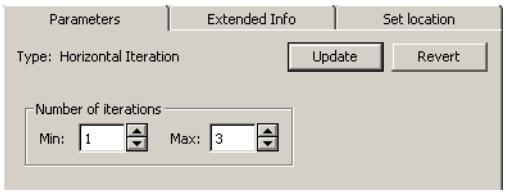


Figure 50. Horizontal Iteration Parameters tab

- In the *Min* box, change the minimum number of child pattern appearances.
- In the *Max* box, change the maximum number of child pattern appearances.
- Click the *Revert* button to undo changes.
- Click the *Update* button to accept changes.

List Column

List Column type Pattern Definitions are related to lists.

A List Column type Pattern Definition *Parameters* tab displays only the Pattern Definition's type and cannot be edited.

List with Parameters

List with Parameters type Pattern Definitions are related to lists. See the chapter on Lists in *JIS Interface Server: Advanced Topics*.

List with Parameters' Parameters Tab

A List with Parameters' *Parameters* tab displays information on:

- The header rows and their location relative to the list.
- Logical list records that extend over two or more physical rows on the screen.
- Logical list columns that are divided into two or more physical columns on the screen.
- Placement of the list and its components.

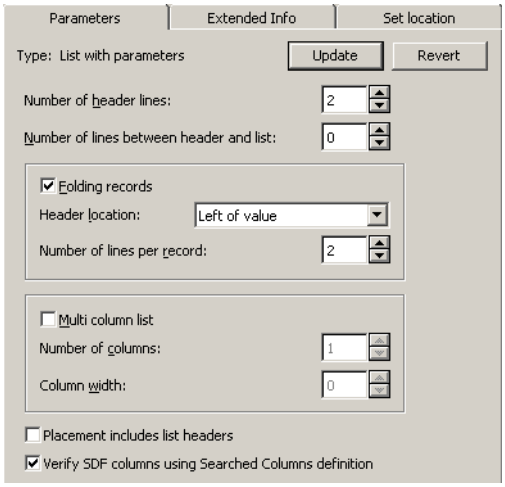


Figure 51. List with Parameters' Parameters tab

In the List with Parameters' *Parameters* tab you can:

- Change the number of physical rows in the header in the *Number of header lines* box.
- Change the number of physical rows separating the header rows from the list rows in the *Number of lines between header and list* box.
- If logical list rows extend over more than one host screen physical row, set the *Folding records* check box.

- From the *Header location* box choose the format for positioning the extended row of headers.
- Change the number of physical rows per logical list row in the *Number of lines per record* box.
- If logical list columns are divided into two or more physical columns on the screen, set the *Multi column list* check box.
- Change the number of logical columns in the *Number of columns* box.
- Change the aggregate width, in characters, of the logical columns in the *Column width* box.
- When working with captured screens (non-SDF), if the Pattern Definition includes the header rows in the total list area, set the *Placement includes list headers* check box.
- To refine the SDF filter designation of list columns via the Pattern Definition's structure, set the *Verify SDF columns using Searched Columns definition* check box.

Note: The Parameters tab is disabled for Pattern Definitions whose name includes "FromWizard". These Pattern Definitions are meant to be used with the *Add List* wizard and should not be edited.

One Of

A One Of type Pattern Definition has two or more child patterns. A character sequence that satisfies a One Of type Pattern Definition is a sequence that satisfies any one of the child patterns.

ACE searches for character sequences satisfying the child patterns according to the order in which the child patterns appear in the One Of type Pattern Definition. The result is that a character sequence satisfying more than one of the child patterns is recognized by the first child and is not recognized by any other child pattern.

A One Of type Pattern Definition's *Parameters* tab displays only the Pattern Definition's type, and the *Parameters* tab cannot be edited.

Popup Border

A Popup Border type Pattern Definition has four child patterns; *Left*, *Top*, *Right* and *Bottom*. A Popup Border type Pattern Definition is satisfied by characters within a rectangular area of the host screen. The left column of the area must satisfy the *Left* child pattern, the top row of the area must satisfy the *Top* child pattern, the right column of the area must satisfy the *Right* child pattern and the bottom row of the area must satisfy the *Bottom* child pattern.

Popup Border Parameters Tab

A Popup Border's *Parameters* tab displays the minimum and maximum number of rows in the popup window and the minimum and maximum number of columns in the popup window:

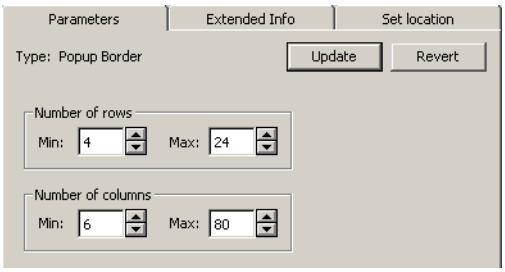


Figure 52. Popup border Parameters tab

- Change the minimum number of rows or columns in the appropriate *Min* box.
- Change the maximum number of rows or columns in the appropriate *Max* box.
- Click the *Revert* button to undo changes.
- Click the *Update* button to accept changes.

Scattered Group

A Scattered Group type Pattern Definition contains one or more child patterns. A host screen character sequence that satisfies a Scattered Group type Pattern Definition is a collection of small character sequences. One small sequence must satisfy the first child pattern, a second different sequence must satisfy the second child pattern, etc.

There is no restriction on the arrangement of the individual small character sequences. They do not need to be adjacent, and other character sequences can lie between the individual small sequences.

A Scattered Group type Pattern Definition's *Parameters* tab displays only the Pattern Definition's type, and the *Parameters* tab cannot be edited.

String

A String type Pattern Definition is a specific sequence of characters. A host screen character sequence that satisfies a String type Pattern Definition must match the String sequence character-by-character, with no omissions.

String Parameters Tab

You define the String's character sequence on the String's *Parameters* tab:

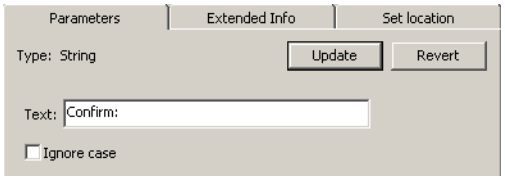


Figure 53. String Parameters tab

- Type the String's character sequence in the *Text* box.
- Set the *Ignore case* check box to make character sequences satisfy the Pattern Definition regardless of the case of alphabetic characters.
- Click the *Revert* button to undo changes.
- Click the *Update* button to accept changes.

Vertical Group

A Vertical Group is a parent pattern to a number of child patterns. A host screen character sequence that satisfies a Vertical Group type Pattern Definition is a group of small character sequences, each lying in a separate row. All the small sequences must start in the same column. The topmost small sequence must satisfy the first child pattern, the next sequence must satisfy the second child pattern, through to the bottom-most sequence which must satisfy the last child pattern.

Example 26. Vertical group

The Pattern Definition CheckBox0or1Identifier1 is of Vertical Group type. CheckBox0or1Identifier1 has two child patterns: ZeroEqualNoStr and OneEqualYesStr.

ZeroEqualNoStr	is satisfied by the character sequence:	0=No
OneEqualYesStr	is satisfied by the character sequence:	1=Yes

CheckBox0or1Identifier1	is satisfied by the character sequence:	0=No 1=Yes

A Vertical Group type Pattern Definition's *Parameters* tab displays only the Pattern Definition's type, and the *Parameters* tab cannot be edited.

Vertical Iteration

A Vertical Iteration type Pattern Definition contains a single child pattern, repeated at least a minimum number of times and at most a maximum number of times. A Vertical Iteration type Pattern Definition is satisfied by a sequence of host screen characters that is composed of several small sequences. Each small sequence must appear in a separate row on the host screen, and all the small sequences must start in the same column. Each small sequence must satisfy the Vertical Iteration's child pattern separately. The entire sequence must contain at least the minimum number of small sequences and at most the maximum number of small sequences.

Vertical Iteration Parameters Tab

A Vertical Iteration's *Parameters* tab displays the minimum number of times and the maximum number of times the Vertical Iteration's child pattern must be repeated:

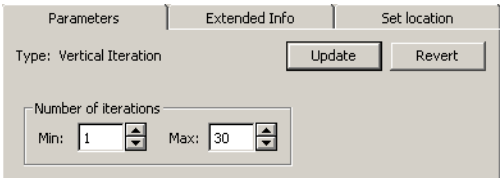


Figure 54. Vertical Iteration Parameters tab

- Change the minimum number of times the child pattern must appear in the *Min* box.
- Change the maximum number of times the child pattern may appear in the *Max* box.
- Click the *Revert* button to undo changes.
- Click the *Update* button to accept changes.

The Extended Info Tab

The *Extended Info* tab displays the Pattern Definition's type, whether the Pattern Definition is a Primary Pattern, and an example of a host screen character sequence that satisfies the Pattern Definition.

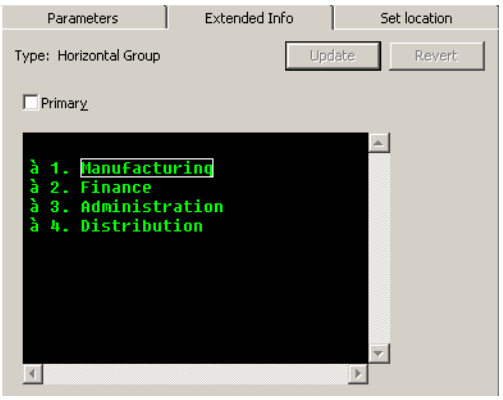


Figure 55. Extended Info tab

- Designate a Pattern Definition as a Primary Pattern by setting the *Primary* check box.
- Click the *Revert* button to undo changes.
- Click the *Update* button to accept changes.

All types of Pattern Definitions may be Primary Patterns except for the Character Set, One Of, and String types.

Primary Patterns

Some Pattern Definitions are designated as primary Pattern Definitions. ACE uses these primary patterns for host screen analysis.

ACE analyses a screen using a layout containing sections, where each section contains primary Pattern Definitions. ACE searches the host screen area covered by a section for all primary patterns in the section.

To make a new primary Pattern Definition active in the analysis, you must add the new primary pattern to all appropriate sections.

The Set/Change Location Tab

The following settings can be set in the Set/Location Tab:

Setting Location

ACE enables you to define the location of a Pattern Definition. Either the specific location can be defined, or the location can be defined relative to a screen item. For example, rather than specifying the screen header as appearing on lines 1-3, the header can be expressed relative to the first line of the screen.

> **Note:** This applies to searching the header of a Popup window as well.

This feature is especially useful for specifying the location of certain list characteristics. List options, for example, can be searched for relative to the list, rather than in a specific location.

Using the Set/Change Location Tab

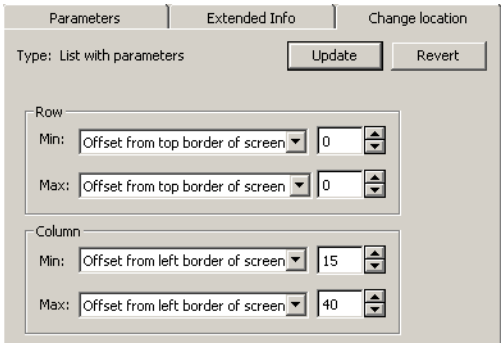


Figure 56. Set/Change Location tab

Change the minimum and maximum locations to begin searching for the Pattern Definition in the corresponding *Row* and *Column* boxes.

Choose from these following values:

No limit

Fixed location

Offset from top border of screen Row location only

Offset from bottom border of screen	Row location only
Offset from left border of screen	Column location only
Offset from right border of screen	Column location only
Offset from top border of list	Row location only
Offset from bottom border of list	Row location only
Offset from left border of list	Column location only
Offset from right border of list	Column location only

- The tab is called *Set Location* when all four fields are set to *No Limit*. Otherwise the tab name changes to *Change Location*.
- Change offset values for each location in the location's edit box. The offset value can be positive or negative.
- Click the *Revert* button to undo changes.
- Click the *Update* button to accept changes.

Note: The *Min* and *Max* fields of the location parameter are limits on where the Pattern Definition must begin. For example, the example parameters pictured above specify that the Pattern Definition must begin between columns 15 and 40, but it can extend beyond column 40. Row and column numbering are 0-based.

Chapter 8. Subapplication-Specific Modifications of the Design

The look and functionality of the windows in your application are determined by two factors:

- Design View
- Working in Design View
- Control Editing
- Modifying Control Properties
- Undoing Changes
- Attaching Functionality to Controls

The KnowledgeBase contains global definitions which are automatically applied to every window in the Application. When creating a window, ACE first applies the KnowledgeBase definitions. Then, any local modifications made to specific windows are applied.

To enhance the design of a specific window you make local modifications in Design View. In Design View, you can arrange the controls on the window, modify control properties, add or delete controls, and attach functionality to controls. Design View provides you with all the tools necessary to make these modifications.

This chapter describes how to perform Subapplication-specific modifications in Design View.

For information on Representation Definitions, see Chapter 9 - "Representation Definitions" on page 199.

This chapter describes the following topics:

- Design View
- Working in Design View
- Control editing
- Modifying control properties
- Undoing changes
- Attaching functionality to controls

Design View

Design View opens by displaying the host screen with a suggested window presentation. Each element on the host screen is represented by a Windows control.

The following picture shows the host screen in the background and the Windows representation in the foreground:

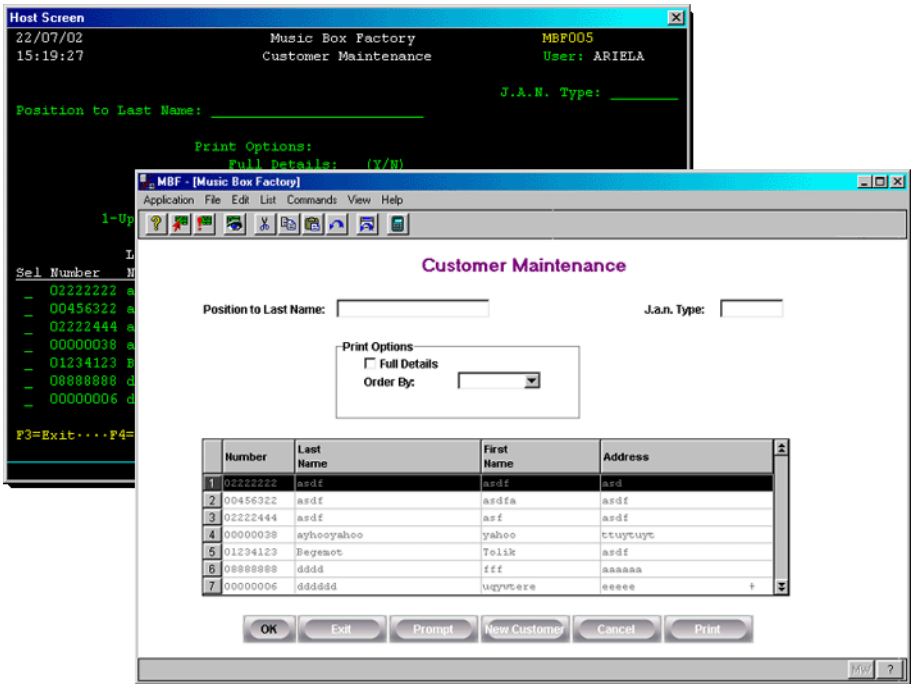


Figure 57. Host screen transformation into GUI window

Notice that the window contains many types of controls:

- *Text Boxes* such as: Position to Last Name.
- *Combo Boxes* such as: Order By.
- *Check Boxes* such as: Full Details.
- *Tables* that contains a list with several columns.
- *Push Buttons*, such as: OK.

Control Appearance vs. Control Functionality

Design View provides tools to customize the look of your Application—which controls are on each window, the appearance of each control and how all controls are arranged on the window. You can also attribute functionality to a control, using Methods.

In Design View you can modify and create Methods. You can then attach Methods to controls in the window either locally in Design View, or globally via the KnowledgeBase. For further information, see "Attaching Functionality to Controls" on page 197.

Representation Definitions

How does ACE know which controls should be used to create a window? The controls displayed in the window are the representations of the Pattern Definitions recognized during the screen analysis. In the KnowledgeBase, a Representation Definition can be linked to a Pattern Definition. The Representation Definition contains one or more components which define the control/s that will be used in the windows presentation.

Example 27. Single representation definitions

The following screen pattern `F3=Exit`, matched by the *FKey* Pattern Definition, has a Representation Definition with a single Button component:

Pattern Definition:	**Representation Definition component:**
FKey	Button

In some cases, a Pattern Definition can have more than one Representation Definition component.

Example 28. Multiple representation definitions

The following screen pattern `F3=Exit`, matched by the *FKey* Pattern Definition, has a Representation Definition with several components: Button, Menu Item, and Accelerator:

Pattern Definition:	**Representation Definition components:**
FKey	- Button
	- Menu Item
	- Accelerator (F3)

Other Representations

Some meaningful patterns are used only to obtain information for the conversion and are not displayed in the window. This type of representation component is referred to as a Variable.

For more information, see “Representation Definitions and How They Work” on page 199.

GUI Control Types

The window shown below features some of the most common controls.

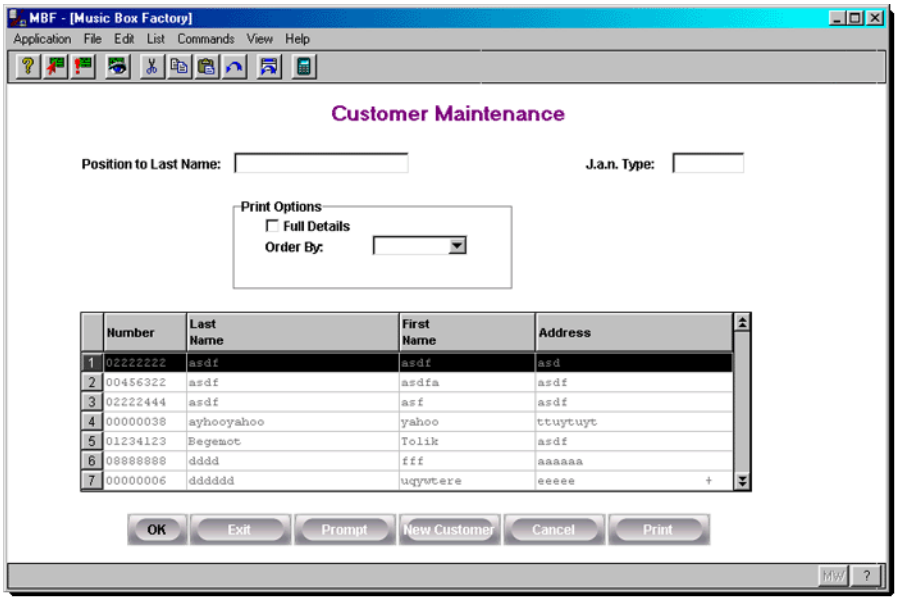


Figure 58. Common GUI controls

The following GUI control types are supported by ACE:

- Button
- Combo box
- Check box
- Date
- Frame
- Group box
- Line
- Link
- Prompt
- Radio Group
- Spin
- Static
- Sub-window
- Table
- Tabs
- TextBox

For information on control properties, see “Control Types and Styles” on page 225.

Working in Design View

Design View provides a number of tools to help you design the windows in your Application.

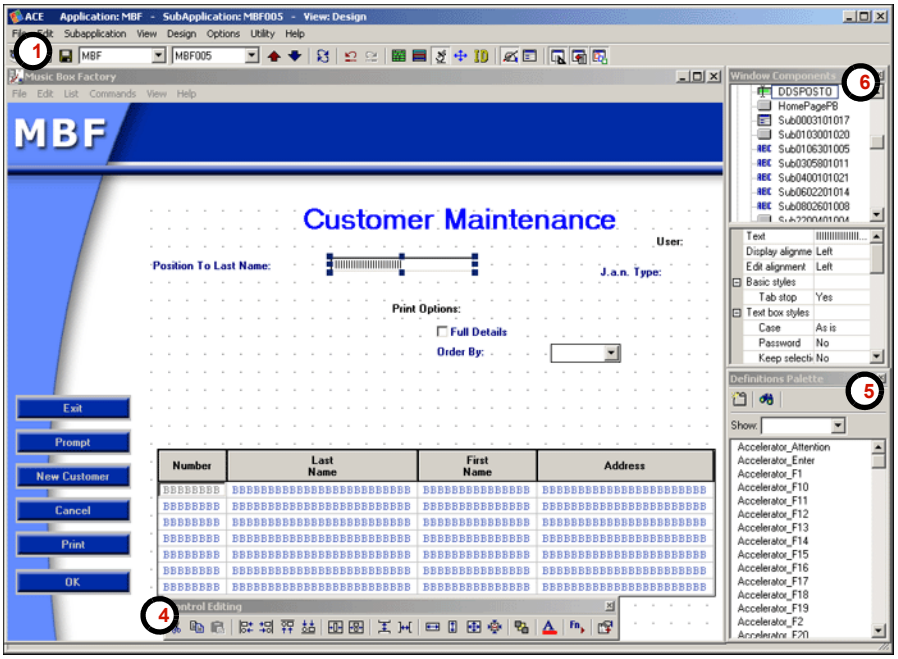


Figure 59. Design View

Design View tools:

1. The Design View toolbar.
2. The *View* menu.
3. The *Design* menu.
4. The *Control Editing* palette.
5. The *Definitions* palette.
6. The *Window Components* palette.

The following sections describe each of these tools in detail.

> **Note:** All changes made in Design View are Subapplication-specific and affect only the current window.

Design View Toolbar

In addition to the Design View icon, there are three additional icons that are only displayed in the toolbar in Design View. These icons enable you to toggle the display of the Design View palettes.

Figure 60. Design View toolbar

Design View Palettes

In Design View, there are three palettes to help you design the window:

1. *Control Editing* Palette
2. *Definitions* Palette
3. *Window Components* Palette

The functionality provided by each palette is described in detail later in this chapter.

You can configure each palette to always be on top of the Design View window. This ensures that when a palette and a window overlap, clicking on the window does not cause the palette to disappear behind the window. This option can be turned on and off as necessary.

To configure a palette to remain always on top of the Design View window:

- Right-click the palette's title bar and select *Always on Top* from the shortcut menu.

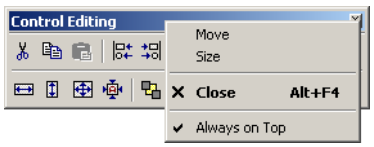


Figure 61. Always on Top option

- The display of the Design View palettes can also be controlled using the *View* menu.

View Menu

The *View* menu contains commands to show or hide Screen and Window Views.

It also contains a *Palettes* submenu which you use to show or hide Design View palettes.

In addition, the *Customize* submenu enables you to show or hide the grid, and to show or hide overlapping controls.

To show or hide the *Control Editing* palette, for example:

- From the *View* menu, choose *Palettes* > *Control Editing*, or click the toolbar icon:

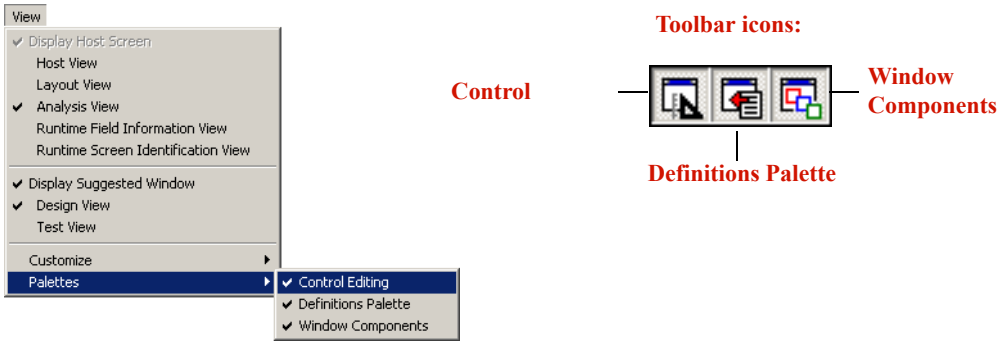


Figure 62. Toggle viewing of Design View Palettes

Design View Grid

In Design View, it is possible to display the window with a grid, as a visual guide to help you align controls. The grid can only be seen in Design View, and it is not part of the runtime Application.

To show or hide the grid:

From the *View* menu choose *Customize* > *Show Grid*.

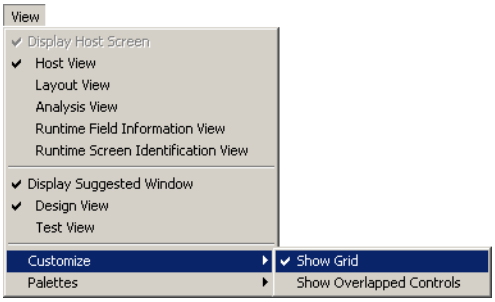


Figure 63. Toggle viewing of Design View grid

It is also possible to configure additional grid properties such as its style, the distance between the horizontal and the vertical grid lines, and the “snap to grid” behavior. This is done in the *Window Options* dialog box.

To configure grid properties:

1 From the *Options* menu select *Window Options*. The *Window Options* dialog box opens.
2 In the *Grid Attributes* tab, specify the desired options

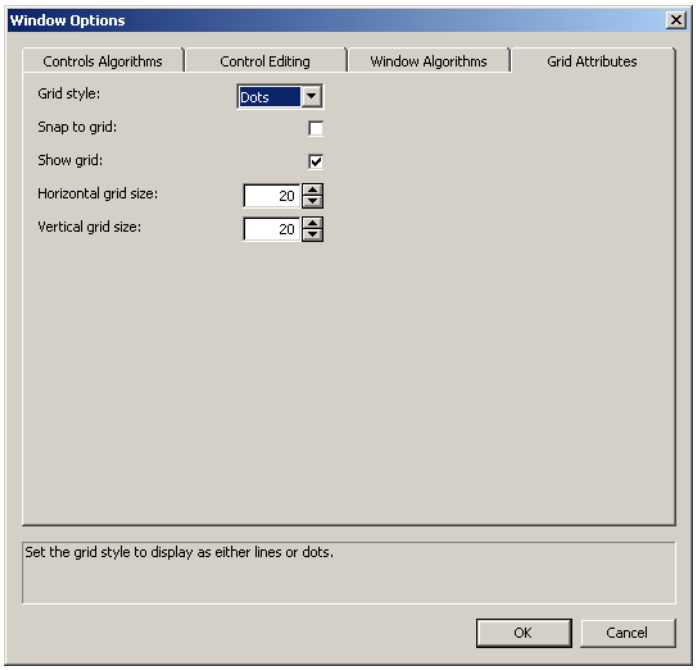


Figure 64. :Setting grid properties in the Window Options dialog box

Grid style	The grid markings can be either lines or dots. The default is dots.
Snap to grid	When selected, the upper left corner of the control snaps to the nearest intersection of grid lines. When a group of controls is selected, the upper left corner of the imaginary rectangle that encompasses the group snaps to nearest grid lines.
Show grid	Select this option to display the window with a grid.
Horizontal grid size	Set the horizontal distance, in dialog units, between the grid lines.

Vertical grid size Set the vertical distance, in dialog units, between the grid lines.

Note: Help for the option in focus appears at the bottom of the tab.

3 Click *OK* to save your settings.

Highlighting Overlapping Controls

In some cases, controls in a window may overlap. When controls overlap, the covered control may be completely hidden by the control on top. The *Show Overlapped Controls* option enables you to see overlapping controls in the window in Design View. When this option is selected, each overlapping control is highlighted with an animated border. In this manner, you are aware of all the controls in your window.

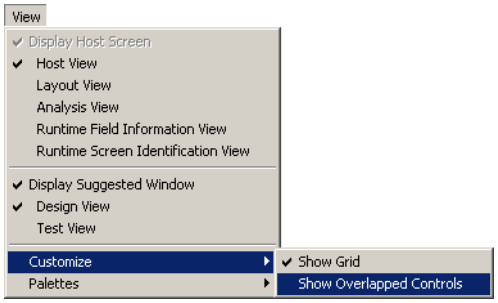
In runtime, the host application manages the display of controls, and typically there should not be overlapping controls.

Example 29. Highlighting overlapping controls

A single screen may have a different header, depending on whether the screen is currently in Edit or in View mode. During the conversion of the screen, in Design View, both headers will appear in the window, and overlap. In runtime, however, only the suitable header will be displayed, depending on the current mode of the screen.

To show or hide animated highlighting of overlapping controls:

- From the *View* menu select *Customize* > *Show Overlapped Controls*.



The Design Menu

Most of the options in the *Design* menu are window and control-editing functions. In addition, there are also options for creating and editing methods.

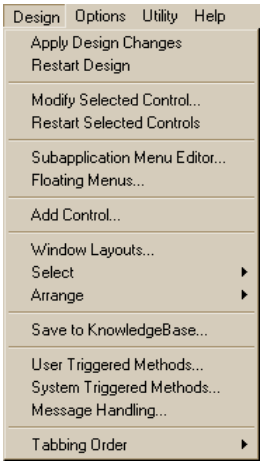


Figure 65. Design View menu options

The following table describes each of the *Design* menu options:

Apply Design Changes	Refreshes the window, taking into account the current status of the KnowledgeBase and retaining all local modifications.
Restart Design	Clears all local modifications but retains the current status of the KnowledgeBase.
Modify Selected Control	Accesses the *Control Component* dialog box of the selected control.
Restart Selected Controls	Clear selected controls of all local modifications but retains the current status of the KnowlegeBase.
Subapplication Menu Editor	Accesses the *Menu Editor* dialog box.
Floating Menus	Opens a submenu to create, open, or attach functionality to floating menus.

Add Control

Adds a single control to a window. If the control is a trigger, a method must be added manually.

Window Layouts

Prompts a list of Window Layout Definitions. You can edit or create Window Layouts, and also apply a Window Layout to the current window.

Select

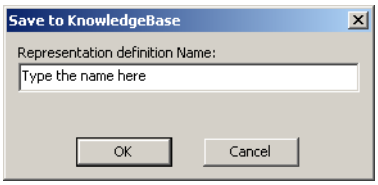
Opens a submenu of criteria you can use to select controls on the window.

Arrange

Opens a submenu of control editing functions.

Save to Knowledgebase

Saves the selected user-added controls in the window as the components of a new floating Representation Definition. Type the name of the definition in the *Save to KnowledgeBase* dialog box. The new definition is added to the KnowledgeBase and to the *Definitions Palette*.



User-Triggered Methods

Accesses a dialog box for creating and editing User-Triggered Methods.

System-Triggered Methods

Accesses a dialog box for modifying UT Methods.

Message Handling

Accesses the Message Handling dialog box.

Tabbing Order

Opens a menu that allows you to enter Modify Tabbing Order mode or reset the current tabbing order.

Control Editing

The following sections describe how to edit controls in the window. The following topics are presented:

- Selecting Controls in the Window
- Arranging Controls on the Window
- Adding Controls to the Window

Selecting Controls in the Window

In Design View objects can be selected in several ways:

Individually	A single object can be selected. Additional objects can be added individually to the selection.
In groups	All the controls can be selected or a group of controls can be selected.

Selecting Controls Individually

To select controls individually in the window, use one or more of the following options:

- To select a single control, click on it.

  A selected control is displayed with sizing handles. You can use the sizing handles to resize the control with the mouse.

Figure 66. Selecting controls individually

- To add controls to a selection, one by one, press the *Shift* key while clicking on each control.
- To remove a single control from a selection, press the *Shift* key and click the control.
- To clear all selected controls, click on an empty space in the window's client area.

Note: Clicking a selected control designates this control as the leading control. For more information, see "Leading Controls" on page 176.

Selecting Groups of Controls

The easiest way to select controls on a Window is by using the mouse. Simply click on the window client area and drag the pointer around the controls to be selected. Dragging the pointer across the window marks a rectangular area. Any control that is intersected by this rectangle is selected.

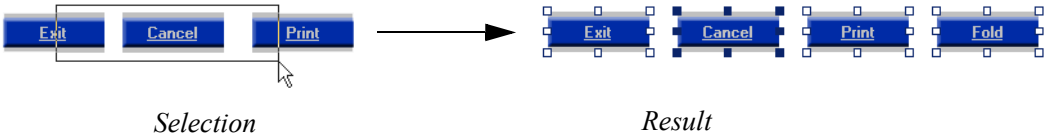


Figure 67. Selecting groups of controls

In addition to using the mouse, it is also possible to use the options in the *Design* > *Select* menu to select more than one control at a time:

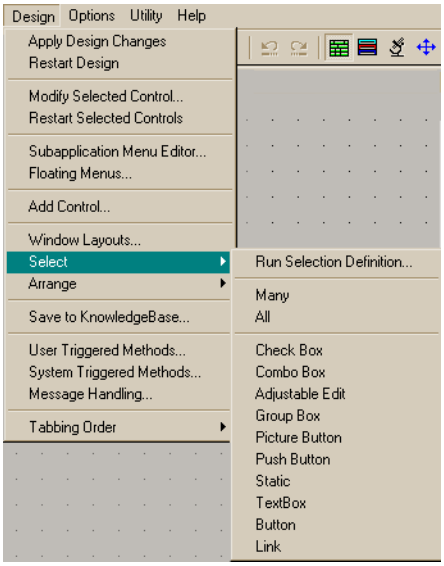


Figure 68. Select menu options

Selecting one of the *Select* menu's options changes the cursor to a cross-hair. Drag this cursor across the window to mark a rectangular area on the screen. The controls that are intersected by this rectangle and that meet the requirements of the selection option are selected.

Note: To cancel the cross-hair cursor, press the *Esc* key.

The following table describes each of the *Select* menu options:

Run Selection Definition	Selects controls based on multiple criteria. This item opens the *Selection Definitions* manager where you choose a Selection Definition to run. You can also create and edit Selection Definitions.
Many	Selects all the controls within the rectangle. This option can also be activated by pressing the *Insert* key when the client window is in focus.
All	Selects all the controls.
Check Box	Selects all the check boxes within the rectangle.
Combo Box	Selects all the combo boxes within the rectangle.
Adjustable Edit	Selects all the adjustable edits within the rectangle.
Group Box	Selects all the group boxes within the rectangle.
Picture Button	Selects all the picture buttons within the rectangle.
Push Button	Selects all the push buttons within the rectangle.
Static	Selects all the static controls within the rectangle.
TextBox	Selects all the textboxes within the rectangle.
Button	Selects all the buttons within the rectangle.
Link	Selects all the links within the rectangle.

- To clear all selected controls, click on an empty space in the window's client area.
- To add a group of controls to a selection, press the *Shift* key while dragging the cross-shaped cursor. You can also continue to select additional controls one by one, by pressing the *Shift* key and clicking each additional control.

Note: For information on positioning selected controls, see"Positioning Controls" on page 191.

Leading Controls

When applying an editing option on a group of selected controls, the option may need to refer to the leading control. For example, when a group of controls are resized they assume the size of the leading control.

The leading control is the control in the group that has focus. By default, the leading control is the last control selected. However, it is possible to select a different leading control simply by clicking it.

The leading control is visually indicated by emphasized sizing handles.

Example 30. Leading controls

In the example below, the Cancel button is the leading control.

Arranging Controls on the Window

ACE provides a wide range of options for designing your GUI, both automatically using the KnowledgeBase, and locally using Design View options.

In general, it is recommended to set as many standards as possible via the KnowledgeBase, using Representation Definitions and Window Layouts. However, in some cases, you may want to enhance a specific set of windows, or controls, beyond the properties provided by the KnowledgeBase.

In Design View, there are numerous options to help you arrange the controls on the GUI, such as alignment, resizing, centering and spacing options, cut, copy and paste options, and more. It is also possible to determine font and color settings. All these options are local editing options that affect the current window only.

Note: For more information on Representation Definitions, see "Editing a Representation Definition" on page 203. For more information on Window Layouts, see *JIS Interface Server: Advanced Topics*.

In Design View there are several ways to access and activate the options for editing and arranging controls:

- The *Control Editing* palette.
- The *Design* > *Arrange* menu.
- The shortcut menu that opens as a result of a right mouse button click on the window client area.

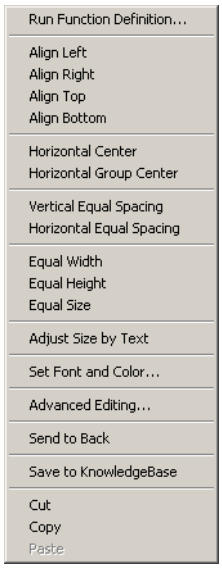


Figure 69. Local editing options in Design View

The following sections describe the control editing and arranging options, using the *Control Editing* palette.

Control Editing Palette

The *Control Editing* palette contains the following options:

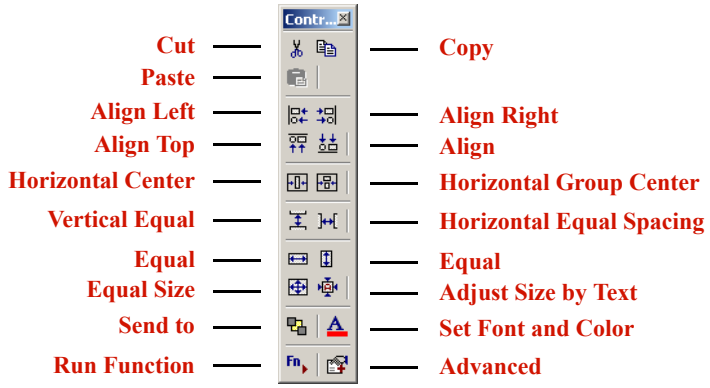


Figure 70. Control Editing Palette options

Note: The palette can be resized; move the pointer over any edge until it changes to a double-headed arrow, and then drag the edge of the toolbar. The palette can be configured to remain always on top of the window. Right click the title bar and select *Always on Top*).

One or more controls must be selected in order for the control editing options to be enabled.

The following table describes each of the control editing options:

Align Left	Aligns the selected controls to the left according to the left edge of the leading control.
Align Right	Aligns the selected controls to the right according to the right edge of the leading control.
Align Top	Aligns the selected controls according to the top edge of the leading control.
Align Bottom	Aligns the selected controls to the bottom edge of the leading control.
Horizontal Center	Places each of the selected controls in the horizontal center of the window's client area. To center several controls that are located on the same horizontal line, use the Horizontal Group Center option.
Horizontal Group Center	Places the selected controls (as a group) in the horizontal center of the window's client area.
Vertical Equal Spacing	Spaces the selected controls at equal vertical distances.
Horizontal Equal Spacing	Spaces the selected controls at equal horizontal distances.
Equal Width	Resizes the selected controls to an equal width, according to the width of the leading control.
Equal Height	Resizes the selected controls to an equal height, according to the height of the leading control.

Equal Size	Resizes the selected controls to an equal size, according to the size of the leading control.
Adjust Size by Text	Resizes the selected controls according to the length of the text. For more information, see “Adjusting a Control’s Size to its Text” on page 179.
Send to Back	When selected controls overlap, sends the top control to the back and makes the next control visible. Does not affect the runtime.
Set Font and Color	Opens a dialog box in which you specify font and color settings for the selected control/s. For more information, see Chapter 11 - "Color and Font Design" on page 307.
Run Function Definition	Applies multiple actions to selected controls. This item opens the *Function Definitions* manager. Use the *Function Definitions* manager to run, create, and edit Function Definitions. For more information, see *JIS Interface Server: Advanced Topics*.
Advanced Editing	Additional editing features. For more information, see “Advanced Editing” on page 180.

Adjusting a Control’s Size to its Text

ACE objects can be resized with a single command, to suit the text on the object. This is important if the text is longer than the size of an object or if the font used causes the text to exceed the size of the object. This function can also be used to reduce the size of an object in the same way.

Example 31. Adjusting a control’s size to it’s text

In this example, a button needs to be enlarged to accommodate the following text: “this text is too long for the button.”

At first, only part of text is visible:

To adjust the control to automatically accommodate the text:

1 Select the button.

2 From the *Design* menu select *Arrange>Adjust Size by Text*.
The button is automatically resized to suit the text, and appears as follows:

Advanced Editing

The *Advanced Editing* option opens the *Control Editing* dialog box, in which you can specify various options to operate on the selected controls.

The advantage of the Advanced Editing options is that you can specify several options at once.

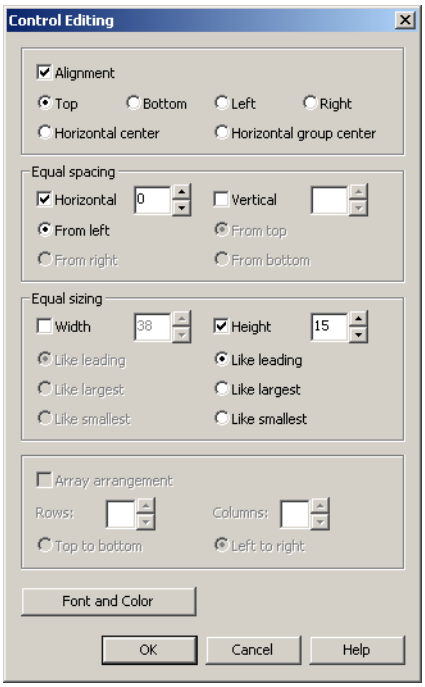


Figure 71. Advanced Editing dialog box

Alignment	Top, Bottom, Left, Right, Horizontal Center, Horizontal Group Center. Specify the alignment of the controls.
Equal Spacing	Horizontal, Vertical. Specify the exact distance, in dialog units, between the controls. Using both Horizontal and Vertical spacing at once, arranges the controls in a diagonal line.

Equal Sizing	Width, Height. Specify the exact width and/or height of the controls, in dialog units.
Font and Color	Specify the font and color.

Adding Controls to the Window

In Design View, it is possible to add controls to the window. This can be done in one of two ways:

- Using the *Definitions Palette* to drag Floating Representations onto the window.
- Using the *Design* > *Add Control* option.

The advantage of using Floating Representations is that they are predefined in the KnowledgeBase with all the desired characteristics, and can be used repeatedly. Furthermore, you can change the properties of such a definition once, and your changes are automatically applied from this point onward. In contrast, when using the *Add Control* option, you need to specify the properties of each control you add from scratch.

The following sections describe both ways of adding controls to the window.

Definitions Palette

The *Definitions Palette* is used to add controls to the GUI:

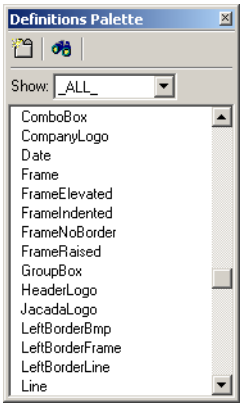


Figure 72. Definitions Palette

A control can be added to the GUI by dragging it from the palette onto the window. The added control is selected and you can drag it to the desired location, resize it using the sizing handles, or edit its properties in the *Window Components* palette. For more information, see “Window Components Palette” on page 184.

When a new Floating Representation control is defined in the KnowledgeBase, it is added to the list in the *Definitions Palette*. For more information, see "Editing a Representation Definition" on page 203.

Customizing the List of Displayed Definitions

The palette displays all Floating Representation Definitions by default (_ALL_). This can be quite an exhaustive list. To customize the list of definitions that is displayed, select the desired display criteria from the *Show* combo box.

Only the definitions that fit the selected criteria are displayed in the list.

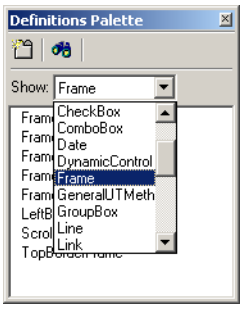


Figure 73. Filter definitions in Definitions palette

New display criteria can be added in Representation Definitions View in the KnowledgeBase:

Figure 74. Display criteria added in RD View in the KnowledgeBase

Adding a Tab Control

To add a Tab control to the window, click in the *Definitions Palette*. This activates the *Create Tab* wizard which guides you through the steps necessary to create a Tab control and adds it to the window.

Finding a Definition in the List

1. Click the right mouse button in the *Definitions Palette* and select *Find* from the shortcut menu, or use the icon, or press *Ctrl+F*. The *Find* dialog box opens.
2. Type the search text, and the first most suitable match is highlighted in the list. For example, typing `lin` in the *Find* field highlights Line in the palette:

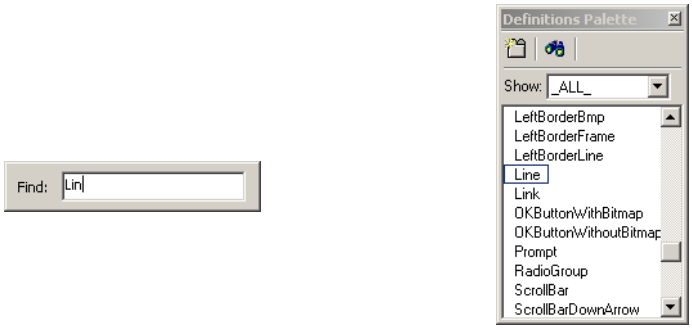


Figure 75. Finding a specific definition in the Definitions Palette

Modifying a Definition in the List

To modify a definition in the list:

1. Click the right mouse button on the desired definition in the list and select *Modify* from the shortcut menu:
2. The KnowledgeBase opens, displaying the Floating Representation Definition, ready for modification.
3. Modify the definition properties as necessary.

   From this point onward, when dragged onto the window, the definition creates the control with its updated properties.

User-Added Controls

The initial window displays only controls that are defined by Representation Definitions. These are controls that are derived from patterns in the screen. You can use the *Definitions Palette* to drag control definitions onto the window. However, it is also possible to add a control that is not derived from the KnowledgeBase. These controls are referred to as User-Added Controls. A user-added control is helpful when you want to add a specific or unique control that is not defined in the KnowledgeBase.

Adding a Control

To add a control:

1. From the *Design* menu, choose *Add Control*.

   The *Add Control* dialog box opens.
2. From the *Control type* combo box choose a control and click *OK*.

> **Note:** Selecting the Tabs control, activates the *Create Tab* wizard which guides you through the steps necessary to create a Tab control and adds it to the window.

3. The *Component* dialog box corresponding to the chosen control opens.
4. Use the *Style*, *Manager*, *Events*, *Format* and *Screen* tabs to specify the control's properties. For more information, see "Control Component Dialog Box" on page 190.
5. Click *OK*.

   The control is added to the top left corner of the window.
6. Position the control in its desired location.
7. Click the toolbar button or *Design* > *Apply Design Changes* to apply the change.

Modifying Control Properties

The properties of a control derived from the KnowledgeBase are determined by its Representation Definition. However, it is possible to change the properties of each control locally in Design View.

There are several ways to change the properties of a control in Design View:

- Using the *Window Components* palette

  This palette enables you to modify the style properties of a control, its size and location.
- Using the control's *Component* dialog box.

  This dialog box is used to view and modify the properties of a control. The information in the dialog box is divided into tabs. Note that the properties in the *Style* tab can also be modified in the *Window Components* palette.

1. By moving or resizing the control in the window.

   You can drag the control to the desired location, and you can resize it using its sizing handles.

The following sections provide more details on each of these options.

Window Components Palette

The *Window Components* palette is used to make local changes to the properties of a control, including it style, size and location.

The *Window Components* palette is composed of two panes. The top pane contains a list of all the components that comprise the current window. The list is divided into three categories: Controls, Accelerators, and Menus, and in each category the items are arranged alphabetically. Each item in the list has a small icon to indicate its type, followed by the component's variable name. This pane is referred to as the *Components List* pane.

The bottom pane displays information on the currently selected component, and enables you to modify this information. This pane is referred to as the *Component Properties* pane.

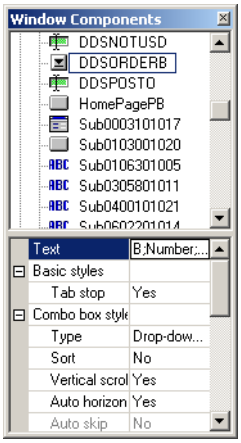


Figure 76. Window Components Palette

> **Note:** Most window components are actual controls that can be seen in the window. However, some components, such as Accelerators and Variables do not have a visual representation in the window.

Some facts about the *Window Components* palette:

- Clicking on a control's name in the list selects the control in the window and clicking on a control in the window, selects the control's name in the list. Accelerators, Menus and controls of Variable type are not selected in the window.
- When a group of controls is selected in the window, the leading control component is selected in the palette.
- Double-clicking a component's name in the list, double-clicking a control in the window, or choosing a menu item, opens its *Component* dialog box. For more information, see "Control Component Dialog Box" on page 190.
- The palette can be shown or hidden using the *View* > *Palettes* option, or using the palette's toolbar icon.
- The palette can be resized.
- To configure the palette to remain always on top of the window, right click the palette's title bar and select *Always on Top* from the shortcut menu.

Component Properties Pane

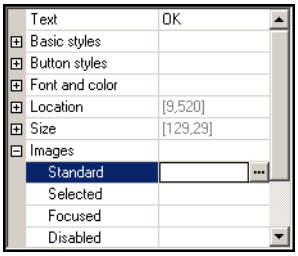
The component properties pane has two columns.The left column lists the control's properties in a tree structure. The right column displays the current value of the property.

The value column can contain the following controls:

- TextBox - for typing text directly.
- Combo Box - for Yes/No or other multiple choice selections.
- A field followed by an 'open-dialog' button - ...
  This button opens a dialog box in which you can specify additional information. For example, select the image file to be displayed on a button.

Example 32. Component properties pane

The image below displays some of the properties of an OK button. The highlighted item shows the 'open dialog' button. When clicked, this button opens a dialog box in which you can select an image to be displayed on the button when in its standard state:



The information displayed in the component properties pane depends upon the type of the selected component. There are properties, such as Text, Font and Color, and Location and Size, which are common to many control types. On the other hand, some properties are unique to a certain control. For example, the image attached to a button or the key/s attached to an accelerator.

The following table describes some of the common properties:

Text	The control's text displayed in the window.
Basic styles	These typically include: Initial capitals - select *Yes* to display uppercase text with initial capital letters only. Tab stop - select *Yes* to cause the cursor to stop at this control when tabbing in the window in runtime.
Control-specific styles	These properties depend on the control type.
Font and color	View the current font and color settings. To change these properties, click the 'open dialog' button located next to the Font and color item in the list. This opens the *Font and Color* dialog box.
Location	Specify the location of the control's top left corner in dialog units relative to the top left corner of the window's client area.
Size	Specify the width and the height of the control in dialog units.

Variable Component Properties

A variable is an abstract component which serves to uniquely identify a Subapplication component. Therefore, it has no style properties and no location or size. When a variable is selected in the components list, the properties pane remains empty.

A variable component has the following icon in the components list: .

To modify a variable component:

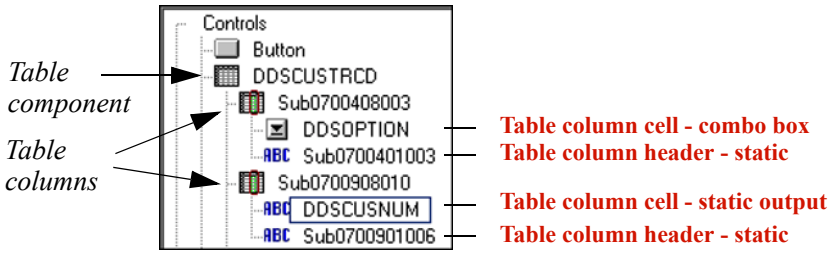
Right click on a variable component in the *Window Components* list and select *Modify* from the shortcut menu. The *Variable Component* dialog box opens, with the *Format* tab in focus.

Table Component Properties

A table representation is hierarchical. A table is composed of table columns, and each table column is composed of a table column item, and one or more table column headers, according to the number of header lines.

Example 33. Table component properties

In the list below, the table component has 2 columns. Each table column is composed of a table column cell and a table column header.



Note: A table column, such as Sub0700408003, is an abstract entity that has no properties. Therefore, when selected in the list, the properties pane remains empty.

Deleting Components

To delete a component:

1. In the *Window Components* palette, select the component you wish to delete.
2. Press the *Delete* key

   -or-

   Click the right-mouse button and select *Delete* from the shortcut menu.
3. You are asked to confirm deletion, click *Yes*.

Note: A control can be also deleted by selecting it in the window and pressing the *Delete* key.

Renaming Components

The name of a component in the component's list is its variable name. A variable uniquely identifies a Subapplication component. By default, the variable name is either derived from a Representation Definition, or else it is allocated automatically the first time the window is created. In some cases, you may wish to change the automatic variable name to a more user-friendly name.

To rename a component:

1. In the *Window Components* palette, select the component you wish to rename.
2. Press the *F2* key

   -or-

   Click the right-mouse button and select *Rename* from the shortcut menu.
3. The component name becomes editable. Type in the new name and press *Enter*.

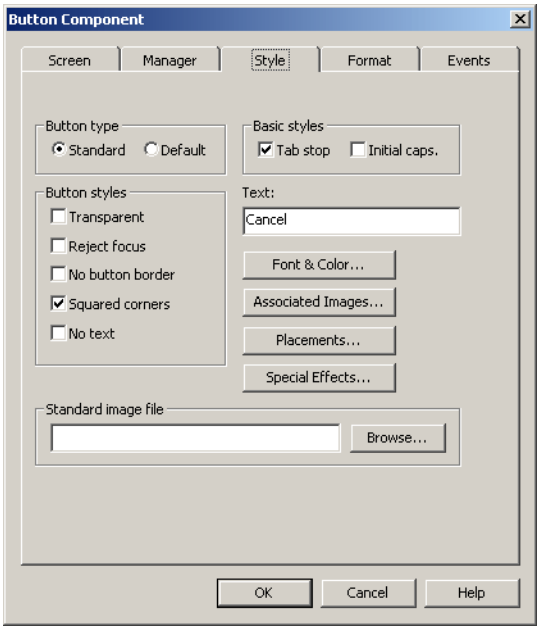
Note: The variable name of a component can also be changed in the *Manager* tab of the control's *Component* dialog box.

For more information on variable names, see "Variable Names" on page 218.

Control Component Dialog Box

A control's *Component* dialog box is used to view and modify the properties of a control. The information in the dialog box is divided into tabs.

This is an example of a Cancel button's *Component* dialog box, with the *Style* tab in focus:



The *Component* dialog box has between three to five active tabs, depending on the control type. For example, the *Screen* tab only appears for controls derived from the KnowledgeBase, and does not appear for user-added controls. The *Events* tab is enabled only when the control is a method trigger, such as a button control.

The tabs are used as follows:

- *Style* tab - used to specify the control styles and properties such as text, font and color, and more.

> **Note:** Control style properties can also be configured in the *Window Components* palette. For more information, see "Component Properties Pane" on page 185.

- *Manager* tab - contains runtime data flow and variable name information. The default variable name can be changed. The limit is 35 characters.
- *Events* tab - for method triggers only, such as a button. This tab displays the method currently attached to the control. It is possible to link a different method, and modify or create new methods.

- *Format* tab - used to define text formatting for the control.
- *Screen* tab - for pattern-linked controls only, indicates which part of the pattern is represented by the control.

All changes made in the *Component* dialog box are local to the specific control.

The *Component* properties dialog box is similar to the *Properties* dialog box for Representation Definitions in the KnowledgeBase. For detailed information on the tabs, see "Representation Component Properties" on page 211.

To access the *Component* properties dialog box:

- In the *Window Components* palette, double-click the component in the list

  -or-

  Right click the component and select *Modify*.

  -or-

  Double click the control on the window.
- For menu items - select the menu item in the window.
- To open the window's *Component* dialog box, double-click on a control-free window client area.
- To modify a List Column component, double-click within the cell of the topmost column item. The *Component* dialog box that opens depends on the type of the column. For example, double-clicking within an input column, opens the *TextBox Component* dialog box.
- To modify a List Column Header component, double-click within the header button in the window. Since column headers are typically static, in most cases the *Static Component* dialog box opens.

Accelerators and Variables cannot be accessed this way.

For most style modifications, you may prefer the *Properties* pane in the *Window Components* palette to the *Component* dialog box. However, some properties can only be accessed via the *Component* dialog box. For example, text formatting which is done in the *Format* tab, linking methods to controls which is done in the *Events* tab, and specifying runtime data flow which is done in the *Manager* tab.

Positioning Controls

There are several ways to position controls, either as a group or individually.

When working with a group of controls, the control editing options, such as aligning, spacing or centering, help you perform sophisticated tasks automatically. For example, centering a group of controls, or aligning a group of controls according to the location of a specific control. For more information, see "Arranging Controls on the Window" on page 176.

To position a single control, you can specify values for the location of its top left corner, under the *Location* property in the *Window Components* palette.

However, in some cases, you may want to position controls simply by dragging them in the window. You can drag a single control or a group of selected controls to the desired location in the window.

There are also ways to move selected controls in fixed increments, using the keyboard arrow keys and depending on the snap to grid option:

- When snap to grid is turned off:
  - Each time you press an arrow key, the selected controls move one dialog unit in the arrow's direction.

    To move the controls ten dialog units at a time, press the *Shift* key together with the arrow keys.
- When the snap to grid option is turned on the controls move according to grid line intervals:
  - Each time you press an arrow key, the top left corner of the selected control or group of controls moves to the closest grid line intersection, in the arrow's direction.

    To move the controls by ten grid line intervals, press the *Shift* key together with the arrow keys.

Note: The window must be in focus (click on its title bar).

When you are satisfied with the results, click the button, or select *Design* > *Apply Design Changes*.

Sizing Controls

There are several ways to size controls, either as a group or individually.

When working with a group of controls, the control editing options, such as equal width, equal height and equal size help you perform sophisticated tasks automatically. For example, you may wish to resize a set of buttons to the same size, according to the size of the largest button in the group. For more information, see "Arranging Controls on the Window" on page 176.

To size a single control, you can specify width and height values under the *Size* property in the *Window Components* palette.

However, it is also possible to size a single control directly in the window, by dragging its sizing handles. Note that when the snap to grid option is turned on, dragging the sizing handles makes the control resize to the nearest grid line/s.

When you are satisfied with the results, click the button, or select *Design* > *Apply Design Changes*.

Undo Repositioning or Resizing

Press *Esc* while positioning or sizing controls to undo the operation.

In addition to Design View, this option is also available when working with sections in Layout View and in the *New Subapplication* wizard.

Design View Shortcut Keys

The following shortcut keys are available in Design View:

Table 10. Shortcut keys available in Design View

Key	Function
F12	Press *F12* to bring all the Design View palettes to the front of the window. *F12* is useful when your window is very large.
F5/Shift+F5	Align the selected controls to the right/left of the leading control.
F7/Shift+F7	Arrange the selected controls in equal horizontal/ vertical spacing.
F8	Size all selected controls according to the size of the leading control.

Undoing Changes

All local changes made in Design View, and table column manipulations made in Test View, can be undone.

The Undo option reverses the most recent operation. It is complemented by a redo option, that reinstates the most recent undo operation. In this manner, you can undo or redo operations one by one.

The undo/redo features only act on changes made in the current Subapplication session.

Note: In contrast to Undo, the *Design > Restart Design* option enables you to remove *all* local modification *ever* made to a window, regardless of when they were made. This means that the window reverts back to its initial state upon conversion. Use this option when you want to start all your local modifications of a certain window from scratch.

The *Design > Restart Selected Controls* option enables you to remove *all* local modification *ever* made to a specific controls in a given window, regardless of when they were made.

Undo/Redo Behavior

The undo/redo features have the following behavior:

- Undo reverses the most recent operation. The following Undo reverses the next most recent operation, and so on.
  Note that the most recent operation may be a redo operation.
- A control affected by an Undo operation remains selected in the window.
- Redo reinstates the most recent undo operation. The following Redo reinstates the next most recent Undo operation, and so on.
- A control affected by a Redo operation remains selected in the window.
- Performing a new operation on the window clears the Redo list, but does not affect the Undo list.
- Once you close a Subapplication, either by opening a different Subapplication, a different Application or library, or closing ACE completely, the Subapplication's undo/redo memory is erased. Next time you open the Subapplication, there are no operations to undo/redo.
- Some Undo operations require performing an *Apply Design Changes* action immediately afterwards, to update the Subapplication.
  If a particular Undo operation requires an apply, you are asked whether or not you wish to proceed with the Undo.

Performing an Undo Operation

You can undo operations in Design View or Test View, in any one of three ways:

From the Toolbar

Click the toolbar *Undo* button:

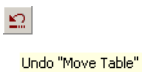


Figure 77. Undo from the toolbar

Before you Undo, the balloon help informs you which operation will be undone.

From the Edit Menu

From the *Edit* menu select *Undo*:

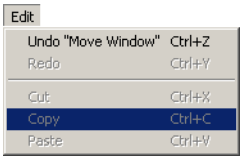


Figure 78. Undo from the Edit menu

The text to the right of the Undo menu item informs you which operation will be undone.

From the Keyboard

Press *Ctrl+Z*.

Performing a Redo Operation

You can redo operations in Design View or Test View, in any one of three ways:

From the Toolbar

Click the toolbar *Redo* button:

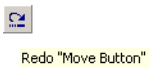


Figure 79. Redo from the toolbar

Before you Redo, the balloon help informs you which operation will be redone.

From the Edit Menu

From the *Edit* menu select *Redo*:

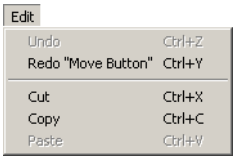


Figure 80. Redo from the Edit menu

The text to the right of the *Redo* menu item informs you which operation will be redone.

From the Keyboard

Press *Ctrl+Y*.

Undo Limitations

The Undo feature has certain limitations:

Changes May Clear the Undo List

Certain changes clear the Undo list. This happens when the change makes the Undo list irrelevant. For example, a KnowledgeBase modification that replaces one control with another makes any changes to the original control irrelevant.

Manual Apply May be Required

Some undo/redo operations require an *Apply Design Changes* before they take effect, but you are not prompted to perform the Apply action.

In such cases, you need to click the *Apply Design Changes* button:

Operations for which undo/redo does not take effect immediately are formatting combo boxes and manipulating table columns in Test View.

Affected Controls May Not Remain Selected

Sometimes, when an undo prompts an Apply, the control affected by the Undo does not remain selected following the Apply.

Attaching Functionality to Controls

In addition to arranging and editing controls, it is also possible to work with methods in Design View:

- You can modify or create new methods.
- You can attach a method to a specific control, in order to determine its functionality in runtime.

In contrast to attaching a method to a Representation Definition in the KnowledgeBase, attaching a method to a specific control in Design View is a local modification. Typically, this local option is used to attach functionality to user-added controls.

Attaching a Method to a Control

You attach a method to a control in the *Events* tab of the control's *Component* dialog box.

1. Open the control's *Component* dialog box.
   For more details, see "Control Component Dialog Box" on page 190.
2. Select the *Events* tab.
3. Select a method from the list.
4. Click *Link*.
5. Click *OK*.

Chapter 9. Representation Definitions

The KnowledgeBase contains a GUI representation for many Pattern Definitions. This defines the control that is used in the GUI window. The GUI representation is called a Representation Definition in the KnowledgeBase. You can make global changes to your Application by editing the Representation Definitions. The changes made to the Representation Definitions typically affect the controls' type, font, color, etc. You can access the Representation Definitions in the KnowledgeBase from any View. Changes that are made to a Representation Definition are global, affecting your entire Application.

This chapter describes:

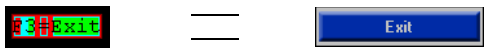
- Representation Definitions and How They Work
- Editing a Representation Definition
- Representation Component Properties

Representation Definitions and How They Work

Representation Definitions are the GUI image of Pattern Definitions. The KnowledgeBase contains many Pattern Definitions that have Representation Definitions attached to them. These Representation Definitions can be edited to better suit your application's needs.

Example 34. Representation definitions

The ACE KnowledgeBase contains the information that the `F3=Exit` character sequence at the bottom of the host screen is represented as a button.



Pattern Definition — **Representation Definition is a push button**

In the KnowledgeBase you can change the representation for the `F3=Exit` pattern to another GUI Representation. For example, a check box representation would appear as follows on the GUI screen:

This is, of course, an inappropriate representation for this pattern.

The KnowledgeBase contains what we consider to be the most appropriate representations for primary patterns. Through the KnowledgeBase you have full access to the Representation Definitions and you can change the representations that are displayed in your GUI Application.

Note: Modifications made to the Representation Definitions in the KnowledgeBase affect all the windows in the Application.

Components and Controls

How do you define the control (or controls) that will represent a Pattern Definition and where do you specify its style?

When defining a new Representation Definition, you specify the various details through a component. A Representation Definition has one or more components. There are various types of components. A component can define a control, a menu item, an accelerator or a method. This section deals with components that define controls.

Each component can define one control, but a Representation Definition can have more than one component that defines a control.

There are two ways to represent a Pattern Definition by more than one control:

- Define a Representation Definition with a component for each control.
- Define Representation Definitions for child patterns of the Pattern Definition to be represented. This way is more commonly-used.

Representation Definitions With More Than One Component

One way to represent a Pattern Definition by more than one control is to define a Representation Definition with more than one component. Each component of the Representation Definition defines a separate control.

Representation Definitions for Lower Level Pattern Definitions

The common way to represent a Pattern Definition by more than one control takes advantage of the hierarchical structure of Pattern Definitions. Instead of defining a Representation Definition with several controls, you define

Representation Definitions for the child patterns of the Pattern Definition to be represented. The parent Pattern Definition will be correctly represented in the window.

Example 35. Defining representations for lower level pattern definitions

The Pattern Definition *Header_Terminator_Fields* matches the screen pattern `Password:BBBBBBBB`

Instead of defining a Representation Definition with two components for the entire Pattern Definition, you define Representation Definitions for its child patterns.

The string `Password` is matched by the Header Pattern Definition, which is a child pattern of Header_Terminator_Fields. Therefore, we create a Representation Definition for Header that defines a static control to represent this Pattern Definition.

The password input field is matched by the PasswordField Pattern Definition, which is a child pattern of Header_Terminator_Fields. Therefore, we create a Representation Definition for PasswordField that defines a textbox control to represent this Pattern Definition.

The entire Pattern Definition is represented in *Windows* in the following way:

The advantage of attaching Representation Definitions to Pattern Definitions that are often used as child patterns, is that a particular Representation Definition must be defined only once. Any parent pattern of this often-used-as-a-child-pattern Pattern Definition will be properly represented.

Components and the Subapplication

Subapplications have the following structure:

Figure 81. Subapplication components

A component of a Representation Definition is related to all three parts of the Subapplication. So far we have discussed a component in relation to the Window section of a Subapplication. How is a component related to the Host Screen section of a Subapplication?

A Representation Definition defines the interpretation of a Pattern Definition. A Pattern Definition is composed of child patterns. A component may define the interpretation of the entire Pattern Definition, or of one of its child patterns.

Example 36. Components of the subapplication

The Pattern Definition:

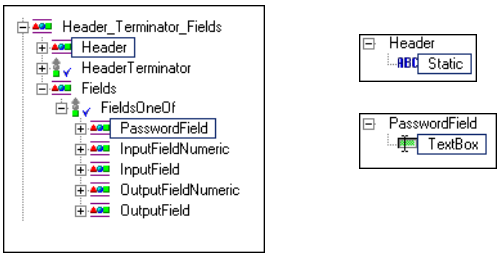
Header_Terminator_Fields

must be represented by two controls: a static control for the header text and an adjustable edit control for the input field.

This can be done either by creating a Representation Definition with two components, or by two separate Representation Definitions.

In both cases, the Pattern Definitions that will be represented by a control are: *Header*, and *PasswordField*.

The following diagram displays the structure of *Header_Terminator_Fields*:



How is a component related to the *Manager* section of a Subapplication?

- The Manager section of a Subapplication controls the interaction between the Screen and the Window, and between the Window and the Screen.
- A Representation Definition component defines the interaction between the Pattern Definition in the Screen and the control that represents it in the Window.

Multiple Components of the Same Type

The components of a Representation Definition are named according to the control type of the component. When a Representation Definition has more than one component of a given type, the names of the components receive a number suffix reflecting the order in which the components were added to the Representation Definition:

Figure 82. Multiple components of the same type

If a component is subsequently deleted, the number suffixes are recalculated to reflect the order of the remaining components of the same type.

Editing a Representation Definition

This section describes how representation definitions can be edited.

Accessing the Representation Definitions

Representation Definitions are in the KnowledgeBase. You can add Representation Definitions to the KnowledgeBase or modify existing Representation Definitions.

To access Representation Definitions:

1. Open the *KnowledgeBase Definitions* window by:

   Clicking the KnowledgeBase icon .

   -or-

   In Analysis View, choose any analyzed character sequence and double-click or press the *F8* key.

   -or-

   To access a particular floating Representation Definition directly, in Design View double-click that floating Representation Definition in the *Definitions Palette*. Note that no other Representation Definition is accessible when you enter the KnowledgeBase this way.
2. Click the button on the KnowledgeBase toolbar to switch to Representation Definition View.

The *Lower Representation* pane of the *KnowledgeBase Definitions* window contains a list of all the Representation Definitions. In this pane you can view or modify an existing Representation Definition, and also define a new Representation Definition. You can also modify a Representation Definition selected from the *Representation Components* area.

To view Representation Definitions:

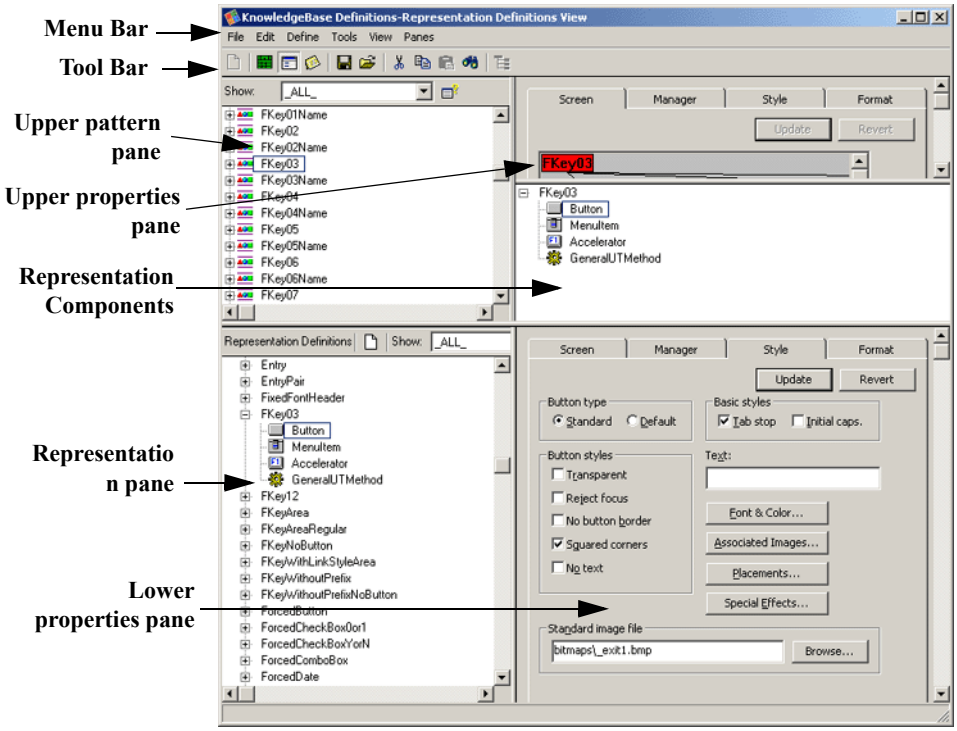


Figure 83. Representation Definition view in KnowledgeBase Definitions window

- When a Pattern Definition is selected in the *Upper Pattern* pane, the Representation Definition attached to it, if any, appears in the *Representation Components* area. The Representation Definition can be expanded to show its individual components.

  For example, the figure above shows the Pattern Definition *FKey03* selected in the *Upper Pattern* pane. Its Representation Definition appears in the *Representation Components* area.
- When a Representation Definition component in the *Representation Components* area is selected, its properties appear in the *Upper Properties* pane.

  The figure above shows the Button component selected in the *Representation Components* area. Its *Screen* properties are shown in the *Upper Properties* pane.
- All the Representation Definitions, both floating Representation Definitions and Representation Definitions attached to Pattern Definitions, are displayed in the *Representation* pane. Each Representation Definition can be expanded to show its individual components.
- When a Representation Definition component in the *Representation* pane is selected, its properties appear in the *Lower Properties* pane.
  The figure above shows the Button component selected in the *Representation* pane. Its *Style* properties are shown in the *Lower Properties* pane.

- Click on the ⊞ symbol beside a Representation Definition to display its components.
- Click on the ⊟ symbol beside a Representation Definition to hide its components.
- Use the scroll bars to scroll through the *Representation* pane or the *Representation Components* area. Scrolling does not affect which representation is selected.
- You can put the focus on the *Representation* pane and then type a letter to jump to the closest Representation Definition whose name begins with that letter. The Representation Definition that is jumped to becomes the selected representation.

Deleting a Representation Definition

You can delete Representation Definitions you created or supplied Representation Definitions that you have previously modified.

To delete a Representation Definition from the KnowledgeBase:

Open the *KnowledgeBase Definitions* window in Representation Definition View and:

1. In the *Representation* pane, or the *Representation Components* area, select the Representation Definition you wish to delete.
2. From the *Edit* menu choose *Cut*.

   -or-

   Click the *Cut* button on the Tool bar

   -or-
3. In the *Representation* pane, or the *Representation Components* area, right click the Representation Definition you wish to cut.
4. From the shortcut menu choose *Cut*.

Creating Representation Definitions and Components

To create a new Representation Definition for a Pattern Definition or to add a new Representation Definition component to an existing Representation Definition:

1. Open the *KnowledgeBase Definitions* window in Representation Definition View.
2. From the *Upper Pattern* pane select a Pattern Definition, or from either the *Representation Components* area or the *Representation* pane select an existing pattern-linked Representation Definition.
3. From the *Define* menu choose *New*. The *New Representation Component* dialog box opens:

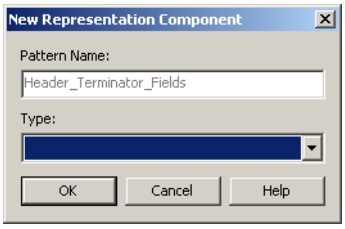


Figure 84. New Representation dialog box

4 From the *Type* drop down list choose the type of the new component for the Representation Definition.

5 Click *OK*. The *New Representation Component* dialog box closes and the new Representation Definition is created or the existing Representation Definition is updated.

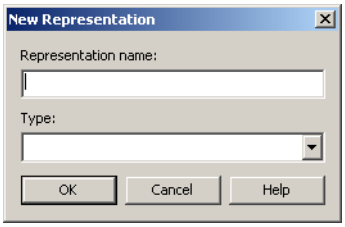
Creating Floating Representation Definitions and Components

Floating Representations are the basis for controls you add manually in Design View. Floating Representations appear in the Design View *Definitions Palette*.

To create a new floating Representation Definition or to add a new Representation Definition component to an existing floating Representation Definition:

1 Open the *KnowledgeBase Definitions* window in Representation Definition View.

2 From the *Define* menu choose *New Floating Representation*. The *New Representation* dialog box opens.



3 For a new floating Representation Definition, in the *Representation name* field, enter the name you wish to give to the new floating Representation Definition. For an existing floating Representation Definition do not edit the representation name.

4 From the *Type* list choose the type of the new component for the floating Representation Definition.

5 Click *OK*. The *New Representation* dialog box closes and the new floating Representation Definition is created or the new component is added to the existing floating Representation Definition.

Deleting a Component

To delete a component from a Representation Definition or a floating Representation Definition:

1. Open the *KnowledgeBase Definitions* window in Representation Definition View and make the component to be deleted visible in either the *Representation* pane, or the *Representation Components* area.
2. Select the component to be deleted.
3. From the *Edit* menu choose *Cut*.

   -or-

   Right click the selected component and choose*Cut* from the shortcut menu.

   -or-

   Click on the Tool bar. A confirmation box opens.
4. Click *Yes*. The confirmation box closes and the selected component is deleted from its Representation Definition.

Renaming Floating Representations

Floating representations can be renamed by right clicking on a main representation or pressing *F2* when standing on a main representation.

To rename a floating Representation:

1. Open the KnowledgeBase
2. Switch to Representation View.
3. Select the floating Representation for renaming.
4. Right click on the floating Representation name and select *Rename*.
   -or-

   Highlight the floating Representation and press *F2*.

   The name can now be edited.
5. Type a new name in the edit box and press *Enter*. ACE asks for confirmation. Press *Yes* to confirm. If the name is already in use an error message is displayed.

Note: A floating Representation supplied with ACE cannot be renamed. Renaming is supported for only those floating Representations that have been added by the developer.

Relative Placement of Floating Representation Controls

The properties of complex floating Representations (floating Representations with more than one component) can be modified in the KnowledgeBase for automatic placement of the components as they are added to the GUI window.

When adding the components of a floating Representation ACE first places them in the GUI window according to the information in the *Component Arrangement* tab in the *Lower Properties* pane. In this tab, you use a simple setting in which you choose between two options that indicate how to place the components relative to each other.

After the initial placement of the components has been determined by this setting, you can exert additional control on the placement and other properties of the floating Representation such as fonts, colors, sizing, and more. This is done by activating a Function Definition after placement of the controls. This option uses an advanced design feature and is a powerful tool for automatic design of floating Representations.

Note: Function Definitions are not covered in this book. For a detailed explanation on this subject see the chapter on function definitions in *JIS Interface Server: Advanced Topics*.

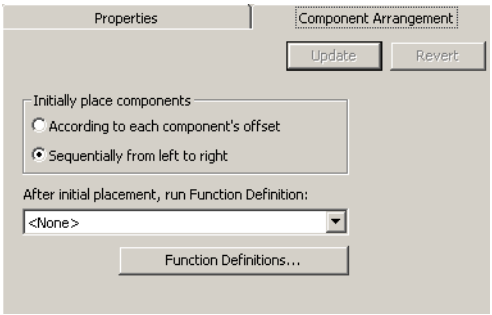


Figure 85. Component Arrangement tab in Representation Definitions view

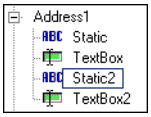
The two options for the initial arrangement of the components on the GUI window are the following:

- *Sequentially from left to right*
- *According to each component's offset*

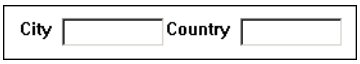
Sequentially from left to right - This is the default option. The components are arranged in one row according to the order in which they appear in the Floating Representation Definition.

Example 37. Relative placement

The floating representation *Address1* consists of a Static and a TextBox followed by another Static and TextBox.



Assuming the default option for their initial placement has not been changed, when dragged from the *Definitions Palette* onto the GUI window they appear like this:



According to each component's offset - This option places the components according to the offset assigned to each component under its *Offset/Size* tab, as described below.

Defining Offset And Size

Offset and size are modified for each individual component in the *Offset/Size* tab in the *Lower Properties* pane. The unit of measurement for offset and size is pixels. Changing the offset moves the control's upper left corner from the representation's anchor point by the specified value. Changing the size accommodates larger fonts or long text strings. Control size is changed by values entered for height and width.

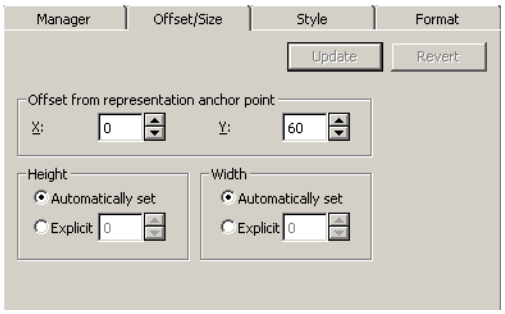


Figure 86. Defining offset and size

Example 38. Offset and size

Change the offset for the components in the floating Representation *Address1*, so that the static control Country is placed directly below City and each static control is followed by its TextBox. To obtain this layout, the offset for TextBox, Static2 and Textbox2 must be modified.

To change the offset for the components of Address1:

1. Go to the Representation Definition *Address1* in the KnowledgeBase.
2. In the *Component Arrangement* tab, select the option: *According to each component's offset*. If the other option is selected, all changes in offset settings will be ignored.
3. Select the component TextBox.
4. In the *Offset/Size* tab, change the X axis of the offset to 50. Press *Update*.
5. Select the component Static2.
6. In the *Offset/Size* tab change the Y axis of the offset to 30. Press *Update*.
7. Select the component Textbox2.
8. In the *Offset/Size* tab, change the X axis of the offset to 50 and the Y axis of the offset to 30. Press *Update*.
9. Return to the Subapplication and apply the design change. The controls for *Address1* are rearranged according to the new offset.

Control Placement Using Function Definitions

Function Definitions will not be explained in detail here. However, it is useful to have a basic understanding of the advantages of using Function Definitions over changing Offset/Size measurements.

A *Function Definition* is a rule for placing, sizing, or performing other design procedures on controls. A *Selection Definition* is a rule for selecting one or more controls on a GUI window according to specific criteria.

Window Layouts define the layout of controls on the GUI window. A Window Layout consists of Selection Definitions paired with Function Definitions. ACE comes with several types of predefined Selection Definitions and Function Definitions, but you can create your own in order to make a unique Window Layout.

One type of Function Definition is the *Apply Window Layout* type. This type of Function Definition applies the instructions contained in a Window Layout to the selected controls.

An Apply Window Layout type of Function Definition can be applied to the floating Representation. Using this type of Function Definition greatly expands the power of this feature by making it possible to automatically apply additional design elements such as fonts, colors, and any other appropriate design feature. Modification of control size and offset can also be included in the Function Definition.

Using the Window Layout process in the Function Definition is advantageous because control placement is relative, whereas placement using the *Offset/Size* tab is absolute. In addition, various other design activities can be performed automatically.

> **Note:** Understanding this advanced feature requires prior knowledge of Window Layouts, Selection Definitions, and Function Definitions. See the section entitled "About Window Layouts" in *JIS Interface Server: Advanced Topics*.

To use a Function Definition for Control Placement:

1. In the KnowledgeBase, write a new Function Definition of the Apply Window Layout type.
2. In the *Component Arrangement* tab, select *According to each component's offset.*
3. Select the new Function Definition from the combo box. Press *Update*.

Representation Component Properties

A Representation Definition component has different properties depending on the Representation Definition it belongs to. You can view and edit these properties in the *KnowledgeBase Definitions* window in Representation Definition View.

- Select a representation component in the *Representation Components* area to display its properties in the *Upper Properties* pane.
- Select a representation component in the *Representation* pane to display its properties in the *Lower Properties* pane.

The *Properties* panes contain up to five tabs:

- Screen
- Manager
- Offset/Size (for floating Representation Definitions)
- Style
- Format

The five tabs represent the parts of a Subapplication that are relevant to the selected component.

Screen Tab	The *Screen* tab displays the Pattern Definition that the selected component is attached to.
Manager Tab	The *Manager* tab contains information regarding the data flow between the screen and the window, and the window and the screen. This information is relevant to the selected component and the Pattern Definition it is attached to.
Offset/Size	The *Offset/Size* tab defines the offset and/or size for each component in a complex floating Representation Definition.
Style Tab	The *Style* tab specifies functional and presentation aspects of properties intrinsic to the selected component.
Format Tab	The *Format* tab specifies presentation aspects of information taken from the host screen.

The Screen Tab

The *Screen* tab displays a graphic presentation of the structure of the Pattern Definition that the selected Representation Definition is attached to.

Pattern Definition Tree

In the Pattern Definition tree diagram each Pattern Definition is represented by a rectangle. The rectangle contains the Pattern Definition name.

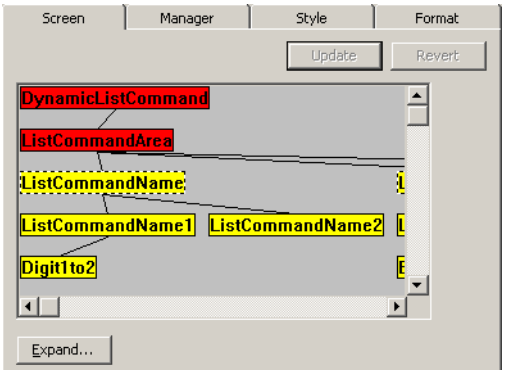


Figure 87. Pattern definition tree in the Screen tab

The currently selected Pattern Definition is at the top of the diagram. Child patterns of this Pattern Definition appear in their hierarchical position.

The diagram indicates which part of the Pattern Definition is represented by the selected component and displays a component's position within the structure of the Pattern Definition.

Expanding the Tree

Click the *Expand* button to view the diagram in its full size.

The diagram displays the whole tree structure of a Pattern Definition. It is often too large to be viewed all at once. Use the scroll bars to scroll through the diagram.

An expanded tree in the Screen tab is displayed as follows:

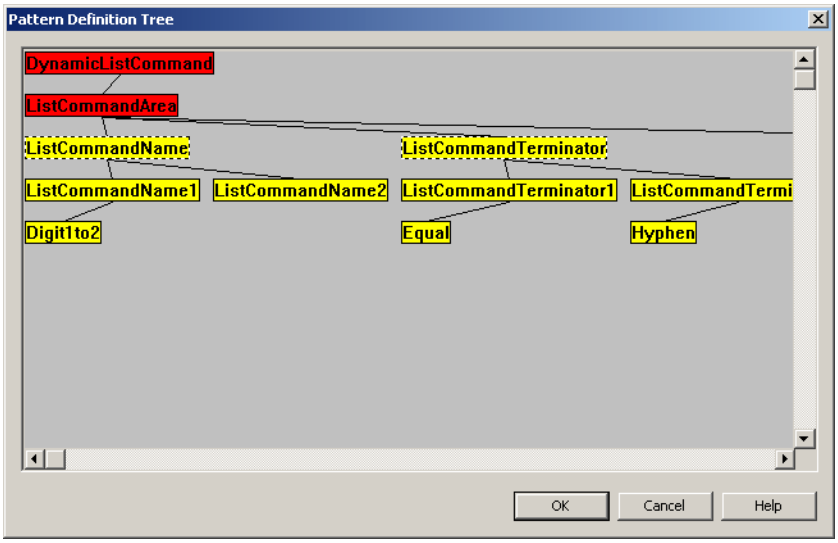


Figure 88. Expanded pattern definition tree in the Screen tab

In the diagram above, the Pattern Definition is *DynamicListCommand*. The part of this Pattern Definition represented by the selected component is the child pattern *ListCommandArea*.

From the diagram above we can see the following:

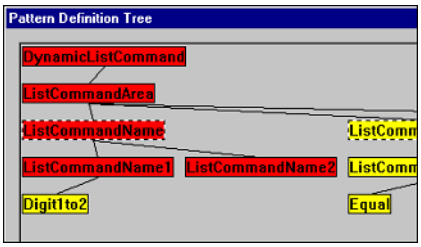
- The parent Pattern Definition name at the top of the tree is colored red (*DynamicListcommand*).
- The Pattern Definition that is represented by the selected component is colored *red* (*ListCommandArea*).
- All other Pattern Definitions are colored *yellow*. More generally, the direct hierarchy of Pattern Definitions between the child represented by the control and the parent pattern is colored red.

In most cases, only one Pattern Definition per level can be colored red, except when the Pattern Definition is a One Of type.

In the Pattern Definition tree, a One Of type Pattern Definition is indicated by a dashed border. When a One Of type definition is colored red, all its child patterns are also colored red.

Example 39. Expanding the tree

The following figure shows a section of a Pattern Definition tree in which the *ListCommandName* Pattern Definition is selected. Characters recognized by this Pattern Definition will be represented by the control.



From the diagram above we can see the following:

- The Pattern Definitions on the path from the top Pattern Definition to the child pattern that is represented by the Representation Component are all colored *red*.
- The *ListCommandName* Pattern Definition is a One Of type definition and is designated by a dashed border. Both its child patterns, *ListCommandName1* and *ListCommandName2*, are colored red.

Screen Options

On the Screen tab, you can view, specify, or change the Pattern Definition represented by the current component.

To view or edit the Pattern Definition represented by a component:

1. Open the *KnowledgeBase Definitions* window in *Representation Definition View* and select a Representation component.
2. In the *Properties* pane *Screen* tab click the *Expand* button.
   A *Pattern Definition Tree* dialog box opens with an expanded diagram, where at least the top Pattern Definition is colored *red*.
   Use the scroll bars to view the different parts of the diagram.

3 When specifying a Pattern Definition for a new component or changing the Pattern Definition represented by an existing component, the following rules apply:
   - Clicking a *yellow* definition selects it and colors it *red*. All the definitions on the path between the selected definition and the top Pattern Definition are also colored *red*. If other definitions were already selected, the new selection cancels the old one.
   - You can only select one Pattern Definition at each level. However, when you select a One Of type definition (marked by a dashed border) all its child patterns are automatically colored *red*. Clicking a child pattern of a One Of type definition has no result. You can only select a Pattern Definition above or below the child patterns.
   - Clicking a *red* Pattern Definition deselects it and changes it back to *yellow* together with all the selected definitions below it (but not above it). However, clicking the child pattern of a One Of type definition has no result. You have to unselect a Pattern Definition of a higher level in order to unselect the One Of child patterns.

4 Click *OK* to accept your changes and return to the *Properties* pane. If you do not want to accept your changes, or if you have only been viewing the diagram, use the *Cancel* button to return to the *Properties* pane.

Note: You can also work on the *Screen* tab without expanding the diagram. In this case click *Update* to accept changes and *Revert* to cancel changes.

The Manager Tab

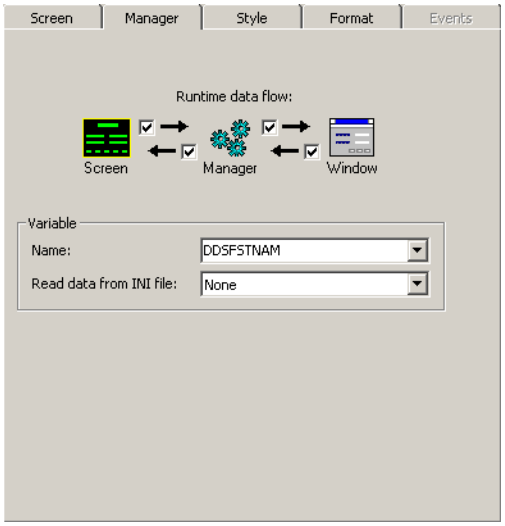


Figure 89. Manager tab

The *Manager* tab handles the interaction between the Screen and the Window. It displays a graphic representation of the parts of the Subapplication Manager that are relevant to the selected component.

The Manager contains the following elements:

- Runtime Data Flow
- Variable Name
- Variable external data source

Runtime Data Flow

The Manager supervises the interaction between the Screen and the Window during runtime. During Runtime, information may be transferred in four directions:

- From the Screen to the Manager
- From the Manager to the Screen
- From the Manager to the Window
- From the Window to the Manager

The Manager section has four check boxes, each of which is embedded in an arrow corresponding to one of the four data flow directions. When a check box is set, this means that information is transferred in that direction during runtime.

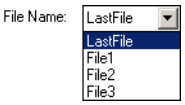
Example 40. Runtime data flow

A screen has a character sequence of the following kind:

```
FileName ===> LastFile
```

This sequence prompts the user to select a file to work on. As a default value, the input field already contains the name of the last file worked on by the user. If this is not the desired file, the user can type the name of a different file.

This pattern could be represented in the window in the following manner:



In this example, information is transferred in all four directions:

- From the Screen to the Manager: the default value "LastFile".
- From the Manager to the Window: the default value "LastFile".
- From the Window to the Manager: the name of the selected file.
- From the Manager to the Screen: the name of the selected file.

A screen has the following output field:

```
Name: Software GmbH
```

This field displays the company name, which changes, and must therefore be updated as follows:

- From Screen to Manager
- From Manager to Window

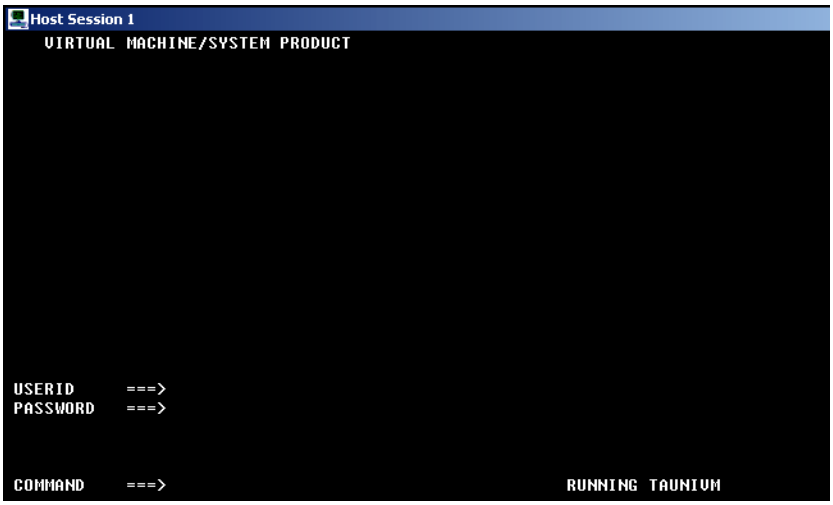
There is no need to update in the opposite direction (Window --> Manager--> Screen) because this is an output field that the user cannot change.

- For variable type representations there is no control representation. Therefore, the directions set are:

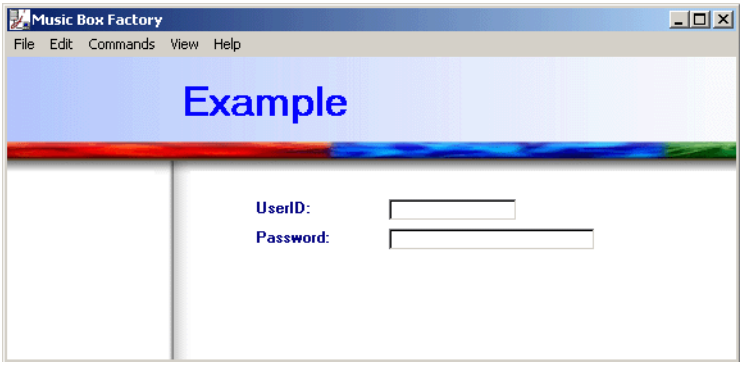
  Screen-->Manager, Manager-->Screen.
- A locally added control is not attached to a host screen character sequence. Therefore, the directions set are:

  Manager-->Window, Window -->Manager

The following example of a logon Screen includes three character sequences: `USERID`, `PASSWORD`, and `COMMAND`:



However, the Windows presentation of this screen does not contain a representation for the COMMAND sequence:



The COMMAND pattern does not need to be represented in the window because a push button or menu item can fulfill the COMMAND sequence's role.

In such a case, information related to the COMMAND sequence is transferred only in two directions:

- From the Screen to the Manager.
- From the Manager to the Screen.

There is no interaction between the Manager and the Window.

Variable Names

A Variable name uniquely identifies a field on the screen. A Representation Definition, in addition to creating the specified control, also creates a variable name for the part of the host screen character sequence represented by the component.

Automatic Variable Name

Each component of a Representation Definition that has runtime data flow from the Screen to the Manager creates an automatic variable name for the character sequence represented by the component.

The automatic name has the following structure:

SubRRCCCHHWWW

"Sub"	A fixed string.
	The rest of the name indicates the character sequence's location and dimension in the screen:
RR	The row number at which the sequence begins (2 digits).
CCC	The column number at which the sequence begins (3 digits).
HH	The height of the sequence in rows (2 digits).
WWW	The width of the sequence in columns (3 digits).

For example, Sub1901501008 is the name of a sequence that begins at row 19 and at column 015, is 01 rows high and 008 columns wide.

When a Representation Definition has more than one component, each component usually refers to a different part of the represented character sequence, so that the variable names are different. There are cases, however, when more than one component of a Representation Definition refers to the same part of the sequence.

This situation occurs, for example, when a Menu Item component represents the same sequence that is represented by a control. In such cases, in order to avoid creating identical Variable names for different components, the second automatic name begins with the string *S2_* (instead of Sub), the third with *S3_*, etc.

For example, the second Variable Name in the above pattern would be S2_1901501008.

Explicit Variable Name

You can change the "Sub..." name of a component to an explicit name. However, the option of changing a name should be used with caution, and only for components associated with a Pattern Definition that is satisfied by only one character sequence per screen.

In contrast to the automatic "Sub..." name, which serves as a unique identifier, when you assign an explicit name, this name will be used in each representation of the Pattern Definition. Therefore, do not change the automatic “sub” name of Representation Definitions of Pattern Definitions that are satisfied by more than one character sequence per screen. If you do, you will not be able to differentiate between instances of the same Pattern Definition.

Example 41. Explicit variable name

InputField is a general Pattern Definition that can match many different sequences on one screen.

The automatic "Sub..." name given to each *InputField* on the screen is determined by its location and dimension in the screen and is therefore unique.

However, if you replace the automatic name with an explicit name, such as InputField for example, all the different input sequences on a single screen will have the same name, and you will not be able to differentiate between them.

To avoid this problem, an explicit name should be given only to sequences which usually appear only once in a screen.

Note: The variable name should never be empty or a duplicate name. These will be rejected by the KnowledgeBase.

Read Data From INI File

In the Manager section you can define whether data is to be imported, and where it is to be imported from. When no data is to be imported, select *None*.

When data is to be imported, it can be imported from one of three INI files:

General INI file	This INI file contains data common to all Applications.
Application INI file	This INI file contains data that is specific to an Application.
Subapplication INI file	This INI file contains data that is specific to a Subapplication.

The Style Tab

Each representation component has two kinds of window presentation properties:

- Some presentation properties are intrinsic to a component's type.
- Some presentation properties include external information, including the formatting of the character sequence that the component represents.

You specify a component's intrinsic representation properties in the *Style* tab. Each type of component has a different *Style* tab.

Modifying Component Styles

To modify a component's style, select the component and then select the *Style* tab in the *Properties* pane

On each *Style* tab there are control elements you use to modify the style. Two of these elements— *Font & Color* and *Validity check*—are common to several controls. Since they require a detailed discussion, they are discussed in separate sections. To learn about fonts and colors, refer to Chapter 11 - "Color and Font Design" on page 307. Validity Checks are discussed in the following section.

When you have configured a style tab to your liking you can:

- Save changes you made by clicking the tab's *Update* button.
- Undo changes made since the last update by clicking the tab's *Revert* button.

Note: Some control types are not represented in the Window and therefore have no style settings. The *Style* tab for these controls is disabled.

Validity Checks

Host input fields often restrict the values they will accept. Application performance can be increased by including these restrictions in the GUI client, thereby avoiding a host-GUI communication short circuit when the user inputs invalid values. In ACE, these restrictions are called *validity checks*. Validity checks are attached to a control as part of the control's style. Set a validity check for a control when the underlying host field has restrictions on the input values it accepts.

Internal Validity Checks

For certain controls, ACE provides a set of validity checks as part of the supplied KnowledgeBase.

None	No validity check is performed.
NotEmpty	Verifies that the input field is not empty.
NotEmptyNorBlanks	Verifies that the input field is not empty and does not contain blanks.
OnlyDigitsAndNotEmpty	Verifies that the input field is not empty and contains only digits.
OnlyDigitsOrEmpty	Verifies that the input field contains only digits or is empty.
ValidateName	Verifies that the first typed character is an upper case letter or one of $,#,@, and that all other characters are either uppercase letters, digits, or one of $,_,#,@. These are AS/400 specifications.
ValidateNameAndBlanks	Same as *ValidateName,* but the field may also contain all blanks and nothing but blanks.
ValidateNameExtended	When the typed characters are enclosed by double quotes, then any character is allowed between the double quotes except for blank, single quote, double quote, *, and ? When the typed characters are *not* enclosed by double quotes, then the first typed character must be a letter or one of $,#,@ and all other characters must be either letters, or one of $,_,#,@. These are AS/400 specifications.
ValidateNameExtendedAndBlanks	Same as `ValidateNameExtended` but the field may also contain all blanks and nothing but blanks.

CompareValue

The typed characters are allowed when they are in a specified ordinal relation with a specified test value.

The defined test values are `*blanks` and `*zeroes`. Use *blanks to specify an ordinal relationship between alphabetic input and the blank character. Use *zeros to specify an ordinal relationship between numeric input and the value "zero".

To set a CompareValues validity check

1. In the control's *Style* tab select *CompareValue* from the *Validity Check* combo box. The *Set Parameters* button is enabled.
2. Click the *Set Parameters* button. The *CompareValue Parameters* dialog box opens:

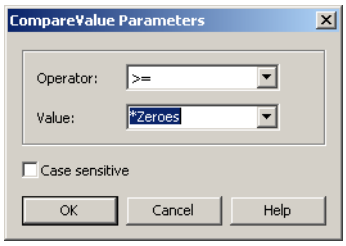


Figure 90. CompareValue Parameters dialog box

3. From the *Value* combo box select a test value.

   For *blanks set the *Case Sensitive* check box to differentiate between upper case and lower case letters.
4. From the *Operator* combo box select the relation the input must meet relative to the test value in order for the input to be valid. The operators are:

==	The input must equal the test value.
!=	The input must not equal the test value.
<	The input must be strictly less than the test value.
>	The input must be strictly greater than the test value.
>=	The input must be greater than or equal to the test value.
<=	The input must be less than or equal to the test value.

5 Click *OK*. The *CompareValue Parameters* dialog box closes. The selected validity check appears in the combo box:

In this example the input field will accept “0” and “13” but reject “-7”.

External Validity Checks

When the supplied validity checks do not match the host’s input restrictions, ACE can perform validity checks defined in an external DLL file. When developed and configured properly, the validity checks can be configured in ACE and the validity checks are performed in runtime.

ACE is supplied with a demonstration DLL called VALIDCHK.DLL. To use the validity checks defined in this DLL, put

```
[ValidityCheck]
DLLs=validchk.dll
```

into the CONVERTER.INI file so that the validity checks may be applied to controls during conversion and into the <Application>.INI file so that the validity checks are applied during runtime.

Validity checks can be implemented from more than one DLL file. In this case the INI file entry in the `ValidityCheck` section should be

```
DLLs=name1.DLL;name2.DLL;name3.DLL.
```

> **Note:** These DLL files must be in the same directory as all the other ACE DLL files.

VALIDCHK.DLL contains the following validity checks:

EXT_AllNumbers	Verifies that the input field contains only digits.
EXT_FieldIsFilled	Verifies that the input field is completely filled.
EXT_FloatOrInt	Verifies that the input field contains only digits or digits and a period arranged as a decimal number.
EXT_NoEmbeddedBlanks	Verifies that the field is either empty or contains exactly one word.

EXT_SignedNumber	Accepts only numbers, but can have a leading plus or minus sign.

Control Types and Styles

Following is a list of control types. The next sections describe each control type together with a description of the options available in each control's *Style* tab.

- Accelerator
- Button
- Check Box
- Combo Box
- Date
- Dynamic Group
- Frame
- General U.T. Method
- Group Box
- HostBasedFormatValue
- Line
- MenuItem
- Prompt
- RadioGroup
- Spin
- Static
- SubWindow
- Table
- Tabs
- TextBox
- Variable
- Window

Accelerator

Accelerators allow the user to perform a command from the keyboard using one or more keys. Menu items, for example, can have an accelerator key defined which performs the option using a key or a key combination. The accelerator is indicated by the underlined letter in the option, or the accelerator keys are expressed in written form next to the item.

The Accelerator *Style* tab has three sections:

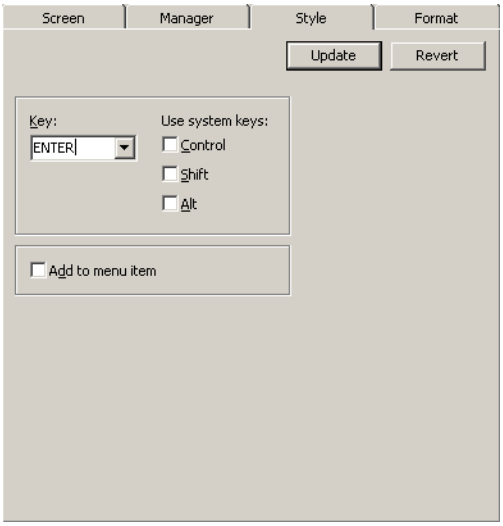


Figure 91. Accelerator Style tab

Key

The *Key* section offers a list of buttons which may be pressed to activate a command:

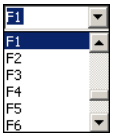


Figure 92. Key section

Use System Keys

The *Use system keys* section offers three system keys which may be used to form a command in conjunction with a key chosen from the *Key* section. More than one system key may be selected.

Add to Menu Item

The *Add to menu item* section allows you to add the accelerator to a menu item.

Button

Buttons have the following characteristics:

- Buttons can display both text and graphics. The user can define the position of the text and the graphics, and the relative position of both.

- Buttons can have up to 4 associated image files for the different button states: standard, pressed, in focus, and disabled. Currently only the standard and pressed states are supported.
- Buttons can be configured to reject focus.
- WAV files (sound) can be attached to the button.
- Balloon help can be attached to the button.
- A default button can be defined.

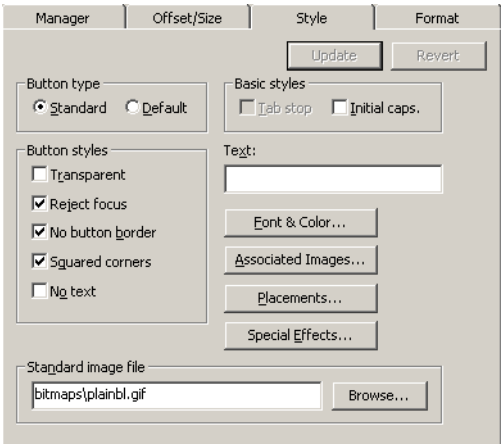


Figure 93. Button Style tab

Button Type

Standard Pressing the *Enter* key will not execute this button.

Default Pressing the *Enter* key executes this button. Only one button on a screen can be the default button.

Basic Styles

Tab stop Enables the button to be accessed by tabbing.

Initial caps When set, text from the host is displayed with initial capital letters.

Button Styles

Transparent The button is transparent. It will assume the color of the window background.

Reject focus	The button cannot receive focus, including after it has been released. When the button is configured to reject focus, the tab stop check box is disabled.
No button border	The button has no border. Only the bitmaps and the text are displayed.
Squared Corners	The button's corners are squared.
No Text	Host screen text is not displayed.
Text	The text to be displayed on the button. An ampersand, "&", makes the following character into an accelerator. This accelerator character appears underlined.
Font and Color	Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.
Associated Images	Other image files to be used for the different button states. ACE enables three additional states: focused, selected, disabled.
Placements	Arrange the position of the text in relation to the graphic.
Special Effects	Attach a WAV sound file or balloon help text to a button

Associated Images

ACE enables you to define a series of images that indicate different states of the button. One image is used for the button when it is displayed in its standard position, and other images are configured for each of the following states: selected, in focus and disabled.

Click the *Associated Images* button to configure the images to be used for each state.

Note: The use of images for additional states is optional.

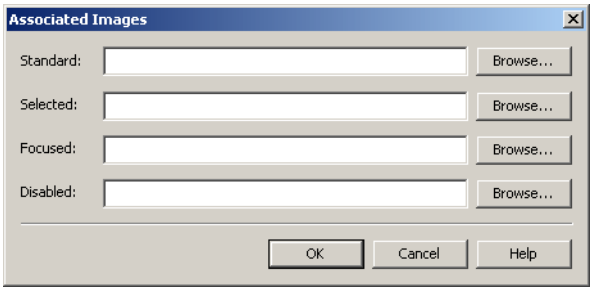


Figure 94. Button associated images

Placements

Click the *Placements* button to set the placement of the text and the graphic on the button. The following dialog box is displayed:

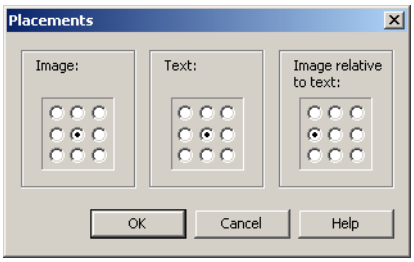


Figure 95. Button placement relative to text

Image	The position of the image on the button.
Text	The position of the text on the button.
Image Relative to Text	Enabled when the image and text are in the same position. Places the image around the text according to this setting.

Special Effects

Click the *Special Effects* button to attach WAV sound files or balloon help to a button. The WAV sound files must be on your disk. To hear the sound you must either have a sound blaster or sound simulation software installed in your system.

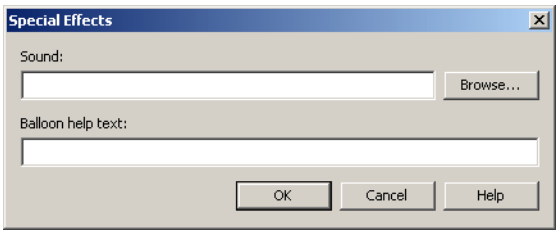


Figure 96. Button special effects

Sound	Add a WAV type sound file that plays when the user clicks on the button.
Balloon Help Text	Enter the text that is displayed when the mouse cursor is pointing at the button.

Example 42. Special effects

You can add sound and balloon help to a button that has other functionality. For example, you can configure your *Cancel* button to display a short textual message when the cursor is pointed at the button.

You can also configure the button to play a short sound sequence (such as chimes) whenever it is pressed:

1. Access the Pattern Definition associated with the *Cancel* button.
2. Modify the representation in the *Representation Component* dialog box and define its style as *Button*.
3. From the *Button Style* tab in the *Lower Properties* pane, click *Special Effects.*

   The *Special Effects* dialog box opens.
4. In the *Sound* edit box click *Browse* to find the WAV file for the sound sequence desired, or enter the text to appear when the cursor is pointed at the button in the *Balloon help text* edit box.

In runtime, when you point the cursor at he button, the balloon help is displayed. When you click the *Cancel* key, the sound sequence is heard while Cancel is executing.

Note: WAV files are not part of the ACE resource. Therefore they are not compiled in the ACE MakeExe. Be sure to copy the WAV files that you have configured in this function, into your Runtime directory.

Check Box

A check box is a toggle for switching an option on and off. A small square appears adjacent (usually to the left) of the option description. The check box can be set and cleared with the mouse.

Clicking a blank check box marks it with a check sign and sets the option it refers to. Clicking it again clears the check box and the option is not set.

A check box can appear alone, or in a group of more than one check box. When a group contains several check boxes, you can simultaneously set as many or as few check box options as required.

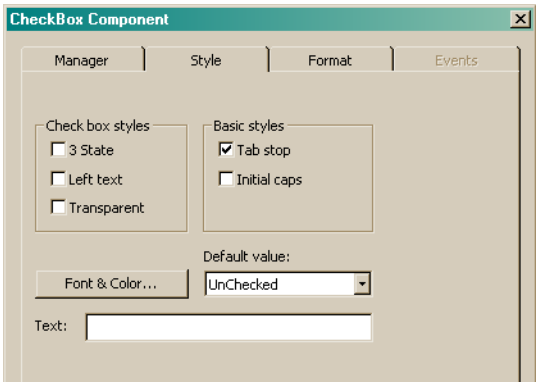


Figure 97. CheckBox Style tab

Check Box Styles

3 State In addition to the two states of every check box, set and cleared, a 3-state check box has also a grayed state. The grayed state is typically used to indicate a disabled check box.

Left text Places the prompt text on the left side of the check box.

Transparent Makes the background of the checkbox control transparent. This permits the layering of controls. For example, you could place the checkbox on top of a button that has an associated image.

Basic Styles

Tab stop Allows you to jump to this control using the *Tab* key. When set, the *Text* property should have a value, even if this value consists of blanks typed in the *Text* field. Otherwise, when the checkbox has the focus in runtime, there is no indication that the checkbox indeed has the focus.

Initial caps When this check box is set, text which comes from the host has initial capital letters.

Font & Color Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

Default value This option allows you to choose the default state of the check box during runtime.

Text This is the descriptive text that appears next to the checkbox. The text can have three different sources:

- The Application's INI file. In Design View, you can specify information to be saved to the Application's INI file. For more information, see "The Manager Tab" on page 215.
- Text typed in the *Text* edit field. This can even be blanks. See the description of the checkbox Tab stop setting above and *Tips About Checkboxes* below for when to do this.
- Text that appears in the host.

Tips About Checkboxes

When a checkbox has the focus, a dotted line appears around the characters associated with the *Text* property of the checkbox:

If there are no characters associated with the *Text* property, the dotted line surrounds an area of zero width, and is thus not visible.

The *Text* property could be empty because you wish to insert space between the descriptive text and the actual checkbox, and then align both the text and the checkbox with other elements on the window. You cannot easily do this when the descriptive text and the checkbox are part of the same graphical element. To have both separation and alignment you need to clear the check box's *Text* property and create a new static control containing the descriptive text.

However, as the checkbox *Text* property is then empty, the checkbox will not indicate that it has the focus. Typing a few blanks, the spacebar character, in the *Text* property gives the dotted focus line something to surround.

Combo Box

A combo box control is composed of a selection field, similar to an edit field control, together with a list box control. The list may be displayed at all times or may have an arrow icon that drops the list down when it is clicked with the mouse. A combo box may be editable or non-editable.

When the list is visible, typing characters in the selection box highlights the first list entry which matches the typed characters. In addition, selecting an item in the list box displays the selected text in the selection field.

If there are more choices than can fit in a list box, scroll bars are provided.

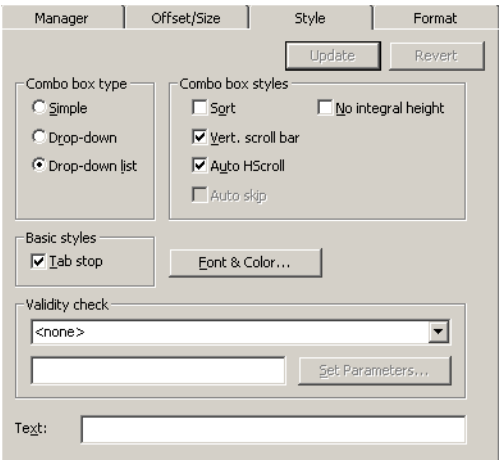


Figure 98. ListBox control Style tab

Combo Box Type

Simple	Causes the list box to always be visible. The current selection in the list box is displayed in the selection field. A string can be selected from the list or text can be typed in the selection field.
Drop-down	Similar to a simple combo box style, except that the list drops down and opens only when the arrow icon next to the selection field is clicked.
Drop-down list	Similar to the drop-down combo box style, except that text cannot be entered. You can only select a string from the drop-down list.

Combo Box Styles

Sort	Sorts the combo box strings alphabetically.
Vert. scroll bar	Places a vertical scroll bar in a combo box when there are too many items to fit into the combo box.
Auto HScroll	Scrolls when the text reaches the end of a line in the edit field. When cleared, the text is limited by the boundaries of the edit control.
Auto skip	When set, if the edit box is full and the cursor is on the last position in the field, ACE skips to the next field on the window. Auto skip is enabled only if the combo box type is Simple or Drop-down. If the combo box type is Drop-down list, the Auto skip check box is disabled, since the user can not type in the field. The Auto skip check box is cleared by default.
No integral height	Creates a combo box that does not adjust to display an integral number of lines.

Basic Styles

Tab stop Allows you to jump to this control using the *Tab* key.

Validity check For information on the available options, see “Validity Checks” on page 221.

Font & Color Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

Text This edit field contains an ordered list of all the items that appear in the Combo box. The items are delimited by semicolons. Note that if the list has a default item, then that item appears first in the list. For example: B; A; B; C produces a combo box that looks like this:

If there is no default item, then a semicolon appears first in the list. For example: ; A; B; C produces a combo box that looks like this:

Date Field

The Date control is a specialized edit field that can provide a standard user interface for all the dates in an Application. The control allows the user to type in a date manually, enabling separator characters to be skipped. It also performs automatic validation while typing.

This feature enables you to configure the date using the short date format as defined in the *Windows Control Panel*.

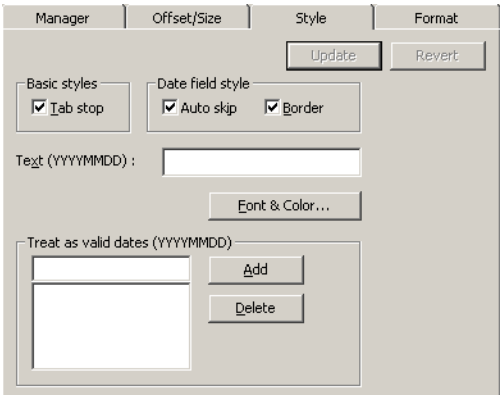


Figure 99. Date control Style tab

Basic Styles

Tab stop Allows you to jump to this control using the *Tab* key

Date Field Styles

Auto Skip When *Auto skip* is set, ACE skips to the next control on the window, once the cursor reaches the end of the date.

Border Draws a border around the control. This option is cleared by default.

Font and Color Information on the available options can be found in the section *Color and Font Designing*

Text (YYYYMMDD)	The default text is represented in canonical (internal) format (YYYYMMDD): The year is four digits; The month is two digits; The day is two digits.
Treat as valid dates (YYYYMMDD)	Forces the date control to accept dates that are in the correct format (YYYYMMDD) but are not true dates, e.g. 19861453. To add a date to the list, type it in the edit box and click *Add*. To remove a date from the list, select the date and click *Delete*.

If an entry is given which is incorrect, for example 13 for the number of months, the following message appears:

Dynamic Group

Dynamic group controls are not actually controls, but algorithms for ordering controls. When a dynamic Pattern Definition is represented by a control, there may be more than one instance of that control generated in runtime, according to how many character sequences satisfying the dynamic Pattern Definition that appears on the runtime screen.

The dynamic group control is used to specify the window arrangement of multiple instances of a control generated from a single dynamic Pattern Definition.

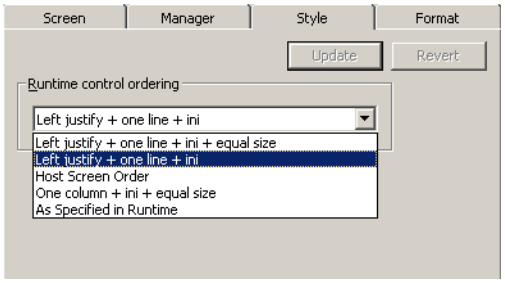


Figure 100. Runtime control ordering section

Runtime Control Ordering

Left justify + one line + ini +equal size	Controls are placed in a single line. The controls are resized to be the same width as the widest control. The key `DynamicControlsOrder` in the `[<subapplication_name>]` group of the <applname>.INI file can define a preferred left-to-right order.
Left justify + one line +ini	Controls are placed in a single line. The key `DynamicControlsOrder` in the `[<subapplication_name>]` group of the <applname>.INI file can define a preferred left-to-right order.
Host Screen Order	The controls are arranged according to the arrangement of the corresponding character sequences on the host screen.
One column + ini + equal size	Controls are placed in a single column. The controls are resized to be the same width as the widest control. The key `DynamicControlsOrder` in the `[<subapplication_name>]` group of the <applname>.INI file can define a preferred top-to-bottom order.
As specified in Runtime	The controls are ordered according to the option specified in the *Dynamic Controls* tab of the runtime environment's *Options* > *General* dialog box.

Frame

The Frame control appears as follows on the GUI window.

Figure 101. Frame control

The Frame *Style* settings are:

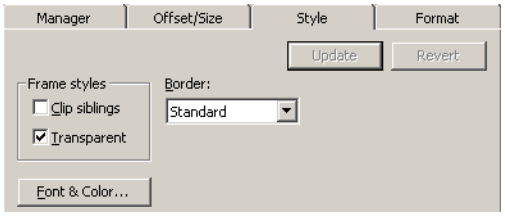


Figure 102. Frame control Style tab

Frame styles

Clip Siblings: Places the control in the foreground when set, or in the background when cleared.

Transparent: See-through or opaque.

Note that non-transparent frames can be displayed with background color. If another control is included within the frame, the frame must be defined as *Clip Siblings.*

Border

Presentation of the frame. This can be one of five:

- Elevated
- Indented
- Raised
- None
- Standard

Font and Color

Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

GeneralUTMethod

The GeneralUTMethod (General User-Triggered Method) control gives functionality to the other controls of the Representation Definition, when necessary. See Chapter 13 - "Methods: Attaching Functionality to the GUI Window" on page 335 for more information.

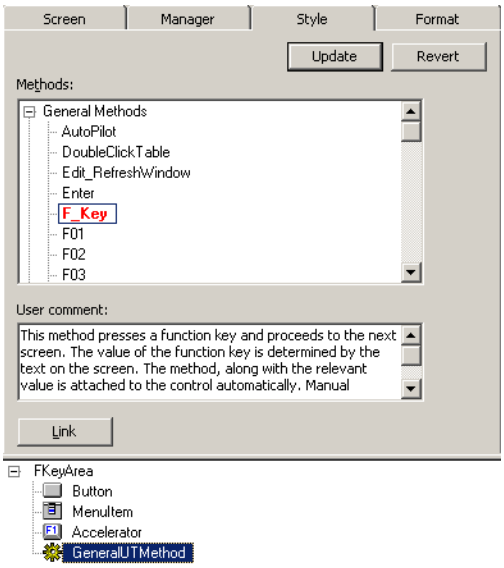


Figure 103. Methods list in Style tab

Select the method to attach to the Representation Definition from the *Methods* list and press *Link*. A description of the method appears in the *User comment* box.

Group Box

The Group Box control is a frame where like controls may be placed. Grouping controls can serve two purposes: a logical purpose and a design purpose.

Controls can be grouped together logically, such as a set of radio buttons or a set of check boxes. In such a case, you move among the grouped controls using the arrow keys. Controls can also be grouped together to enhance the design and the clarity of the dialog box. In this case you can move between the controls using the *Tab* key.

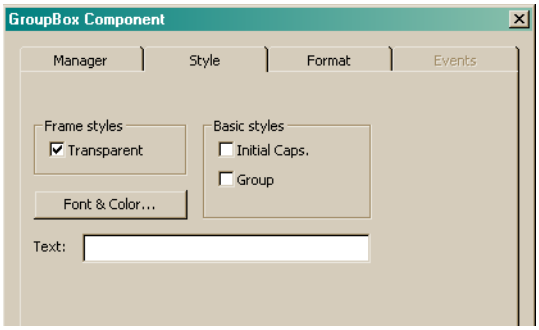


Figure 104. Group Box control Style tab

Frame Styles

Transparent When selected, makes the background of the Group Box control transparent. This permits the layering of controls. For example, you could place the Group Box on top of a button that has an image associated with it.

Additional details of the Group Box's appearance and behavior can be set in the **Basic Styles** panel.

Basic Styles

Initial Caps When set, text from the host has initial capital letters.

Group When set, the arrow keys cycle through the controls in the group.

Font and Color Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

Text The text that appears at the head of the group box. The text can have three different sources:

The application INI file. For more information, see "Read Data From INI File" on page 220.

Text typed in the *Text* edit field.

Text that appears in the host.

HostBasedFormatValues

The HostBasedFormatValues *Style* tab is a collection of five independent component styles. These styles, although independent, are all related to the Representation Definition *ComboBox*.

Combo boxes are typically attached to Pattern Definitions that include the Pattern Definition *EntryPair*, or the Pattern Definition *Entry*, as a child pattern. The various HostBasedFormatValues styles are attached to *EntryPair* and its child patterns, or to *Entry*.

Pattern Definitions Including EntryPair

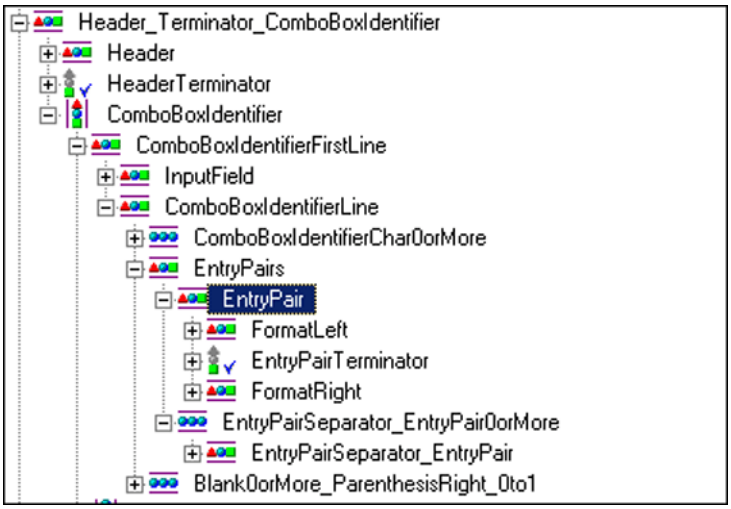


Figure 105. Pattern definition including EntryPair definition

The Pattern Definition *Header_Terminator_ComboBoxIdentifier* has a child pattern *ComboBoxIdentifier*. *ComboBoxIdentifier* has a *ComboBox* representation component attached to it.

ComboBoxIdentifier contains the child pattern *EntryPair*, which in turn contains the child patterns *FormatLeft* and *FormatRight*.

- *FormatLeft* is satisfied by a host character sequence corresponding to a value to be entered into a host screen field.
- *FormatRight* is satisfied by a sequence corresponding to the meaning of a value to be entered into a host screen field.
- *EntryPair* is satisfied by the combined sequence that informs a viewer that the value on the left has the meaning shown on the right.

Pattern Definitions Including Entry

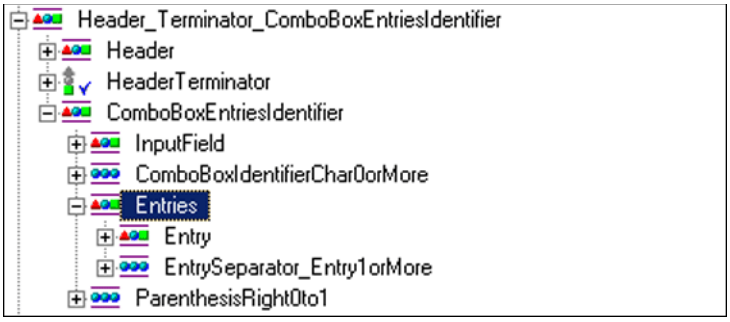


Figure 106. Pattern definition including Entry definition

The Pattern Definition *Header_Terminator_ComboBoxEntriesIdentifier* has a child pattern *ComboBoxEntriesIdentifier*. *ComboBoxEntriesIdentifier* has a *ComboBox* Representation Component attached to it.

ComboBoxEntriesIdentifier contains the child pattern *Entry*.

Entry is satisfied by a host character sequence corresponding to a value to be entered into a host screen field, and where there are no additional characters to describe the meaning of the value.

The Five HostBasedFormatValues Styles

The five different styles included on the HostBasedFormatValues *Style* tab are:

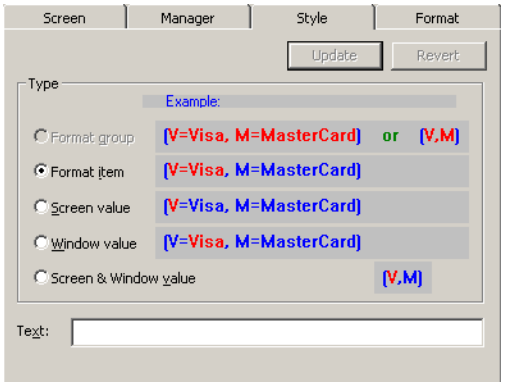


Figure 107. HostBasedFormatValues Style tab

Format group	A list of Format Items. (Not yet available).
Format item	One building block of the format. Each format item includes a value on the host screen paired with its description. *Format item* is attached to *EntryPair*.
Screen value	The value that goes into the field on the host screen. *Screen value* is attached to *FormatLeft*.
Window value	The value that is displayed in the control in the window. *Window value* is attached to *FormatRight*.
Screen & window value	In the case where there are no descriptions for the values, the value is placed in both the host field and the control. *Screen & window value* is attached to *Entry*.

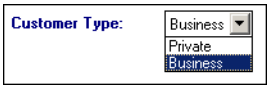
Note that you do not build a primary Pattern Definition that contains a HostBasedFormatValues representation component. The HostBasedFormatValues representation component must be attached to the lower-level Pattern Definitions that make up the primary Pattern Definition. Practically this means that the HostBaseFormatValues representation component must be placed under the ComboBox representation component in the pattern's hierarchy. The KnowledgeBase prevents you from building a primary Pattern Definition that has a HostBasedFormatValues representation component on the upper hierarchical level and warns you with an error message.

The following figure shows a cutout from a host screen, labeled to indicate the correct usage for three of the *HostBasedFormatValues* styles

Figure 108. Correct usage for three of the HostBasedFormatValues styles

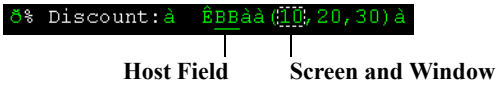
- Each of the character sequences `P=Private` and `B=Business` satisfy the Pattern Definition *EntryPair*. Each is represented by a *Format Item*.
- Each of the character sequences `P` and `B` satisfy the Pattern Definition *FormatLeft*. Each is represented by a *Screen Value*.
- Each of the character sequences Private and Business satisfy the Pattern Definition *FormatRight*. Each is represented by a *Window Value*.

This figure shows the cutout represented as a combo box.



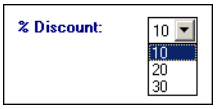
When you select *"Private"* its screen value **P** is placed in the relevant host field of the application.

This next example illustrates the fifth style that appears in the HostBasedFormatValues *Style* tab. In this case the value goes into both the host field and the control.



Each of the character sequences `10`, `20` and `30` satisfy the Pattern Definition *Entry*. Each is represented by a *Screen & Window value*.

This figure shows the cutout represented as a combo box:



When you select "`10`", its screen value, which is also "`10`", is placed in the relevant host field of the application.

Runtime Update of Controls with HostBasedFormatValues Components

Formats created using HostBasedFormatValues components can be updated during runtime according to text on the host. To do so, set the data flow checkboxes as below for each HostBasedFormatValues component.

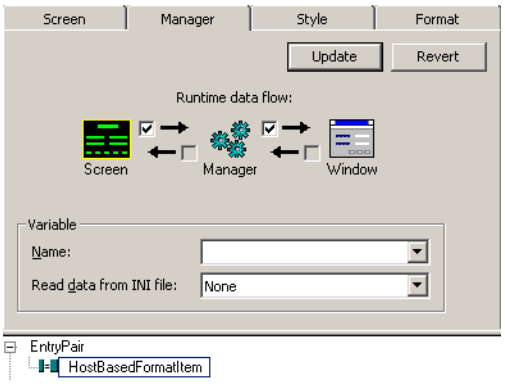


Figure 109. Updating formats created using HostBasedFormatValues

Line

The Line control appears as follows on the GUI window. The illustration shows two lines, each with a different border presentation.

Figure 110. Line control styles

The following parameters can be set through the Line component's *Style* tab.

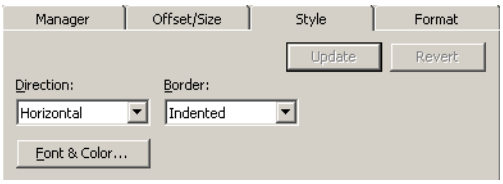


Figure 111. Line control Style tab

Direction	The line can be either horizontal or vertical.
Border	Presentation of the line. This can be one of five: • Elevated • Indented • Raised • None • Standard
Font and Color	Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

Link

The Link control provides a URL link to an IP address. During runtime, clicking a Link control sends you to another web page. The end-user is sent to a new location either in the same window or in a new window.

The Link control looks like this:

Contact Software AG

Figure 112. Link control

The Link control is set in the *Link* component's *Style* tab:

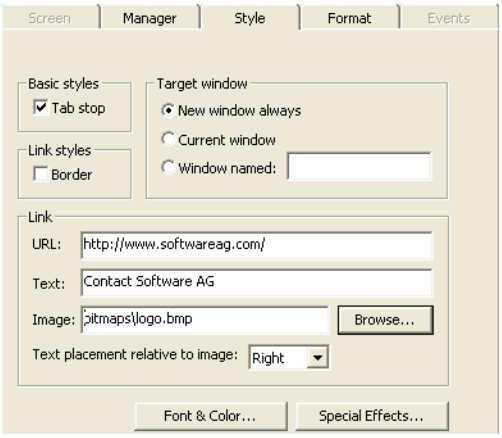


Figure 113. Link control Style tab

In the *Link Component* dialog box you can set the following:

Basic Styles

Tab stop Allows you to jump to this control using the *Tab* key.

Link Styles

Border Draws a border around the Link control.

Target window

New window always	The web page is always displayed in a new window.
Current window	The web page is displayed in the current window. The end-user can return to the runtime by pressing the browser's *Back* button.
Window named	The web page is displayed in a new window with a specified name. You specify this name in the edit box. This is useful if your Subapplication contains several Link controls and you want each one to open in a separate window with a different name.
Link	
URL	The URL of the new web page. Clicking the Link control during runtime sends the end-user to this destination.
Text	The text that appears in the Link control.
Image	Allows you to attach an image to this control (optional).
Text replacement relative to image	Enabled only when an image is attached. Here you define where the text is located relative to the image. Choose *Left*, *Right*, *Above* or *Below*.
Font & Color	Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.
Special Effects	Attach a WAV sound file or balloon help text to the Link control.

MenuItem

MenuItem is used for automatically adding menu items to the window menu.

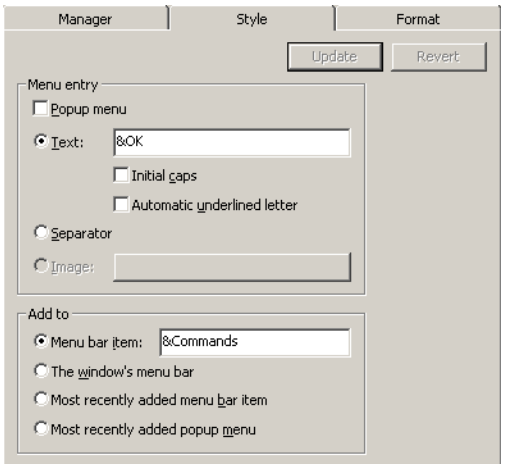


Figure 114. MenuItem component Style tab

Menu Entry

Popup menu	Makes the menu item prompt a popup menu.
Text	The text that appears in the menu item. This text has precedence over text taken from the screen, and is usually used in special cases to represent a particular item as a menu item.
Initial caps	This formats the text by converting words that were previously entirely uppercase, to words with initial capitals.
Automatic underlined letter	A letter of the menu item is underlined so that the menu item can be accessed via the keyboard.
Separator	Places a separating line before the menu item.

Add to

Menu bar item	Write the name of the menu bar item under which the menu item will be located.

The window's menu bar	The menu item will be added to the menu bar.
Most recently added menu bar item	The menu item will be added to the most recently added menu bar item.
Most recently added popup menu	The menu item will be added to the most recently added popup menu.

For more information on Menu components see Chapter 10 - "Editing Menus in Design View" on page 287.

Prompt

The Prompt control is used to display options that are prompted through the *F4* key (both on the AS/400 and mainframe). The Prompt control appears as follows on the GUI window.

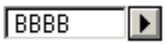


Figure 115. Prompt control

The Prompt control is a combination of an edit box and a push button. The edit box is configured through the styles dialog box. Functionality is added to the button by attaching an appropriate method to the control.

The following parameters can be set through the *Style* tab:

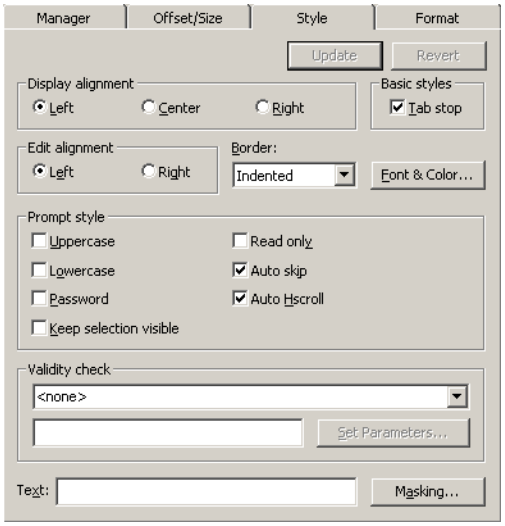


Figure 116. Prompt control Style tab

Display alignment	Choose the appropriate radio button to determine whether displayed text is left-aligned, centered or right-aligned.
Edit alignment	Choose the appropriate radio button to determine whether user-entered text is left-aligned or right-aligned.
Tab stop	Allows you to jump to this control using the *Tab* key.
Border	Draws a border around a control. There are four possible borders: *Elevated:* The edit box appears higher than the background. *Indented:* The edit box appears lower than the background. *None:* No border. *Standard:* The border is a line.
Prompt Style	
Uppercase	Converts characters to uppercase as they are typed.

Lowercase	Converts characters to lowercase as they are typed.
Password	Displays all characters as asterisks (*) as they are typed.
Keep selection visible	By default, an edit control does not appear selected when the control loses input focus and displays the selection with reversed colors when the control receives input focus. Setting this checkbox overrides the default.
Read only	The field may not be edited when this option is set.
Auto skip	When set, ACE skips to the next field on the window, if the edit is full and the cursor is on the last position in the field.
Auto HScroll	Automatically scrolls text ten characters to the right when a user types a character at the end of a line.

Radio Group

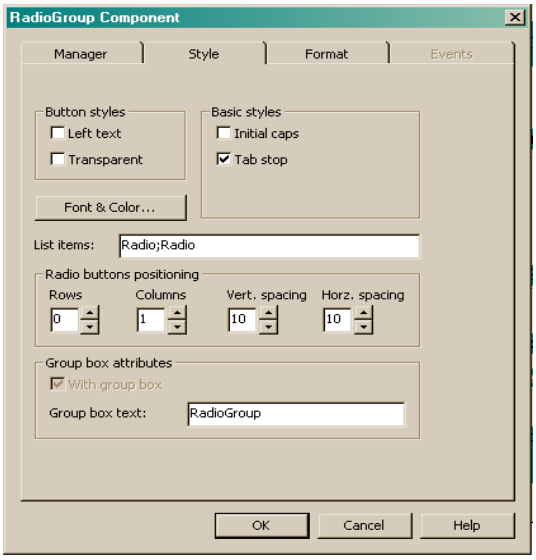


Figure 117. RadioGroup control Style tab

Button styles

Left Text Places text on the left side of the radio button.

Transparent Makes the background of the Radio Group control transparent. This permits the layering of controls. For example, you could place the Radio Group on top of a button which has an associated image.

Font and Color Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

Basic Styles

Initial Caps When selected, text from the host has initial capital letters.

Tab Stop Allows you to jump to this control using the *Tab* key.

List items The names of the buttons in the group are listed here, separated by semicolons. The first name, preceding the list, expresses the default radio button selected in the group. In the above screen, Red is the default selection. The ensuing titles are the headings which accompany the radio buttons. The resultant radio group as configured in the example is shown on the left.

Radio buttons positioning

Rows Determines the height of the radio group box.

Columns Determines the width of the radio group box.

Vert. spacing Determines the vertical positioning of the radio button within the group box.

Horz. spacing Determines the horizontal positioning of the radio button within the group box.

Group box attributes

With group box	The group box is part of the radio box, thus this option is set and disabled.
Group box text	Places text in the upper left corner of the group box.

Spin

The Spin control enables you to add spin boxes to your application. Spin boxes are text boxes that accept a limited set of ordered input numeric values. A spin box consists of a text box with a pair of arrows—an arrow pointing upward and an arrow pointing downward. The arrows are located on the right-hand side of the text box. The end user can type a new value into the text box manually, and/or increase or decrease the number by clicking the arrow keys.

A spin box appears as follows on the GUI window:

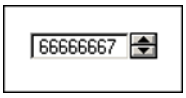


Figure 118. Spin control

The following parameters can be set in the *Style* tab:

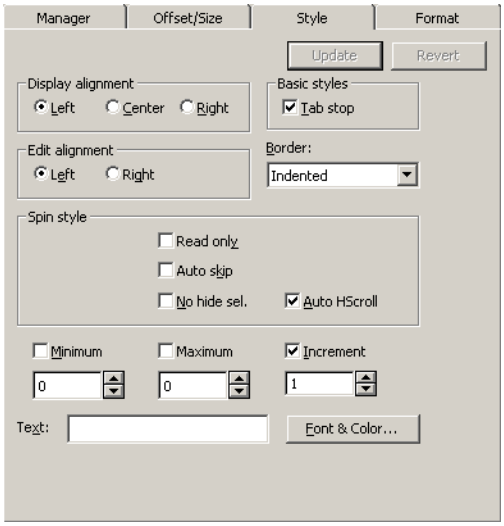


Figure 119. Spin control Style tab

Display alignment

Choose the appropriate radio button to determine whether displayed text is left-aligned, centered or right-aligned.

Edit alignment

Choose the appropriate radio button to determine whether user entered text is left-aligned or right-aligned.

Basic Styles

Tab Stop

Allows you to jump to this control using the *Tab* key.

Border

Draws a border around a control. There are four possible borders:

Elevated: The edit box appears higher than the background.

Indented: The edit box appears lower than the background.

None: No border.

Standard: The border is a line.

Spin Style

Read only

The field can not be edited when this option is set.

Auto skip

When set, ACE skips to the next field on the window if the edit is full and the cursor is on the last position in the field.

No hide sel

By default, an edit control does not appear selected when the control loses input focus and displays the selection with reversed colors when the control receives input focus. Setting this checkbox overrides the default.

Auto HScroll

Automatically scrolls text ten characters to the right when a user types a character at the end of a line.

Minimum Specifies the minimum number defined in the spin box. If cleared, the maximum number the end user can type is 2,147,483,647. If checked, you must specify the maximum number which can be used.

Maximum Specifies the maximum number defined in the spin box. If cleared, the maximum number the end user can type is 2,147,483,647. If checked, you must specify the minimum number which can be used

Increment Specifies the increment/decrement value whenever the user clicks the up/down arrows.

Note: There is a limit to the number of decimal characters which can appear on the window. The number is based on the decimal characters specified on the host.

Font and Color Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

Text This is the text that appears next to the spin box. The text can have three different sources:

- The Application INI file. In Design View, you can specify information to be read from the Application INI file. For more information, see “The Manager Tab” on page 215.
- Text typed in the *Text* edit field.
- Text that appears in the host.

Special Considerations for iSeries Host Applications

When you are converting an iSeries screen from a DDS file that has a spin control definition, the *minimum*, *maximum*, and *increment* values of the spin control as defined in the host screen definition are compared with the minimum, maximum, and increment defined for the spin control in the KnowledgeBase.

Based on this comparison, the values assigned to the spin control in the subapplication screen are as follows:

- The larger of the two minimum values specified becomes the minimum value of the spin control.

- The smaller of the two maximum values specified becomes the maximum value of the spin control.
- The incrementation value is always taken from the KnowledgeBase definition, if specified there. If it is not specified, the incrementation value defaults to 1.

Examples of automatic assignment of spin control values for subapplication screens created from a DDS host screen definition:

Table 11. Example of spin control values from DDS

Where Defined	minimum	maximum	increment
KnowledgeBase	undefined	undefined	5
DDS	COMP(GT 8)	undefined	undefined
Result	9	undefined	5
KnowledgeBase	10	60	undefined
DDS	undefined	COMP(LE 40)	undefined
Result	10	40	1
KnowledgeBase	20	60	undefined
DDS	RANGE (5 40)	RANGE (5 40)	undefined
Result	20	40	1

Static

A Static control contains text or graphics that are usually used to label another control such as a combo box, an edit field or a list box. A static control can also be used to create boxes and lines that separate one group of controls from another. Static controls have no input and provide no output.

The following parameters can be set on the *Style* tab:

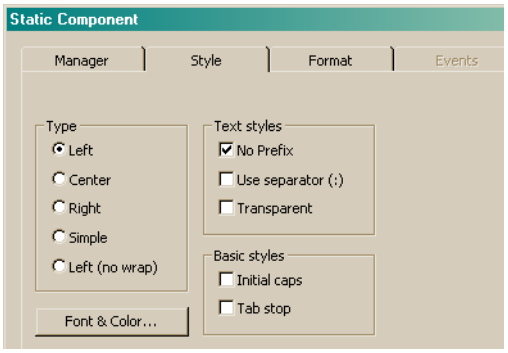


Figure 120. Static control Style tab

Type

Left	Displays text left-aligned in a simple rectangle. Text is formatted before it is displayed. Words extending past the end of a line are wrapped automatically to the beginning of the next line.
Center	Displays text centered in a simple rectangle. Text is formatted before it is displayed. Words extending past the end of a line are wrapped automatically to the beginning of the next line.
Right	Displays text right-aligned in a simple rectangle. Text is formatted before it is displayed. Words extending past the end of a line are wrapped automatically to the beginning of the next line.
Simple	Displays a single line of text left-aligned in a simple rectangle.
Left (no wrap)	Displays text left-aligned in a simple rectangle. Tabs are expanded, but words are not wrapped. Text extending past the end of a line is truncated.

Text Styles

No Prefix	Unless this style is specified, *Windows* interprets the ampersand character (&) in the control's text as an accelerator prefix command. Each “&” character is removed and the next letter underlined.
Use Separator (:)	Adds a colon (:) to the end of the static text.
Transparent	Makes the background of the Static control transparent. This permits the layering of controls. For example, you could place the Static control on top of a button which has an associated image.
Font and Color	Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.

Basic Styles

Initial Caps	When selected, text from the host has initial capital letters.
Tab Stop	Allows you to jump to this control using the *Tab* key.
Text	This is the text that appears next to the control. The text can have three different sources: The Application INI file. In Design View, you can specify information to be read from the Application INI file. For more information, see “The Manager Tab” on page 215. The second source is text typed in the *Text* edit field. The third source is the text that appears in the host.

SubWindow

For a discussion about SubWindows, see the chapter entitled “One-to-Many” in *JIS Interface Server: Advanced Topics*.

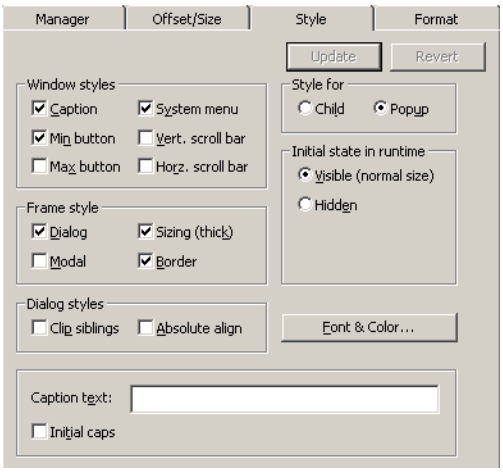


Figure 121. SubWindow control Style tab

Window styles

Caption	Displays a window with a title bar.
Min button	Creates a window with a minimize button.
Max button	Creates a window with a maximize button.
System menu	Creates a window with a system menu in its title bar.
Vert. scroll bar	Displays a window with a vertical scroll bar.
Horz. scroll bar	Displays a window with a horizontal scroll bar.

Style for

Child

Child windows are similar to a group box without a visible caption. Their advantage is that all of their controls are owned by the child window. If you hide or move the child window, all of its controls are hidden or moved as well.

Child windows can be dragged, using the arrow keys in the *Control Properties* Dialog Box. You can also drag them as pop-ups and then change their style to Child.

All of the *Window styles* options are disabled for child windows.

The *Frame style, Sizing (thick)* option is disabled for child windows.

The *Frame Style* options *Dialog* and *Border* are mutually exclusive, but not required.

Popup

A popup window is similar to a dialog box in windows. It has no menu and always appears above the main Subapplication window in runtime.

Frame style

Dialog

Creates a standard dialog box border.

Modal

Creates a dialog box with a modal frame that can be combined with a title bar and a system menu.

Sizing (thick)

Creates a standard sizing border that allows the dialog box to be resized.

Border

Creates a dialog box with a thin border.

Initial state in runtime

Visible (normal size)

During runtime, the SubWindow's initial state is to be displayed.

Hidden	During runtime, the SubWindow's initial state is to be hidden. You will need to attach a method to a trigger on the main window that calls up the hidden SubWindow.
Dialog styles	
Clip siblings	Clips child windows relative to the window.
Absolute align	Makes the x,y origin of the window relative to the screen.
Font and Color	Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.
Caption text	The text that will appear in the title bar of the SubWindow.
Initial caps	When set, text from the host has initial capital letters.

Table

The following parameters can be set in the *Style* tab

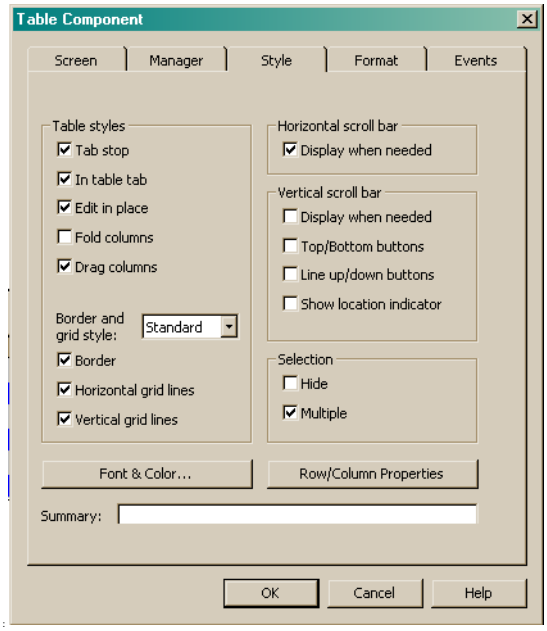


Figure 122. Table control Style tab

Table flags

Tab stop	When set, the user can jump into and out of the table using the *Tab* key.
In-table tab	When set, the runtime user can tab between the fields of the table.
Edit in place	Set this checkbox to edit tables in line. When cleared, the table lines are edited in a pop-up window.
Fold columns	For folded lists, set this checkbox to fold the table display. Should not be set together with the runtime multi-page option.
Drag columns	When set, the user can reorder the columns by dragging them. The feature must be enabled in runtime. In *JIS Administrator*, in the *Runtime Configuration* tab, select *List* from the *Category* combo box, and set the *Save new column order* checkbox. When *Fold columns* is set, the *Drag columns* option changes to *Drag columns when list not folded*.

Border and grid style

Standard	Gives table borders and grid lines a flat, one-dimensional appearance
Raised	Gives table borders and grid lines a slightly three-dimensional appearance
Border	When set, a border appears on the outside boundary of the table.
Horizontal grid lines	When set, horizontal grid lines separate the table rows from one another.
Vertical grid lines	When set, vertical grid lines separate the table rows from one another.

Horizontal scroll bar

Display when needed

Set to display a horizontal scrollbar when the table is too wide for its display area.

Vertical scroll bar

Display when needed

Set to display a vertical scrollbar when the table is too long for its display area.

Top/Bottom Buttons

This option is only relevant when the host application includes the ability to scroll to the beginning or end of a list.

When set, buttons are displayed at the top and bottom of the vertical scroll. These buttons trigger methods entitled *GetToTopOfList* and *GetToBottomOfList*. These methods are supplied empty. You must include method lines that perform the respective host operations for scrolling to the beginning or end of a list.

For Example:

If the *PF8* key scrolls to the end of a list, then the *GetToBottomOfList* method should include the line `HostType: AidKey: AidPF08 RemainInScreen: False`

Line up/down buttons

Toggle this option to hide or display the buttons located on the vertical scroll bar which are responsible for scrolling the data line by line. The default setting displays the buttons.

Show location indicator

Toggle this option to conceal or display the arrow on the sidebar which indicates the user's relative location within the list. The default setting displays the arrow.

Selection

Hide By default, when an item is selected in a table, it remains highlighted even when the focus changes to a different part of the screen. To cause the selection to lose focus, or to “hide” your selection, set this option.

Multiple Enables you to select multiple rows, columns or units. When cleared, the user may only select a single unit, row or column. The default setting permits multiple selections.

Font and Color

This button brings up the Font and Color dialog, but that dialog currently has no effect on the appearance of the table control.

Row and Column Properties

Selecting the Row/Column Properties button on the Table Component dialog displays the Row and Column Properties dialog.

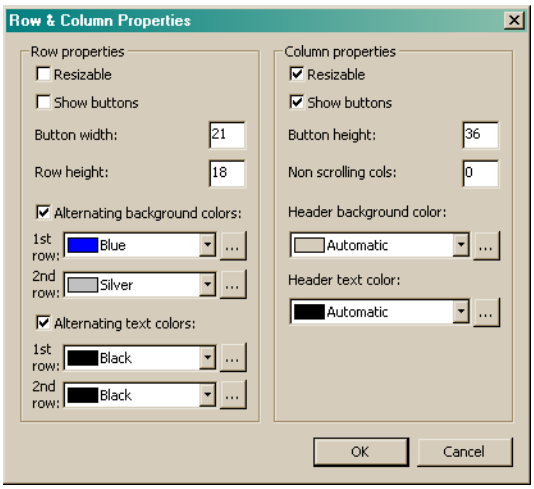


Figure 123. Row and Column Properties dialog

Row Properties

Resizable	When set, the user can resize a row's height by dragging the table's grid lines. All the rows are resized uniformly.
Show buttons	Affects the display of the numerical buttons for the rows. When set, these buttons are displayed in runtime. When cleared, they are not displayed in runtime.
Button width	The numerical size of the row buttons, in pixels.
Row height	The numerical height of the rows, in pixels.
Alternating background colors	When set, lets you alternate the background color of the table rows.
1st row	When using alternating background colors for table rows, controls the background color of the first table row and every second row after that.
2nd row	When using alternating background colors for table rows, controls the background color of the second table row and every second row after that.
Alternating text colors	When set, lets you alternate the color of the text in the table rows.
1st row	When using alternating text colors, controls the color of the text in the first table row and every second row after that.
2nd row	When using alternating text colors, controls the color of the text in the second table row and every second row after that.

Column Properties

Resizable	When set, the user can resize a column's width, by dragging the table's grid lines. All the columns are resized uniformly.
Show buttons	Affects the display of the column header buttons for the columns. When set, these buttons are displayed in runtime. When cleared, they are not displayed in runtime.
Button height	The numerical size of the column buttons, in pixels.
Non-scrolling cols	Specify how many columns remain stationary (and therefore visible) while scrolling the table horizontally.
Header background color	Lets you set the background color of the table headers.
Header text color	Lets you set the color of the text in the table headers.

Tabs

The following parameters can be set on the *Style* tab:

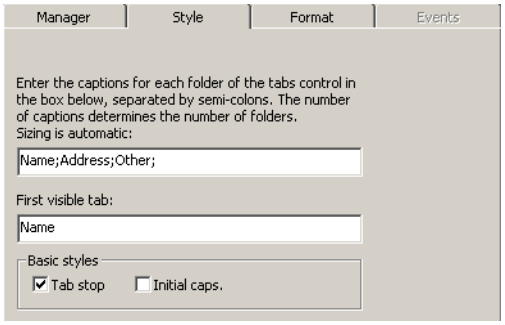


Figure 124. Tabs control Style tab

Tab stop	When set, allows you to jump to this control using the *Tab* key.

Initial caps	When set, text from the host has initial capital letters.
Uncaptioned edit field	Type the captions for each folder of the tabs control. Each folder name should be separated by a semicolon. The names of the folders should be the same as the corresponding SubWindow captions. You can adjust the sizes of the folders in Modify mode.
First Visible Tab	Type the name of the initially visible folder.

Variable

The Variable type does not appear on the window. Therefore, it does not have a *Style* tab associated with it.

Window

A Window is a rectangular area that displays a dialog box or a window.

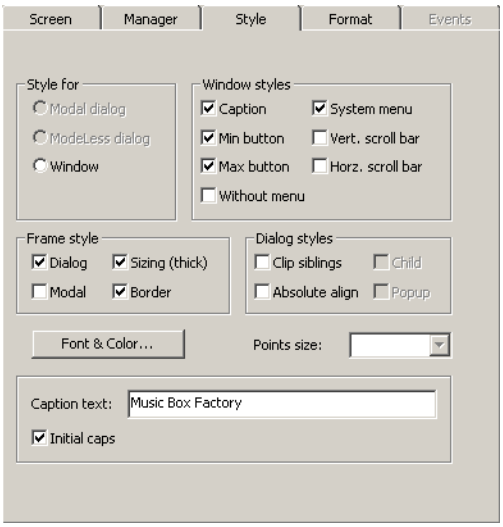


Figure 125. Window control Style tab

Style for

Modal dialog	For a dialog box. While open, a modal dialog does not allow you to shift focus to its parent dialog box. However, you can shift focus to any other dialog box. This option is not available yet, and is therefore disabled.
Window	When set, the control displays a window, instead of a dialog box.

Window styles

Caption	Displays a window with a title bar.
Min button	Creates a window with a minimize button.
Max button	Creates a window with a maximize button.
Without menu	Creates a window without a menu bar.
System menu	Creates a window with a system menu in its title bar.
Vert. scroll bar	Displays a window with a vertical scroll bar.
Horz. scroll bar	Displays a window with a horizontal scroll bar.

Frame style

Dialog	Creates a standard dialog box border.
Modal	Creates a dialog box with a modal frame that can be combined with a title bar and a system menu.
Sizing (thick)	Creates a standard sizing border that allows the dialog box to be resized using the mouse.

Border	Creates a dialog box with a thin border.
Dialog styles	
Clip siblings	Clips child windows relative to the window.
Absolute align	Makes the x,y origin of the window relative to the screen.
Child	Creates a child dialog box. (Currently Disabled)
Popup	Creates a child dialog box. (Currently Disabled)
Font and Color	Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307. You can not set the font or the text color for a Window. Currently, the only option available for a Window is to set the background color.
Caption Text	The text that will appear in the title bar of the dialog box/ window.
Initial Caps	When set, text from the host has initial capital letters.

TextBox

A TextBox control is a field in which the end user can type and edit information. The end user places the cursor in the control by using the *Tab* key or by clicking the mouse within the border.

The TextBox control appears as follows in a GUI window.

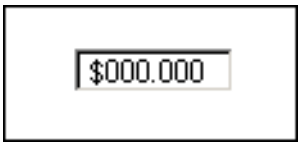


Figure 126. TextBox control

The typed text starts at the point configured for the control in the TextBox *Style* tab. If the box already contains text, all of the text in the box is automatically selected when the end user clicks in it and typed text replaces the original contents.

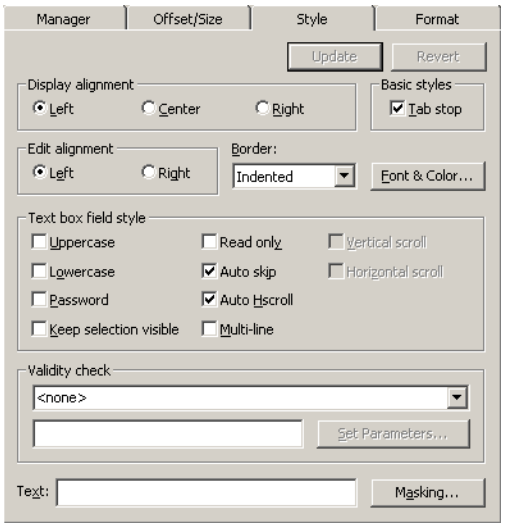


Figure 127. TextBox control Style tab

The following parameters can be set through the style tab:

Display alignment	Choose the appropriate radio button to determine whether displayed text is left-aligned, centered, or right-aligned. Centered text is applicable only when the control is multi-lined.
Edit alignment	Choose the appropriate radio button to determine whether user-entered text is left-aligned or right-aligned.

Basic styles

Tab stop	When set, allows you to jump to this control using the *Tab* key.
Border	Draws a border around a control. The border options are: *Elevated*: The TextBox appears higher than the background. *Indented*: The TextBox appears lower than the background. *None*: No border. *Standard*: The border is a line.

Text box field style

Uppercase	Converts characters to uppercase as they are typed.
Lowercase	Converts characters to lowercase as they are typed.
Password	Displays all characters as asterisks (*) as they are typed.
Keep selection visible	By default, an edit control does not appear selected when the control loses input focus and displays the selection with reversed colors when the control receives input focus. Setting this checkbox overrides the default.
Read only	The field may not be edited when this option is set.
Auto skip	When set, ACE skips to the next field on the window, if the edit is full and the cursor is on the last position in the field.
Auto HScroll	Automatically scrolls text ten characters to the right when a user types a character at the end of a line.
Multi-line	Creates a multiple-line edit control. When set, the *Vertical scroll* and *Horizontal scroll* options become enabled.

Vertical scroll	Gives the multiple-line edit control a vertical scroll bar.
Horizontal scroll	Gives the multiple-line edit control a horizontal scroll bar.
Validity check	For information on the available options, see “Validity Checks” on page 221.
Font and Color	Information on the available options can be found in Chapter 11 - "Color and Font Design" on page 307.
Text	This is the text that appears in the edit box. The text can have three different sources: • The Application INI file. In Design View, you can specify information to be read in from the Application INI file. For more information, see “The Manager Tab” on page 215. • Text typed in the *Text* edit field. • Text that appears in the host.

Masking Option

You can control user input using the *Masking* option in the *Style* tab. To illustrate this, let’s say you want a dollar sign to appear in the field on the end user’s GUI but you do not want it to be sent back to the host. Using the Masking option you can configure the runtime so that the dollar sign appears in the TextBox control followed by any number of alphanumeric characters. When the end user enters data in the field and presses the *OK* button, a mask prevents the dollar sign from being sent back to the host. This is just one example of how using the Masking option in the TextBox *Style* tab improves application performance.

Use the *Masking* dialog box to place characters that display as default in a TextBox field during runtime. These characters show the end user exactly what to enter in the field and how to enter it.

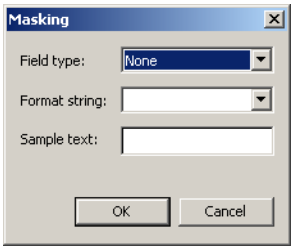


Figure 128. Masking dialog box

Assume that a GUI screen has a field where the end user enters the price of an item. In this situation a blank field can be confusing for the end user. The end user may not know if it is necessary to enter a dollar sign or a decimal point.

The Masking option allows you to display the dollar sign, and a decimal point separated by spaces for the numbers, as default in runtime. The user places focus on the field and enters the information. ACE automatically enters the user's input in the field as it should appear. When the user clicks *OK* only the information required by the host is sent back. In this case the host may not require the dollar sign.

In ACE's Test View, you can preview how the field will look in the runtime to make sure it is configured in the best possible way.

The Field type options in the Masking dialog box are:

None	Removes any formatting applied to the TextBox field.
Date and Time	Not supported at this time.
Mask	Customize the default display of TextBox fields in runtime. Determine which alphanumeric characters are sent back to the host and which characters are only used for display in the runtime.
Number	Use for numerical and monetary types. Symbolic placeholders are replaced by digits or other characters, when the field is displayed in runtime and when information is sent back to the host.

Field Token Place Holders

Mask formatting allows you to create custom TextBox fields. These fields contain characters that act as place holders showing your end users what to type in the field. You can control which characters to display in the field and which characters are sent back to the host. The format strings may be a combination of literal characters, which appear as themselves and/or tokens. Tokens are symbolic placeholders that are replaced by digits or other characters when the number is formatted as text. You, the developer, see only the token place holders in the *Mask* dialog box.

Table 12 shows the tokens and their permissible replacements:

Table 12. Field token place holders for masking option

Token Place Holders	Permissible Replacements
#	Any numeric digit (0-9).
@	Any alphabetic character (a-z, A-Z).
!	Any punctuation character.
*	Any single printable character.
\	Causes the next character to be treated as a literal.
&	Replaces any alphanumeric character.

To create masks for TextBox controls:

1. Open any ACE.INI file located in your ACE installation directory and create a new section called `[Mask Format]`.
2. On the next line enter the token characters that make up the mask, followed by the equals sign (=). For example, the following line could be used as a mask for the string Feb, 1992.
   `@@@,####=`
   You may add as many masks as you choose. Place each mask on its own line. End all masks with the equals sign.
3. Save and close the .INI file. Restart ACE.
4. In the TextBox control *Style* tab click *Masking*.
   The *Masking* dialog box opens.
5. From the *Field Type* list choose *Mask*.
6. Open the *Format String* list. The masks previously placed in the `[Mask Format]` section and saved in the INI file appear in the drop down list.
7. Choose a mask to apply to the field and click *OK*.
8. In the *Text* field in the TextBox component *Style* tab enter sample text. Make sure that you enter sample text in the TextBox component *Style* tab and not in the *Masking* dialog box. The sample text appears as default in runtime and serves as a guide for the end user. You do not have to enter sample text. Press *OK*.
9. From the *Design* menu choose *Apply Design Changes*. The TextBox control appears with the sample text formatted to match the mask.

Formatting Number Fields in TextBox Controls

The difference between the *Number* and *Mask* options in the *Mask* dialog box are in their token place holders. Tokens are symbolic placeholders that are replaced by digits or other characters when the number is formatted as text. You, the developer, see the tokens in the *Masking* dialog box only.

The *Number* field option has the following allowable token place holders:

Table 13. Field token place holders for textbox controls

Token Place Holders	Permissible Replacements
, (comma)	Inserts the international thousands separator every three places to the left of the decimal.
. (period)	Is replaced with the current international decimal indicator.
; (semicolon)	Separates formats for positive and negative numbers.
#	Is replaced by no digits or one digit.
*****	Is replaced by zero or more digits.
0 (zero)	Is replaced by one digit or a “0” (zero).
****	Causes the next character to be treated as a literal.
$	Followed by string.

To create Number fields for TextBox controls:

1. Open any ACE.INI file located under the ACE directory and create a new section called `[Number Format]`.
2. On the next line enter the characters that make-up the number field, followed by the equals sign. For example if a TextBox control is configured with a number field, `*0.*=` and the end user types 30.2500 in the field, 30.25 is sent back to the host in runtime.

   You may add as many number fields as you choose. Place each field on its own line. End all fields with the equals sign.
3. Save and close the INI file. Restart ACE.

4 In the TextBox control *Style* tab, click *Masking*.

   The *Masking* dialog box opens.

5 From the *Field Type* list choose *Number*.

6 Open the *Format String* list. The Number fields previously placed in the `[Number Format]` section and saved in the INI file appear in the drop down list.

7 Choose a number field to apply to the TextBox control and click *OK*.

8 In the *Text* field in the *TextBox* component *Style* tab, enter sample text. The sample text appears as default in runtime and serves as a guide for the end user. You do not have to enter sample text. Click *OK*.

9 From the *Design* menu choose *Apply Design Changes*. The TextBox control appears in the GUI window with the sample characters configured to match the number field.

The Format Tab

Each representation component has two kinds of window presentation properties:

- Some presentation properties are intrinsic to a component's *type*. See "The Style Tab" on page 221 for more information.
- Some presentation properties include external information, including the formatting of the character sequence the component represents. This formatting is performed on the *Format* tab.

Modifying Component Formats

To modify a component's format:

1 Open the *KnowledgeBase Definitions* window in Representation Definition view.

2 Select the desired component. In the *Properties* pane, select the *Format* tab.

3 Click *Format*. The *Text Format Definition* dialog box opens. In certain components, such as CheckBox and ComboBox, the *Format - Screen and Window Values Connection* dialog box opens.

4 Modify the format using the control elements in the dialog box.

5 Click *OK*. Your changes are stored and you return to the *Format* tab.

6 Save your changes by clicking the tab's *Update* button.

   Undo changes made since the last update by clicking the tab's *Revert* button.

Format Categories and their Purposes

- The control types Frame, GeneralUTMethod, Line, Button, Table, Tabs and Variable cannot be formatted.

- The CheckBox control is formatted using one form of the Format - Screen and Window Values Connection dialog box. The format specifies the character sequence, known as a screen value, associated with the two CheckBox states.
- The ComboBox, RadioGroup and ListBox controls are formatted using a second form of the Format - Screen and Window Values Connection dialog box. The format specifies the character sequence, known as a screen value, associated with each state of the control. The format also specifies the graphic element, known as a window value, associated with each state of the control.

All other controls are formatted using the Text Format Definition dialog box. The format modifies character sequences taken from the host screen before they are displayed on the window.

Check Box

In this dialog box you associate one or more host screen character sequences, known as screen values, with each of the two states of the check box. You can also specify that any other host screen value will result in a default value for the check box.

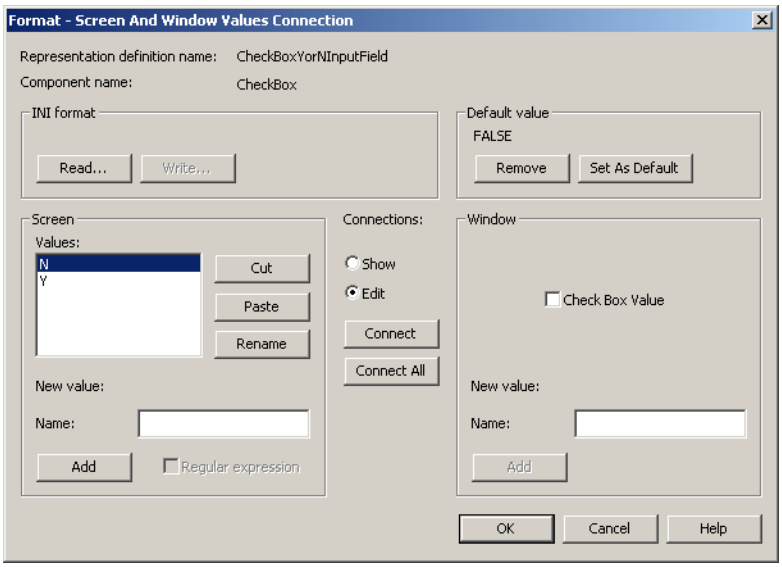


Figure 129. Format dialog box for Check Box

A Check Box control in a window has only two possible values: TRUE (set) or FALSE (cleared).

Screen Values

A list of all the possible screen values.

Connections

Lets you toggle between two modes: *Show* and *Edit*:

Show When selected, you can see the control value of the currently selected screen value in the *Window* area.

When the screen value is Y, the check box value in the window is TRUE (set).

To see the control value of a screen value, select the screen value by clicking it. This causes the corresponding control value to appear in the *Window* section. You can also click on the window value to see the *first* screen value that is connected to the window value.

You can connect more than one Screen Value to the same Window value.

Use the *Connect All* button to connect between screen values and window values when they have the same number of elements.

Edit When selected, the following options are available:

- Editing the values in the Screen area:
  Use the *Cut* and *Paste* buttons to edit the values.
- Adding new values to the values in the Screen area:
  Under *New Value*, type the new value in the *Name* field, and use the *Add* button to add it to the values list.

Figure 130. Adding new values

To change the format:

1. In the *Connections* area select the *Edit* radio button.
2. Edit the *Screen* pane so that it contains the desired values:
   - Cut unwanted values from the *Screen* list by selecting them and clicking the *Cut* button.
   - Add new values to the *Screen* list by typing the value in the *Screen* area's *Name* field and clicking the *Add* button.
   - Change a value by typing the new value in the *Screen* area's *Name* field, highlighting the *Screen* value to be changed, and clicking the *Rename* button.
   - Change the order of the *Screen* pane's values by cutting a value and then clicking the *Paste* button. The cut value appears at the end of the list.
   - To make the cut value appear immediately above a value, select that value and then click *Paste*.

Note: The order of the *Screen* values is purely cosmetic.

3. Select a check box value by clicking *Check Box Value* in the *Window* pane.
4. Select a corresponding screen value by highlighting an entry in the *Screen* pane.
5. From the *Connections* area click the *Connect* button to connect the selected screen and window values.
6. Select the other check box value and repeat steps 4 and 5.

Set As Default

You can also set the default value to be displayed in the Window, in case the host screen contains a value which is not listed under Screen, Values. To set the default value, select a *Check Box Value* in the *Window* area, and click the *Set As Default* button.

Ini Format

Write Choose a variable name for the checkbox in the *Manager* tab. Save the format under a symbolic format name either for the Application as a whole or for the current Subapplication.

Read When formats have been saved, apply one of the saved formats by selecting its symbolic format name.

Combo Box, Radio Group or List Box

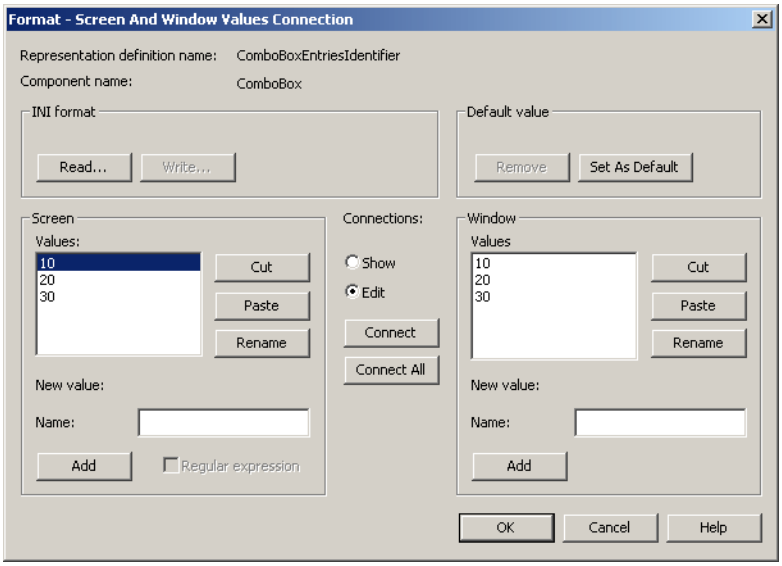


Figure 131. Format dialog box for Combo Box and Radio Group

The *Format* dialog box for a Combo Box and a Radio Group is only slightly different from the Check Box dialog box.

The difference is in the *Window* section of the dialog box. In contrast to a Check Box that can have only one of two known values, a Combo Box or a Radio Group can contain many different values. Therefore, the *Window* section contains a *Values* list, and the *New Value* option allows you to add any necessary values.

A small difference also exists between the way ACE handles INI file information for a CheckBox and a ComboBox. Write/Read INI format is not available for Radio Groups:

Write Choose a variable name for the control in the *Manager* tab. Save the format under a symbolic format name and the window values under a symbolic description name, either for the Application as a whole or for the current Subapplication.

Read When formats have been saved, apply one of the saved formats by selecting its symbolic format name and one of the saved list of window values by applying its symbolic description name.

Text Formatting

When a representation component displays host screen text, you can change the format of the text on the representation component's *Format* tab. The representation component's *Format* tab leads to the *Text Format Definition* dialog box:

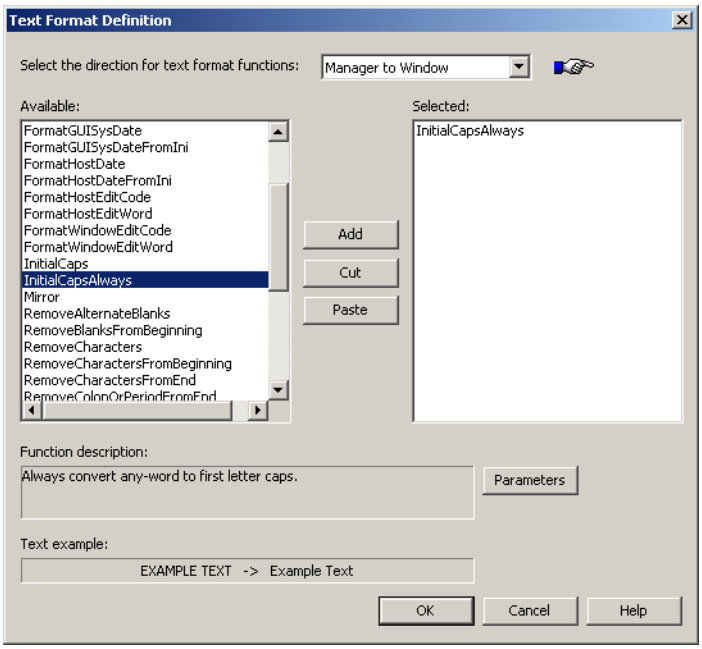


Figure 132. Text Format Definition dialog box

1. Choose a formatting function from the *Available* group box.
   A description of the function appears in the *Function description* box. The result of applying the formatting function to text appears in the *Text example* box. The host screen text appears to the left of the arrow, the effect of the formatting appears to the right of the arrow.
2. Click the *Add* button to add the chosen formatting function to the *Selected* group box.
3. Remove formatting functions from the *Selected* group box by clicking *Cut*.
4. Use the *Cut* and *Paste* buttons to change the order of functions in the *Selected* group box.

Formatting User Input

To format user-input text that is passed to the host, set *Select the direction for text format functions* to *Window to Manager*.

Formatting Accelerators with User Dictionaries

Certain function keys which exist on the host computer may not be present on the standard PC keyboard. To access these function keys, ACE provides default accelerators defined in user-defined dictionaries. The definitions for the accelerator keys are registered in these dictionaries.

To modify the function key assignments in the dictionary:

1. In the *Text Format Definition* dialog box select the Use Dictionary and Use Dictionary Phrase options from the *Available* list and add them to the *Selected* list.
2. Double-click on the *UseDictionary* selection or click the *Parameters* button. The *List of Dictionaries* dialog box opens:

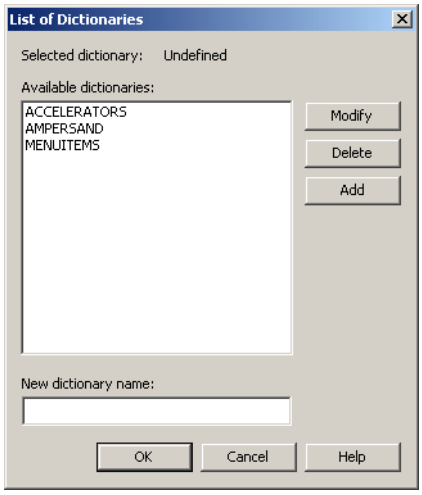


Figure 133. List of Dictionaries dialog box

3. Select the *Accelerators* option from the *Available Dictionaries* list and click *Modify*. The *Word Dictionary* dialog box opens:

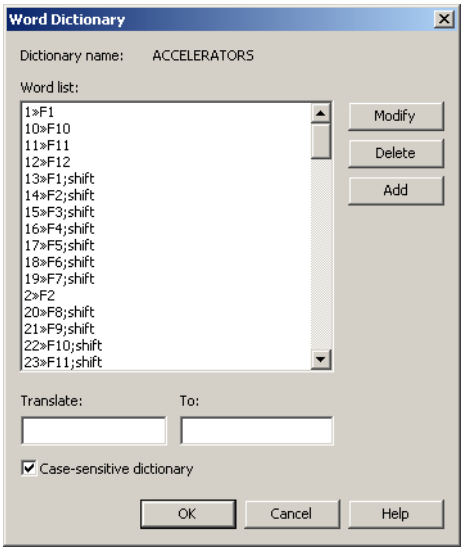


Figure 134. Word Dictionary dialog box

4 In the *Translate* field, type the host function key combination. In the *To* field type the PC function key combination.

Formatting Text with User Dictionaries

You can replace host text with text specified in a user dictionary.

To add a formatting function to a user dictionary:

1 In the *Text Format Definition* dialog box select the UseDictionary, UseDictionaryPhrase, or UseDictionaryLongestString format functions from the *Available* list and add them to the *Selected* list.
2 Double-click on the UseDictionary selection or press the *Parameters* button. The *List of Dictionaries* dialog box opens:
3 Select the required dictionary, or add a new dictionary by typing its name in the *New dictionary name* field and clicking *Add*.
4 Double-click on the selected dictionary or click *Modify*. The *Word Dictionary* dialog box opens.
5 In the *Translate* field, type the text as it appears on the host; in the *To* field, type the text that you want to appear on the GUI.
6 Click *Add*. Click *OK* to close the dialog boxes.

Formatting Function UseDictionaryPhrase

The formatting function UseDictionaryPhrase applies the format to the whole text but also to each word of the sentence, even if this means a double round of formatting.

For example, if the dictionary translates:

```
SYS REQ ==> SYSTEM REQUIREMENTS
SYS  ==> SYSTEM
```

the text `SYS REQ` is translated to:

```
SYSTEMTEM REQUIREMENTS
```

To apply the format to the whole given text, treating it as a single sentence, use the following setting in the FORMATS.INI file located in the <InstallDir>\<applname> directory:

```
[General]
CorrectUseDictionaryPhraseFunction=1
```

Chapter 10. Editing Menus in Design View

Every window has a menu bar displayed immediately below the caption bar. This is sometimes referred to as the top-level menu.

ACE contains a tool for editing the menu bar and the items in each menu. Using the Menu Editor, you can customize your window's menu. In addition, ACE enables you to create menus that are prompted in the runtime application when the user clicks the right mouse button.

This chapter describes:

- The Structure of a Pull-down Menu
- Editing Options
- Working with the Menu Editor
- Floating Menus
- Changing the Default Menu for All Subapplications
- Creating Menus Automatically with Representation Definitions

The Menu Bar

Every window is created with a standard menu bar containing the following items:

- *Application* - contains the following submenus:
  - *File* - contains the items: Exit; Print GUI; Print Host Screen.
  - *View* - contains the item Host Screen.
  - *Emulator* - contains the item Save Host Screen Image.
  - *Application* - contains the items: Run; Synchronize; Undo Window; Printer Emulation.
  - *Help* - contains the item About.
- *File* - contains the item Exit Application.
- *Edit* - contains the items: Undo; Cut; Copy; Paste, Refresh Window.
- *Commands* - contains the item OK.
- *View* - contains the items: Host Screen; Window Information.

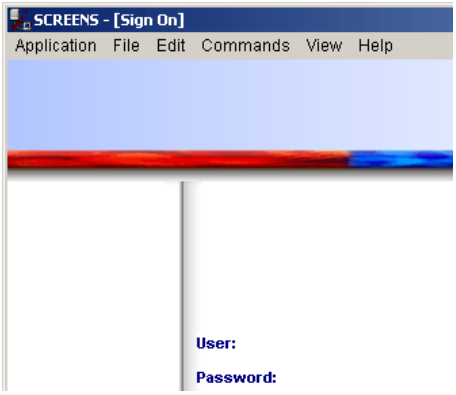


Figure 135. A menu bar

In most cases, the standard menu items are not sufficient. You will need to develop the window menu by adding other menu items.

For example, your Application may have a toolbar with options that are common to a group of screens. You can add these options to the menu bar so that they can be accessed both through the on-screen buttons and as options in a pull-down menu. You can also assign a key combination so that they can be accessed from the keyboard.

The Structure of a Pull-down Menu

A pull-down menu can contain:

Item	This initiates a command, invokes a dialog box, or represents an option.
Popup	This opens a submenu.
Separator	A separator is a horizontal line that visually separates a menu's items into groups.

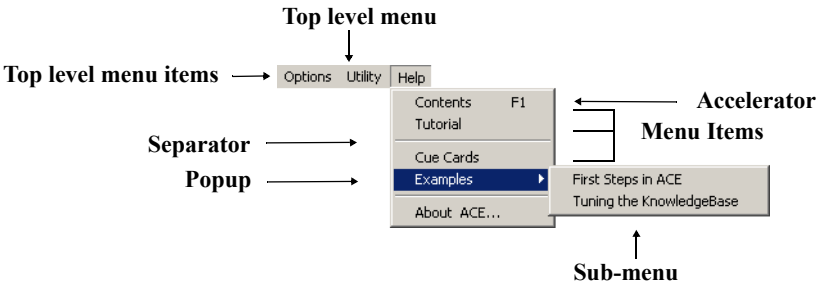


Figure 136. Structure of a pull-down menu

Selecting the Menu Type

ACE enables you to configure two types of pull-down menus:

Subapplication Menus	Pull-down menus accessed from the menu bar at the top of the window.
Floating Menus	Control-specific menus accessed by right-clicking a specific control.

In Design View, from the *Design* menu, select either the *Subapplication Menu Editor* or *Floating Menus* to open the respective menu editors.

Subapplication Menu Editor

Select *Subapplication Menu Editor* from *Design* menu to display the following dialog box. Here you can configure your pull-down menus.

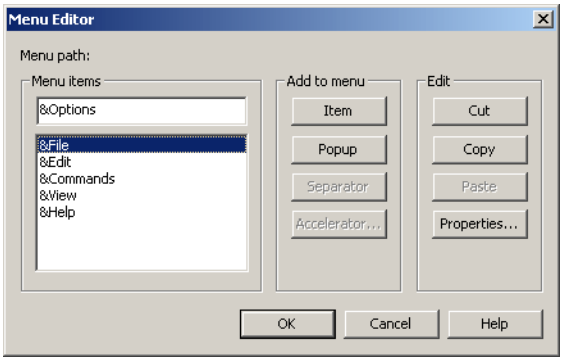


Figure 137. Subapplication Menu Editor dialog box

The *Menu Editor* dialog box is composed of the following elements:

Menu path	Displays the path of the current menu beginning from the top level menu.
Menu items	Contains a list of all the items that belong to the current menu. Contains an edit box for typing menu item entries. Ampersands (&) appearing as part of a menu item indicate that the character following the ampersand functions as a hot key.

Add to menu	Contains tools for adding an Item, a Popup (sub-menu), a Separator, and an Accelerator to a menu.
Edit	Contains editing tools that operate on an item selected in the *Menu items* list.

Editing Options

The *Menu Editor* contains the standard editing options: *Cut*, *Copy*, and *Paste*. Use these options to edit or rearrange the items in the *Menu items* list.

Cut, *Copy* and *Paste* function in the standard manner:

Cut and Copy	Operate on the selected item in the list. **Note:** When using Cut or Copy on a Popup item, they operate on the Popup item and all its sub-items.
Paste	Inserts an item before the selected item in the list, or at the end of the list when no item is selected.

One of the common uses of *Cut* and *Paste* is to move the *Help* menu bar item to the end of the list. This causes *Help* to be the right-most item on the menu bar, as is customary in *Windows* applications.

The Menu Items List Box

You can access any menu item of the window menu through the *Menu Item*s list. The *Menu Editor* uses some of the conventions of DOS directories:

- Double-clicking a Popup item in the list causes it to become the current menu. *Menu items* displays a list of all the items that belong to the current menu, preceded by two dots (..) at the top of the list.
- Double-clicking the two dots at the top of the list takes you back to the previous menu level.

The Properties Button

Selecting an item in the *Menu items* list and clicking the *Properties* button displays the *MenuItem Component* dialog box, where you can change the properties of that item. This dialog box is slightly different from the *Style* tab of the *MenuItem* component in the KnowledgeBase. In this dialog box you cannot set the *Popup menu* or *Separator* properties, since they are already determined at this point.

Also, this dialog box does not have the *Add to* area that appears in the *Style* tab since the item is already in the menu.

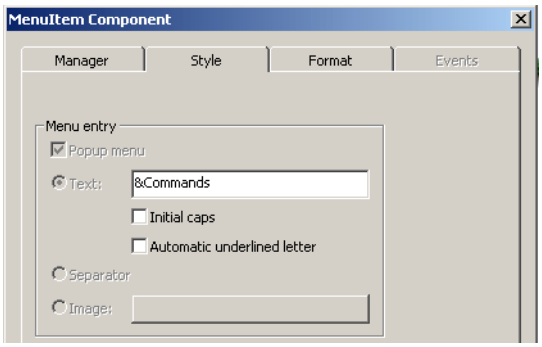
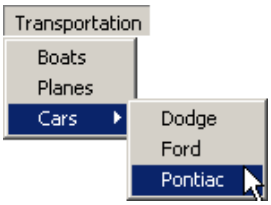


Figure 138. MenuItem Component dialog box

> **Note:** If you have made changes to your KnowledgeBase and not yet applied the changes, do not click *OK* in the *MenuItem Component* dialog box followed by clicking *Cancel* in the *Menu Editor* dialog box. Doing so causes the changes to your KnowledgeBase to be applied to your Subapplication.

Working with the Menu Editor

In this section you will learn how to configure the Menu Editor to create new menus and modify existing ones. Take for example the following menu and sub-menu:



To produce such a menu you have to:

- Create a new top level menu
- Add a menu item to a top level menu

- Add a sub-menu
- Attach functionality to a menu item

To enhance your menu:

- Add a separator to a menu
- Create an accelerator for a menu item
- Create a hot key for a menu item

In addition, you may wish to execute the following operations on an existing menu:

- Reorder menu items
- Change the name of a menu item

Creating a New Top Level Menu

To create a new top level menu:

1. In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2. In the *Menu items* edit box type the name of the new menu and click *Popup*. ACE adds the new menu item to the *Menu Items* list.
3. Click *OK*.

Adding a Menu Item to a Top Level Menu

To add a menu item to a top level menu:

1. In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2. Double click a top level menu.
3. In the *Menu items* text box type the name of the new menu item and click *Item*.
4. Click *OK*. ACE adds the new menu item to the *Menu Items* list.

Adding Sub-menus

To add a sub-menu:

1. In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2. In the *Menu Editor* dialog box navigate to the menu level at which the sub-menu will be added.
3. In the *Menu items* text box type the sub-menu name and click *Popup*.

Note: Notice the menu path of the new popup changes to reflect the fact that it is a sub-menu.

4 In the *Menu Items* text box, one-by-one, type the names of the new menu items contained in the sub-menu and click *Item*.
5 Click *OK*.

Note: A new menu item has no functionality. Add functionality by assigning methods.

Reordering Menu Items

To reorder menu items:

1 In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2 Navigate to the menu item to be moved.
3 Select the menu item that you want to move and click *Cut*.
4 Select the menu item just below where you want to place the cut menu item and click *Paste*. The menu item is placed in the new position.
5 Click *OK*.

Changing the Name of a Menu Item

To change the name of a menu item:

1 In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2 Navigate to the menu item that you want to rename.
3 Select the menu item to rename and click *Properties*. The *MenuItem Component* dialog box appears.
4 In the *Text* box, type the new name.
5 Click *OK*.

Attaching Functionality to Window Menu Items

To add functionality to window menu items:

1 In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2 Navigate to the item to which you want to attach a method.

3 Select the menu item and click *Properties*. The *MenuItem Component* dialog box appears.
4 Select the *Events* tab.
5 Select the desired method from the *Activate method* list.
6 Click *Link*.
7 Click *OK* twice.
The desired method is now attached to the menu item.

You can verify that the correct method is attached by selecting the menu item in Test View and reading the information in the message box.

Note: Detailed information about how to use the *Events* tab to attach methods is found in "Linking Methods to Controls in a Subapplication" on page 356.

Adding a Separator to a Menu

A separator is a horizontal line which visually separates menu items. Separators can enhance the usability of your drop down menus and sub-menus by separating items into groups. Use the *Separator* button in the *Menu Editor* to add separators.

To add a separator to a menu:

1 In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2 Navigate to the menu item which will sit just below the separator and select it.
3 Click *Separator*.
4 Click *OK*. ACE adds a separator to the drop down menu.

Creating an Accelerator for a Menu Item

Accelerators provide access to a menu item without having to open the window menus. Accelerators differ from hotkeys in the respect that hotkeys only function once the menu has been opened.

To create an accelerator for a menu item:

1 In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2 Navigate to a menu item.

3 Select the menu item and click *Accelerator*. The *Accelerator Style* dialog box appears.
4 From the *Key* list choose the accelerator to attach to the item. Click *OK*.

Note: Check *Add to menu item* if you want the accelerator key shortcut to appear alongside the menu item during runtime. Top level menu items and menu items designated as pop-ups do not accept accelerators.

Creating a Hotkey for a Menu Item

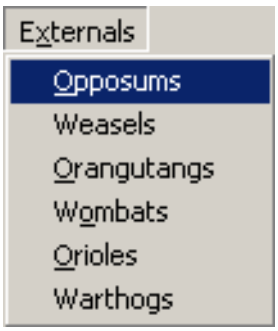
Once the menu is open, hotkeys provide quick keyboard access to a menu's items. A menu item's hotkey is one of the letters spelling out the item.

Each menu item created through KnowledgeBase analysis of a screen image has a hotkey. This hotkey is indicated on the menu item by an underscore. Within the *MenuItem Component* dialog box the hotkey is preceded by an ampersand (&).

Automatically Assigning Hotkeys to Menu Items

When you create a menu item manually in the *Menu Editor* and do not include an ampersand, by default the first letter of the item is the hotkey, and there is no underscored letter in the menu item. If you create two or more items that all start with the same letter and you do not use an "&" to assign the item's hotkeys, the items will all have the same letter—their first letter—as the hotkey.

Example 43. Automatically assigning hotkeys

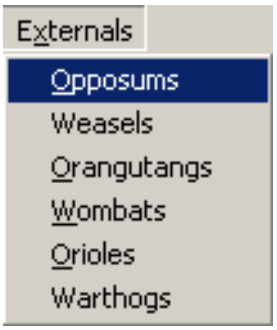


- The menu items *Opossum*, *Orangutangs*, *Wombats* and *Orioles* all have *O* explicitly assigned as their hotkey.
- The menu items *Weasels* and *Warthogs* both have *W* automatically assigned as their hotkey.

When a menu contains items with identical hotkeys you can cycle the selection highlight among the items with the shared hotkey and invoke the appropriate item.

To invoke a menu item with a shared hotkey:

1 Press the hotkey until the correct menu item is selected.
2 Press *Enter*. The selected menu item is invoked.



In this example, *W* is the automatically assigned hotkey for *Weasels* and *Warthogs* and also the explicitly assigned hotkey for *Wombats*. Thus, typing *W* does not cause the selection highlight to cycle among *Weasels, Wombats* and *Warthogs*.

Explicitly Assigning Hotkeys to Menu Items

You can set an underscored hotkey for a menu item at any time, by placing an "&" in the desired position in the item's name. This is valid for menu items at any menu level.

To set the underscored hotkey for a menu item:

1 In Design View select *Design* > *Subapplication Menu Editor*. The *Menu Editor* dialog box appears.
2 Navigate to a menu item and select it.
3 Click *Properties*. The *MenuItem Component* dialog box appears.
4 In the *Style* tab, in the *Menu Entry* text box, retype the menu item name with an "&" preceding the character you wish to be the underscored hotkey.

   -or-

   Set the *Automatic underlined letter* checkbox. In this case, ACE inserts/moves the "&" before the first item-letter that has not yet been used as a hotkey for any other item in the menu. Consequently, it is best to assign hotkeys first to shorter items and then to longer items.
5 Click *OK* twice. The selected letter becomes the item's underscored hotkey.

Note: Dynamic menu items also have hotkeys assigned to them. Therefore, if a menu might receive dynamic menu items, you should assign the static item hotkeys in such a way that the dynamic items contain letters previously unused as hotkeys.

Floating Menus

One of the user-friendly features of some *Windows* programs is the ability to call up content-specific menus. Rather than always accessing menu options from the pull-down menus on the program menu bar, menus with options that are specific to the item selected can be displayed. These are accessed in runtime by clicking the right mouse button after a control has been selected. The *Floating Menu* tool is used to develop your own action-specific menus that are prompted by the user in runtime after selecting one of the GUI controls.

In this feature you must establish the conditions and parameters of the menu, including; the name of the menu and its items, the types of items, and the functionality of the items. The title of the menu and its items are created in Design View. When this is completed you can attach methods to controls which prompt a response when selected by the right mouse button.

During runtime these user-generated menus are called up by placing the cursor on a control that has been assigned a menu, and right-clicking. The menu which is prompted by this action is called a “floating menu.”

The *Floating Menu Editor* is accessed via the *Floating Menus* dialog box.

To work with floating menus:

In Design View, select *Design* > *Floating Menus*. The *Floating Menus* manager opens.

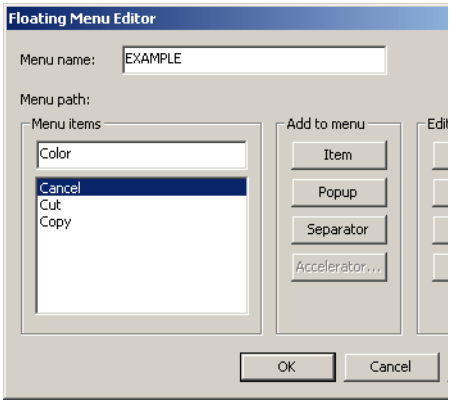


Figure 139. Floating Menus manager

New	Opens the *Floating Menu Editor* dialog box with empty fields.
Modify	Opens the selected floating menu in the *Floating Menu Editor* dialog box.
Assign	Opens the *Assign Floating Menu Triggers* dialog box. This is used to assign functionality to items in the floating menu.
Delete	Deletes the selected floating menu.
Close	Closes the *Floating Menus* manager.

Note: The *Floating Menu Editor* can also be opened via the *Define* menu in the KnowledgeBase.

Clicking *New* or *Modify* in the *Floating Menus* manager opens the *Floating Menu Editor* dialog box:

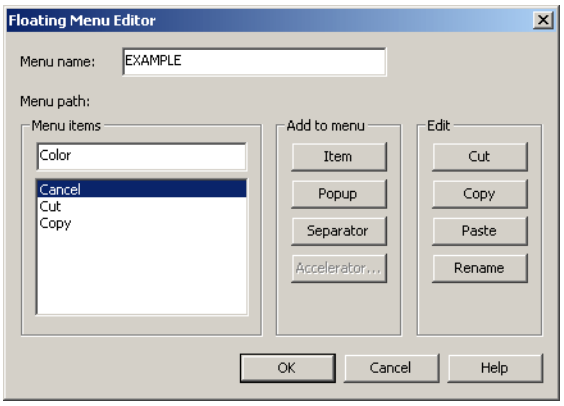


Figure 140. Floating Menu Editor dialog box

Menu Name	The identifier for the menu. You must enter the name as part of the process of making the floating menu functional.
Menu Items	Displays all the menu items currently in the menu. In the edit field type the names of new menu items.

When adding menu items to the menu consider the following:

- Select the type of item(s) to appear in your menu from the *Add to Menu* options. An *Item* directly performs an action, including opening dialog boxes. A *Popup* opens a sub-menu.
- If choosing *Popup*, you will be prompted to input the name of the child menu.

When you have finished creating your menu, click *OK* to return to the main window in Design View.

To use the floating menu you must give each item functionality and attach the menu to a right mouse button trigger.

Attaching Functionality to a Floating Menu

Floating menus are shortcut menus that provide options which are specific to controls. These options are usually more narrow in scope than those found in the main menus. The runtime can be configured to display floating menus when you right-click on a control.

To create a floating menu you must:

- Create a floating menu and menu items
- Attach methods to the menu items
- Attach the floating menu to a right mouse button click

A floating menu can be assigned to a control. When this control receives a right click during runtime, a floating menu is displayed. The items in the floating menu are called menu items. Methods are attached to the menu items giving them functionality. During runtime, when a menu item is clicked, the function of the method attached to the menu item is performed.

Creating a Floating Menu and Menu Items

This section describes how to create a floating menu, give it a name and assign it menu items. The floating menu's name should be general in nature and indicate the menu's basic function. Menu item names should be more specific. A menu item name should reflect the function that it performs during runtime. It should also describe the action that takes place when you click the menu item.

To create a floating menu:

1. In Design View select *Design* > *Floating Menus*.
   The *Floating Menus* manager opens.
2. Click *New*. The *Floating Menu Editor* dialog box opens.
3. In the *Menu name* field, type a name for the new floating menu.
4. In the *Menu items* field, type the name of a menu item. Click *Item* to add a menu time that directly performs a function. Click *Popup* to add a menu item that opens a sub-menu. Repeat this step for each menu item.
5. Click *OK*. ACE creates the new floating menu.

Attaching Methods to Floating Menu Items

This section describes how to attach methods to menu items. Menu items are the options in the floating menu. They are like triggers that instruct ACE to activate methods. Methods give menu items functionality. A menu item without an attached method does not do anything during runtime.

To attach a method to a floating menu item:

1. In Design View select *Design* > *Floating Menus*.
   The *Floating Menus* manager opens.
2. From the list of defined menus select a floating menu and click *Assign*.
   The *Assign Floating Menu Triggers* dialog box opens.

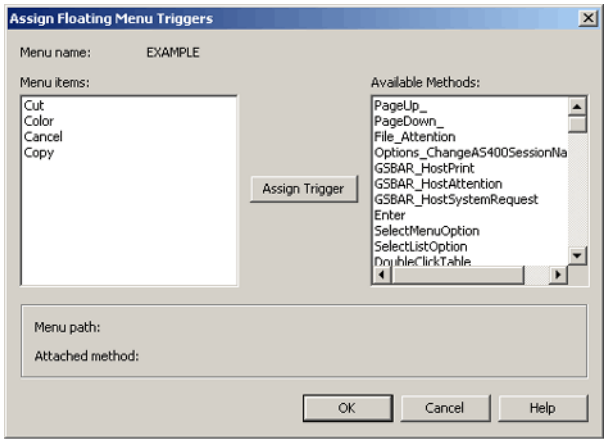


Figure 141. Assign Floating Menu Triggers dialog box

3 Highlight an entry in the *Menu Items* list and an entry in the *Available Methods* list.
4 Click *Assign Trigger*.
5 Click *OK*.

All menu items must be attached to a method. If a menu item does not receive a method (that is, it is not assigned a trigger) a warning message is prompted. It is permissible, though not advised, for a menu item to be void of functionality.

Menu Items	Displays the menu items specified in Design View.
Available Methods	Displays the list of available General UTMs including the General UTMs provided by ACE.
Assign Trigger	Assigns functionality to items when a single item in the *Menu Items* list and a single General UTM in the *Available Methods* list are selected.

Note: A popup does not receive a method; only the popup's child receives a method.

Attaching the Floating Menu to an RMB Click

This section describes how to configure ACE to display a floating menu when you right-click on a control during runtime. This is done by modifying a System-Triggered Method corresponding to the type of control on which an RMB opens the floating menu.

To attach the floating menu to a specific control type:

1. In Design View, select *Design* > *System-Triggered Methods*.

   The *System-Triggered Methods* dialog box opens.
2. From the list of General Methods, select the *UserRMB* method corresponding to the specific control.

   If there is a red check mark to the left of the method's name, click *Empty* to discard the contents of the method. Click *Yes* when asked to confirm your choice, and then click *Modify*

   If there is no check mark to the left of the method's name, click *Modify*.

   The *Define Method* dialog box appears.
3. Select *DoMethod* from the *Line type* list and click *Add Method Line* . The *DoMethod: Method Activation* dialog box appears.
4. From the *Method* drop-down list, select *Full List*.
5. From the *Executed By* list choose *Window*.
6. From the *Method* list choose *SetCurrentMenuFromFloating* and click *Assign Values*. The *Method Parameters* dialog box appears.
7. In the *Parameter value* field, within double quotes type the name of your floating menu, exactly as it appears in ACE. For example: "FLOATING MENU".

Note: The floating menu name appears in the *Floating Menu Editor* dialog box. To open the *Floating Menu Editor* dialog box refer to the procedure "Creating a Floating Menu and Menu Items" on page 300.

8. Click *OK* to close the *Method Parameters* dialog box and then click *OK* to close the *DoMethod: Method Activation* dialog box. The *Define Method* dialog box remains open in order to add another method line.
9. Select *DoMethod* from the *Line type* list and click the *Add Method Line* icon. The *DoMethod: Method Activation* dialog box appears.
10. Select *Full List* from the *Methods* drop-down list.
11. In the *Executed By* pane select *Window*.
12. From the *Method* list choose *TrackRightMBCurrentMenu*.
13. Click *OK*, then click *Close* to return to the Subapplication. The new floating menu will now open in runtime when the user right clicks on the control type specified in this procedure.

Note: If you wish to have a floating menu attached to a control in a specific Subapplication but not in controls of the same type in the whole Application, perform this procedure in the Subapplication and modify the Current Subapplication System-Triggered method instead of the General method.

Changing the Default Menu for All Subapplications

Menu items in the pull-down menus and accelerators have language-dependent Application-wide default values. You can create new defaults that will apply to all new Applications of a given language.

> **Note:** The number of pull-down menu, submenu and separator items defined as the default must total together less than 570. Moreover, a large number of defined default items can slow ACE and the runtime significantly.

To change the default entries for menu items and accelerators in pull down menus:

1. Open any Application and Subapplication.
2. Follow the regular ACE procedures for editing menu items and accelerators.
3. Save the Subapplication. This generates the files ...\APPLS\<appl_name>\<subappl_name>.MNU and ...\APPLS\<appl_name>\<subappl_name>.ACC
4. Make a copy of the *MNU file, naming the copy --XX----.--- (8 dashes+'.'+3 dashes, where the dashes are minus signs) where XX is the two letter language identifier; EN=English; DT=Dutch; FR=French; GR=German; IT=Italian; PT=Portuguese; SG=Swiss-German; SP=Spanish and SW=Swedish.
5. Copy the renamed file to ...\INITAPPL_DEFAULT.MNU, overwriting the existing file.
6. Return to ...\APPLS\<appl_name>\ and make a copy of the *.ACC file, naming the copy --------.--- (8 dashes+'.'+3 dashes).
7. Copy the renamed file to ...\INITAPPL_DEFAULT.ACC, overwriting the existing file.

> **Note:** The accelerator defaults are not language specific.

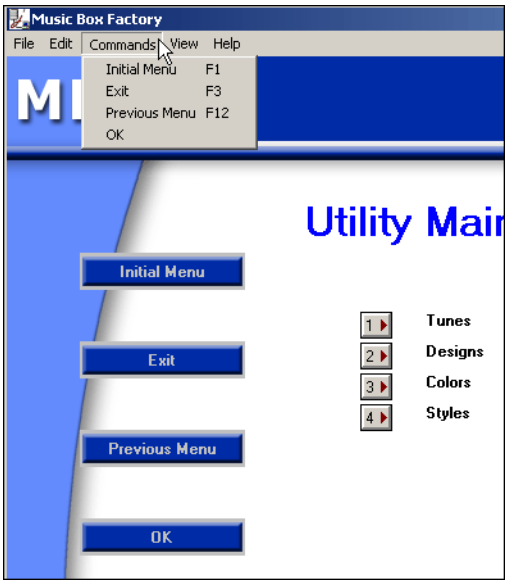
Creating Menus Automatically with Representation Definitions

ACE enables you to add items to the window's menu automatically, using Representation Definitions. A component usually defines a control to represent a pattern in the window. However, you can also use a component to define a menu item.

Example 44. Creating menus automatically

Menu options on a screen are often represented by a button in the window. You can define another component for the representation of a menu option, that creates a menu item for each menu option. You can also specify under which menu bar item the menu items will appear. In this way, every menu option that is represented by a button, is also represented by a menu item in the window.

In the following figure, the menu item representation created the *Commands* menu that contains all the menu options:



Creating an Automatic Menu Item

A Menu Item representation is defined like any other Representation Definition. In most cases, you define a Menu Item representation in addition to the control representation of a Pattern Definition. Therefore, you usually modify an existing Representation Definition by adding a component to it. Occasionally, you may also have a Pattern Definition you want represented only by a Menu Item, and by no control. In such a case, you define a new Representation Definition.

The options available in the MenuItem's *Style* tab are:

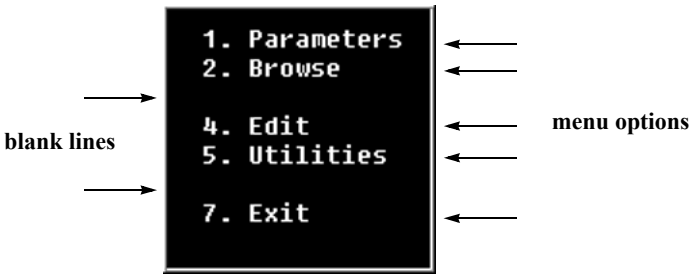
Popup Menu When you set this check box, the Menu Item will be a popup. A popup item is an item that invokes another popup menu.

Text	The text you specify in the edit field overrides the text derived from the Pattern Definition.
Initial Caps	When set, the text will have initial capital letters.
Automatic Underlined Letter	When you set this check box, a letter from the text is automatically underlined. This means that you can activate this item with the keyboard, using the *ALT* key and the underlined letter.
Add to	Specify the location of the new menu item: • Menu bar item: In the edit field, specify the menu to containing this option.In the example below, the menu name is *Options*. • The window's menu bar. • Most recently added menu bar item. • Most recently added popup menu.
Separator	You can specify a Menu Item representation that is a Separator. A Separator is a horizontal line that separates between the items of a menu. A separator has no functional significance.

Example 45. Creating automatic menu items

You could use the Separator in the representation of a menu in which there are blank lines between the menu options.

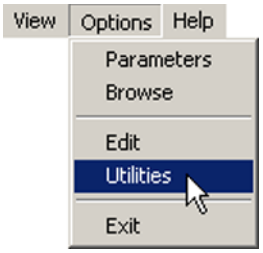
Let us refer to the following extract of a screen that contains a menu:



In such a case, you use two Menu Item representations:

- Each menu option is represented by a textual Menu Item.
- Each blank line is represented by a Separator.

If we assume these items appear under the *Options* menu bar item, this is how the *Options* menu looks:



Note: Currently, it is recommended not to use Menu Item representations for shrinking menus. In the host application there may be menus in which some of the options are disabled, according to the user. Such a menu may be displayed in two ways. Either the unavailable options are replaced by blank lines so that the menu maintains its original size, or else the menu is shrunk to contain only the available options (with no blank lines in between). In the latter case, do not use Menu Item representations.

Chapter 11. Color and Font Design

The colors and fonts used in the windows of your generated Application are an important tool both for making your generated window more interesting, and for emphasizing certain areas of the screen. This chapter provides you with the necessary information required to design the color and font of your Application.

This chapter describes:

- About Control Colors
- The Scope of a Color Mechanism
- Setting a Control's Color in the Font and Color Dialog Box
- Selecting the Font

About Control Colors

You can specify the colors of each control in your Application. There are two aspects to this specification:

- The specification mechanism. The type of information that is translated into a control's color, as well as detailed mapping of the possible values of the information to the possible control colors.
- The scope of the specification. Generally, you use the KnowledgeBase to specify that controls attached to a given Pattern Definition, or all controls that are components of a given Representation Definition have the same color mechanism. However, you can override the general mechanism on a control-by-control basis.

You are free to mix mechanisms and scopes as you wish. You can use the KnowledgeBase to specify that a particular control linked to a particular Pattern Definition uses one mechanism, while a second control, even of the same type, linked to the same or any other Pattern Definition uses a different mechanism. Moreover, when you override a particular control's color specification mechanism. You are not limited by the mechanism generally used by that control, or any other control.

Color Specification Mechanisms

A control can receive a fixed color. The control is always displayed in this color, regardless of the state of the host/JISApplication.

A control can be the same color as some element of the *Windows* system. In this situation, if the end user changes the color of that *Windows* element, then the control's color will also change in the same way. This allows you to ensure that control colors blend with the overall look of each user's desktop. The color is not affected by the state of the host/JIS Application.

A color table sets the control's color according to the runtime color of the host characters recognized by the control's underlying Pattern Definition. In this way, you can preserve how the host visually indicates the state of the Application. For example, when you use a color table, you can make a control change color whenever the host characters recognized by the underlying Pattern Definition become highlighted.

The color that is actually displayed by the control can be either a fixed color or the color of a *Windows* system element.

To use a host color table, enable your Application to read colors from the host.

Note: A control manually added in Design View is not affected when setting its color from the host color table.

The Scope of a Color Mechanism

Most control components in the KnowledgeBase can be given a color. A default color mechanism is supplied for every such control component, whether you add the component to the KnowledgeBase or whether the control component already exists in the KnowledgeBase.

Controls generated from the KnowledgeBase take their color from the KnowledgeBase settings. This is true both for controls attached to Pattern Definitions and for controls attached to floating Representation Definitions that you add manually in Design View.

You can also make a local change to any control's color mechanism.

Specifying the Color Mechanism Globally

Setting up your colors through the Representation Definitions in the KnowledgeBase is a good way of automatically giving your generated application a uniform look.

To set a control component's color mechanism in the KnowledgeBase:

1. Open the KnowledgeBase and go to Representation Definitions View.
2. In the *Representation Components* area select the control you wish to modify.
3. In the *Style* tab, click the *Font and Color* button:
4. The *Font and Color* dialog box opens. See the following sections for details.

Specifying the Color Mechanism Locally

To set the color mechanism of a single control.

1. In Design View, open the Subapplication containing the control you wish to modify.
2. Select the control.
3. From the *Design* menu choose *Arrange* > *Set Font and Color*. The *Font and Color* dialog box opens.

   -or-
4. In Design View, open the Subapplication containing the control you wish to modify.

5 Double click the control. The control's *Component* dialog box opens.
6 In the *Style* tab, click *Font & Color*. The *Font and Color* dialog box opens.

Setting a Control's Color in the Font and Color Dialog Box

The *Font and Color* dialog box is where you specify both a control's colors and the fonts the control uses. The color tools are located in the central area of the dialog box:

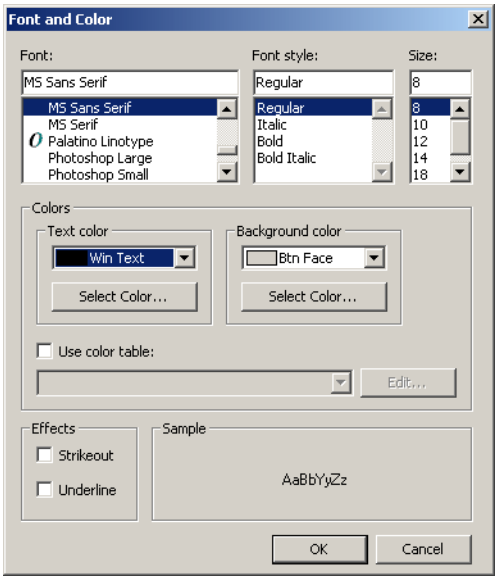


Figure 142. Font and Color dialog box

The *Sample* pane shows the effect of the color and font settings.

Working with Colors

Colors can be fixed, or they can depend on either or both of:

- The end user's *Windows* color setup.
- The color of the host screen characters in runtime.

Moreover, each control has two colors: one for text and one for background. These two colors can be set independently. However, if you want one color to depend on the corresponding host screen color and the other color to be independent of the corresponding host screen color, a particular procedure—detailed in "Combining Host Dependent and Host Independent Colors" on page 317—must be used.

Remember, controls added manually do not have a color table mechanism available.

Fixed Colors

Fixed colors do not depend on either the user's *Windows* settings or the color of the runtime host screen.

To specify a fixed color mechanism for either text or background:

1. In the *Font and Color* dialog box make sure the *Use color table* checkbox is cleared.
2. From the desired *Text/Background color* combo box select one of the supplied fixed colors:

Figure 143. Text/Background color combo box

Or click the *Select Color* button and from the *Color* dialog box select a color or define and select a custom color:

Figure 144. Color dialog box

3. Click *OK*.

Colors Depending on the User's Windows Setup

You can set a color to be the same as a system element of the end user's *Windows* setup. In this case, if the end user changes the color of that *Windows* system element, the corresponding color on the control also changes.

The *Windows* elements you select are fixed, and cannot be changed by the end user.

To make either the text or background color the same as a *Windows* system element:

1. In the *Font and Color* dialog box make sure the *Use color table* checkbox is cleared.
2. From the desired *Text/Background color* combo box select one of the *Windows* system elements. See the following section for a list of *Windows* system elements.
3. Click *OK*.

Table of Windows System Elements

Table 14 shows the value and the meaning of each of the items in the *Windows* system colors. These are the elements you can choose from the *Text/Background color* combo boxes.

Table 14. Table of windows system elements (Sheet 1 of 2)

Value	Description
SB Gray	Scroll-bar gray area.
Desktop	Desktop.
Act Caption	Active window title.
InAct Caption	Inactive window title.
Menu Back	Menu background.
Win Back	Window background.
Win Frame	Window frame.

Table 14. Table of windows system elements (Sheet 2 of 2)

Value	Description
Menu Text	Text in menus.
Win Text	Text in Windows
Caption Text	Active window title.
Act Border	Active window border.
Inact Border	Inactive window border.
MDI Back	Background color of multiple document interface (MDI) applications.
Select Back	Background of selected item in a control.
Select Text	Text in title bar, size button, scroll-bar arrow button.
Btn Face	Face shading on push buttons.
Btn Shade	Edge shading on push buttons.
Grayed Text	Grayed (dimmed) text. This color is zero if the current display driver does not support solid gray.
Btn Text	Text on push buttons.
Btn Select	Selected button in a control.
Special	Allows you to define a special color.
Bin Select	Text of selected item in a control.

Remarks about Window System Colors

- If the two elements you have selected for the control's text and the control's background happen to be set to the same color on the end user's machine, the text on the control will not be legible. Consequently, you should choose elements that are intended to be paired, such as WinText/WinBack or MenuText/MenuBack, as it is not likely that these elements will have the same color on any given system.
- You may wish to include with your application's documentation a remark that the application's colors change with the color scheme on the platform running the Application.

Color Tables

A color table sets the color of the control's text according to the runtime color of the host characters recognized by the Pattern Definition that generates the control. The control's background color is set according to the background color of these host characters.

The color itself can be either fixed or a *Windows* system element. The result is that instead of choosing one color mechanism for the control's text and another color mechanism for the control's background, you choose a different text color mechanism for each underlying host character color and you choose a different background color mechanism for each underlying host background color.

Note: To use a color table, in the control's *Manager* tab enable runtime data flow from the host to the manager and from the manager to the window. You must also enable the color table mechanism in the runtime environment. See "Selecting the Font" on page 318.

To specify a color table mechanism:

1 In the *Font and Color* dialog box set the *Use color table* check box.

Figure 145. Use color table check box

2 From the drop down list select a color table. If this color table already meets your requirements click *OK*. Otherwise continue with step 3.
3 Click the *Edit* button. The *Color Table* dialog box opens:

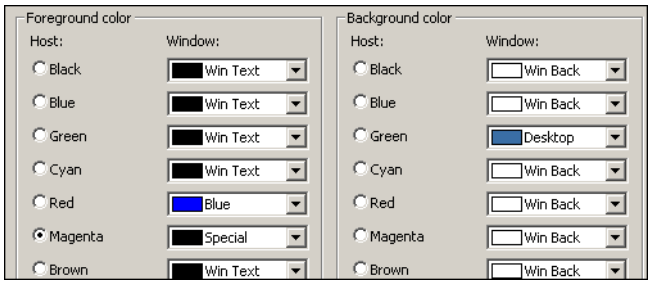


Figure 146. Color Table dialog box

4 Set the color table. You can select an existing color table, modify an existing table, or create a new color table. Follow the procedure in the corresponding section below.

Selecting an Existing Color Table

To select an existing color table:

1 In the *Color Table* dialog box, from the *Table name* combo box select the name of the color table you wish to use.

2 Click *OK*. The *Color Table* dialog box closes.

3 In the *Font and Color* dialog box click *OK*.

Modifying an Existing Color Table

To modify an existing color table:

1 In the *Color Table* dialog box, from the *Table name* combo box select the name of the color table you wish to modify.

2 Change settings as desired. See *Color Table Settings* below.

3 Click *OK*. The *Color Table* dialog box closes.

4 In the *Font and Color* dialog box click *OK*.

Creating a New Color Table

To create a new color table:

1 In the *Color Table* dialog box, in the *Table name* combo box type the name of the new color table you wish to create. If you want to base the new table on an existing table, select the existing table and then overwrite the name.

2 Change settings as desired. See *Color Table Settings* below.

3 Click *OK*. The *Color Table* dialog box closes.

4 In the *Font and Color* dialog box click *OK*.

Color Table Settings

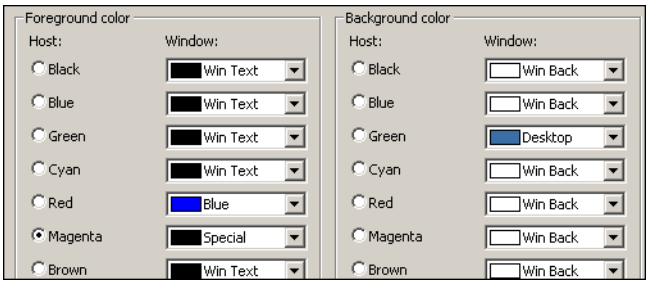
The color table sets the control colors according to the color of the host characters recognized by the Pattern Definition associated with the control. For every possible host character color you can set a color for the control's text. For every possible host background color you can set a color for the control background.

The actual operating procedures for setting character colors and background colors are identical.

To associate a host color with a window color:

1. In the *Color Table* dialog box, identify the host color of interest. For host characters, this is a color in the *Host* column of the *Foreground color* group. For host backgrounds, this is a color in the *Host* column of the *Background color* group.
2. In the list box opposite the host color select a color for the control. There are three types of choices you can make:
   - You can select a fixed color. You choose the corresponding colored bar from the list box.
   - You can select the color of a *Windows* system element. You choose the corresponding element from the list box. See "Colors Depending on the User's Windows Setup" on page 312 for an explanation of the different *Windows* system elements.
   - You can select a color from the *Color* box. To do this, set the radio button beside the host color of interest and choose *Special* from the list box. The *Special Color* button is enabled. Click the *Special Color* button and choose a color in the *Color* dialog box:
3. The color you choose can be a *Basic* color, an existing *Custom* color, or a new custom color that you *Define*.
4. When you are satisfied with your choices, click *OK*.

Example 46. Color table settings



From the settings in the example you can see that:

- In runtime, when the host characters recognized by the Pattern Definition underlying the control are red, the text on the control is blue.

- In runtime, when the host characters recognized by the Pattern Definition underlying the control are magenta, the text on the control is a color specified in the *Color* dialog box.
- In runtime, when the host characters recognized by the Pattern Definition underlying the control are any color other than red or magenta, the text on the control is whatever color is set as "Win Text" in the end user's *Windows* system setup.
- In runtime, when the host background color of the host characters recognized by the Pattern Definition underlying the control is green, the background color on the control is whatever color is set as the "Desktop" color in the end user's *Windows* system setup.
- In runtime, when the host background color of the host characters recognized by the Pattern Definition underlying the control is any color other than green, the background color on the control is whatever color is set as "Win Back" in the end user's *Windows* system setup.

Note: The above example does not reflect a suggested color table. In fact, it is not good design practice to mix fixed colors with Windows system-element colors on the same control. The reason is that a fixed text color may not blend properly with the color your end user has chosen for the Windows element governing the background color.

Combining Host Dependent and Host Independent Colors

The *Color and Font* dialog box requires that either both the text and background be set from a color table or neither text and background be set from a color table. If you want, say, the text to be dependent on the host character color but the background to always be the Desktop color, use a color table in which all the background colors are set to the Desktop element. This effectively makes the control's background color independent of the host background.

Selecting the Font

ACE enables you to set the fonts and characteristics of text displayed in runtime in the *Font and Color* dialog box:

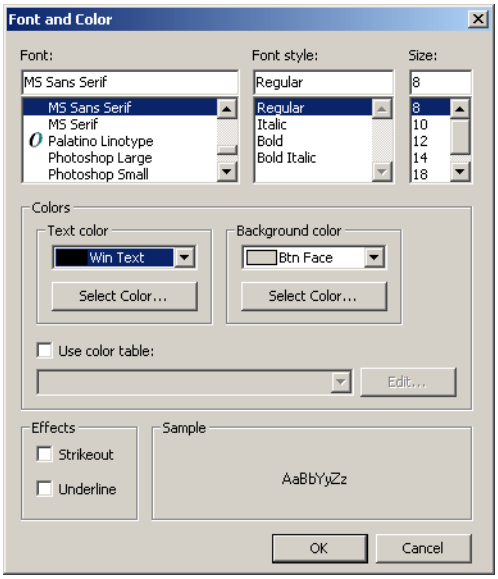


Figure 147. Font and Color dialog box

1. Select the font (typeface), font style and character size in points from the list boxes displayed at the top of the *Font and Color* dialog box.
2. Set the *Strikeout* checkbox to draw a horizontal line through the midpoint of the text.
3. Set the *Underline* checkbox to draw a horizontal line under the text.

Defining Fonts to Produce a Consistent GUI

ACE's advanced features make extensive use of bitmaps and thus provide a high quality GUI, a GUI that can be presented consistently across varying display drivers and screen resolutions. However ACE generated windows that combine bitmaps and system fonts, when displayed on a system other than that on which they were converted, can result in a GUI with disproportionate text. This undesirable result can easily be avoided by explicitly defining the fonts during conversion as non-system fonts.

The problem outlined above occurs because a discrepancy exists between the methods used in *Windows* to measure the size and position of bitmaps and system fonts.

Measuring Changes in Control and Font Sizes

Windows can determine control and font size in a window in one of two ways: either by using absolute pixel size as a means of measurement, or by using a theoretical measure called a dialog unit. Let's compare the two briefly.

Absolute Pixels

If object size is measured by absolute pixels then an object's size will be consistent on all display drivers and screen resolutions. For example if a rectangular object is 20 pixels in length and 10 pixels high then no matter what display driver or screen resolution is used the rectangle will still be 20 pixels long and 10 pixels high. However, the relative position and size of the object with respect to the screen may change. The size of the object is then independent of the display driver and screen resolution.

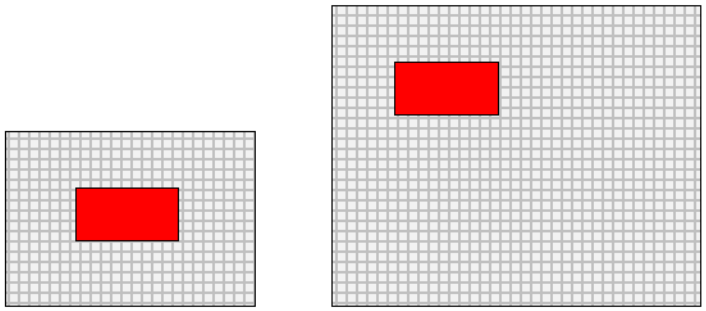


Figure 148. Absolute pixels

Dialog Units

A dialog unit is a theoretical measure based on the average size of the default system font. Since the default system font is dependent on the display driver and screen resolution, the dialog unit as a measure is also dependent on these factors. Therefore, when an object is displayed on different display drivers and screen resolutions, the object changes size in proportion to an equivalent change in dialog units.

In this case, the relative size and position of the object with respect to the screen remain constant, assuming the display driver suits the actual resolution and screen size.

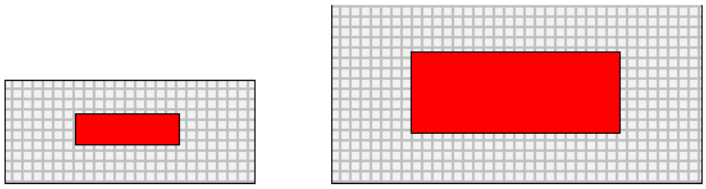


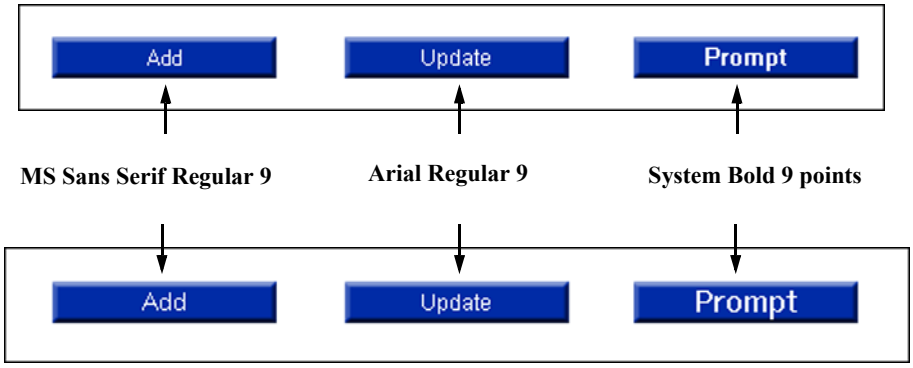
Figure 149. Dialog units

Combining Items Measured in Pixels and Dialog Units

Combining items measured in pixels and items measured in dialog units gives undesirable results when the object is presented on a different display driver or with a different screen resolution to that with which it was created. Items measured in absolute pixels maintain their exact dimensions but are positioned differently on the screen. Items measured in dialog units maintain their position but change size. This problem is particularly obvious in buttons where the text is defined as a system font. The button is a bitmap so it is measured in pixels, whereas the text, being a system font, is measured in dialog units.

How This Affects the Appearance of Your GUI in ACE

The following two illustrations each show the same series of buttons. The first row shows the buttons as they appear when the monitor's properties are set in the control panel to display small fonts, and the second row shows the buttons as they appear when set to display large fonts. Each of the buttons in the first figure is labeled to indicate the font, font style and point size used for the text.



- The *Prompt* button shows the result where an object that contains items measured in pixels and dialog units is displayed on a different display driver. The text uses a system font where the style and point size have been explicitly defined. The text is disproportionately large compared to the button when displayed with large fonts, since the font is measured in dialog units.

- The *Add* button uses text that has been explicitly defined in the Representation Definition. The result is improved since MS Sans Serif is a non-system font. Nevertheless the text is not identical under both conditions. While MS Sans Serif is a non-system font it is not a TrueType font. As a result, if the exact font characteristics are not resident in the new system, *Windows* provides what it believes to be the closest available alternative. In this case a slightly larger point size.
- The *Update* button is the only button of the three where the text appearance remains constant when presented with both small fonts and large fonts. This results from several facts:
  - Arial is a non-system font
  - Arial is a TrueType font
  - The font was explicitly defined in the Representation Definition

As long as the font is resident on the new system, any TrueType font whose name, style, and point size have been explicitly defined in the Representation Definition will provide a consistent look and feel across all display drivers and screen resolutions.

Suggested Fonts

Fixed fonts	Courier New
Serif	Times New Roman
Sans Serif	Arial

Chapter 12. Tabbing Order Modifications

When pressing the *Tab* key in runtime, the input focus moves from one control to the next, without having to use the mouse. This movement is defined by the tabbing order of the controls in the Subapplication. ACE has several ways to calculate the tabbing order of the controls in a Subapplication, and you can choose the one that suits the needs of your Application.

You can make local changes to the tabbing order in case it is not optimal. The tabbing order of a control can be manipulated in such a way that it becomes the first or last tab, or is reached before or after a specific control. You can also make changes to the tabbing order in a selected group of controls.

This chapter includes:

- The Tabbing Order
- Modifying Tabbing Order
- Tabbing Order in Subwindows and Tab Controls

Setting Tabbing Order

In runtime, "tabbing" relates to the ability to press the *Tab* key and advance the input focus from one control to another, instead of using the mouse.

ACE automatically sets the tabbing order in the GUI window, however, in certain situations it is advantageous to modify that order.

The Tabbing Order

How ACE Sets the Tabbing Order

ACE automatically assigns a tabbing order for all controls on a window, even for controls such as Statics that cannot receive input focus during runtime. There are three mechanisms by which ACE can assign the tabbing order to a control:

- *By GUI location:* this mechanism sets the control's tabbing order according to the position of the control's top left corner on the GUI window. If the top left corners of two controls lie on the same horizontal line, the left-most control is first in the tabbing order.
- *By host screen location:* this mechanism sets the control's tabbing order according to the position of the top left corner on the host screen of the area

that corresponds to the control. Tabbing order for a control that is not related to a host screen area (manually-added control) is calculated by the control's relative location on the window.

- *Smart:* this mechanism sets the control's tabbing order according to the position of the control's top left corner on the GUI window (like the By GUI location mechanism), except for one case: when the control is positioned to the left of a control whose top is higher and whose bottom is lower, like this:

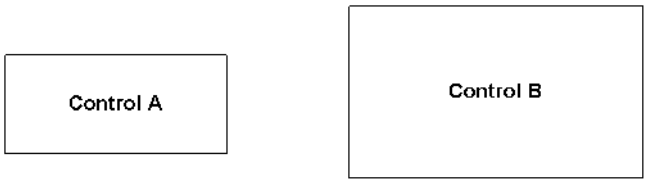


Figure 150. Smart tabbing order mechanism

In the figure above Control A precedes Control B in the tabbing order. The Smart mechanism is ACE's default mechanism for assigning the tabbing order.

Automatic Recalculation of the Tabbing Order

ACE calculates the tabbing order of the controls when a Subapplication is entered.

In addition, every time the following events take place, ACE automatically recalculates the tabbing order on the basis of the specified ordering mechanism, or the default mechanism if no other is specified:

- The Subapplication is saved.
- *Apply Design Changes* is performed.
- Test View is entered.
- Tabbing Order mode is entered.

Automatic recalculation only takes place if you have not yet performed a local change in the tabbing order, such as a Make First modification, or a "drag and drop" modification.

How to Specify the Tabbing Order Mechanism that ACE Uses

You can specify which of the three mechanisms ACE should use to assign tabbing order. This is done in the *Windows Options* dialog box.

To specify which mechanism ACE uses to set the tabbing order:

1 In Design View or Test View, from the *Options* menu select *Window Options*.

2 In the *Window Options* dialog box, go to the *Window Algorithms* tab.

3 Select your choice from the *Tabbing order algorithm* drop-down list.

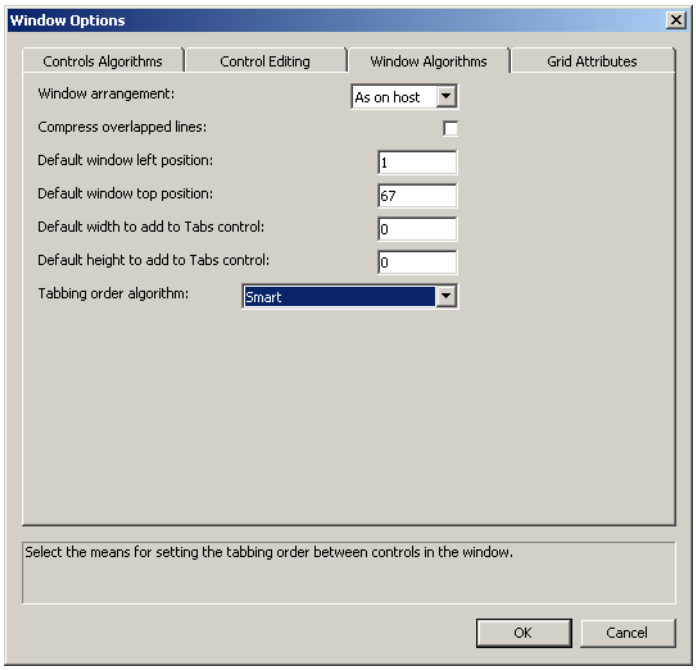
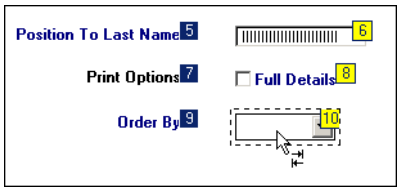


Figure 151. Window Algorithms tab

Viewing the Tabbing Order

You can see the tabbing order in Test View and in Design View.

- In either View, press the *Tab* key repeatedly. This lets you scroll through the controls tabbing order.
- In Modify Tabbing Order mode (see next section for details), press the *Tab* key repeatedly. An animated frame appears around each control in the tabbing order.



- In both Views and in Modify Tabbing Order mode, pressing *Shift+Tab* scrolls through the tabs in reverse order.

Modifying Tabbing Order

Modifications to the tabbing order are made in a special mode called the Modify Tabbing Order mode. This mode is accessed from Design View.

The various operations you can perform in Modify Tabbing Order mode are detailed in this section.

They include:

- Making a control first or last in the tabbing order.
- Changing the tabbing order of a group of controls.
- Resetting the tabbing order to default.
- Resetting the tabbing order of a group of controls.

Entering Modify Tabbing Order Mode

To enter Modify Tabbing Order mode:

1 In Design View, select *Design >Tabbing Order > Modify Tabbing Order*.

Control Display in Modify Tabbing Order Mode

When you enter Modify Tabbing Order mode, all other operations in Design View are disabled, and the tabbing order of the controls in the Subapplication is displayed as a number in a small box to the right of each control.

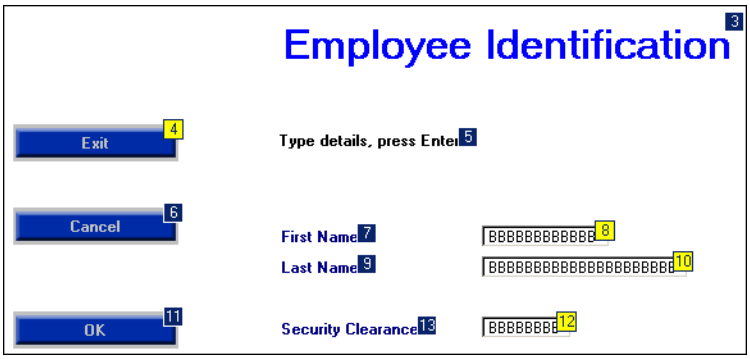


Figure 152. Setting Tabbing order in design view

The box's background color indicates whether or not the focus can be moved to the control using tabbing in runtime, that is, whether pressing the *Tab* key can move input focus to the control, or not.

Tabable and Non-tabable Controls

Controls with a dark number on a yellow background are tabable.

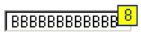


Figure 153. Tabable controls

Controls with a white number on a blue background are not tabable.

Figure 154. Not tabable controls

You can make a control tabable by checking the *Tab stop* checkbox in the *Style* tab of the control's *Component* dialog box.

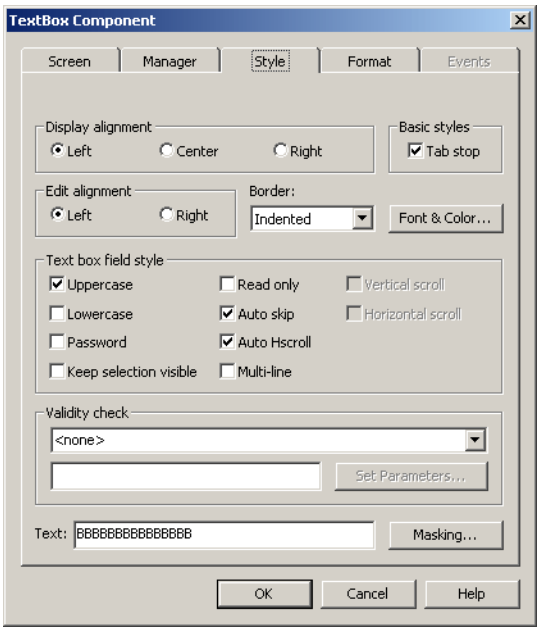


Figure 155. Tab stop checkbox in the Style tab

When a control's *Tab stop* check box is cleared, you cannot tab to that control in runtime.

Runtime Behavior

Pressing the *Tab* key causes the input focus to scroll through the tabable controls, in the order shown in Modify Tabbing Order mode.

> **Note:** Some controls, such as the Static control, cannot receive input focus. When *Tab stop* is checked in such a control it has no effect. In runtime, the control is skipped when tabbing and input focus moves to the next tabable control.

Local Tabbing Order Modifications

Tabbing order can be modified by:

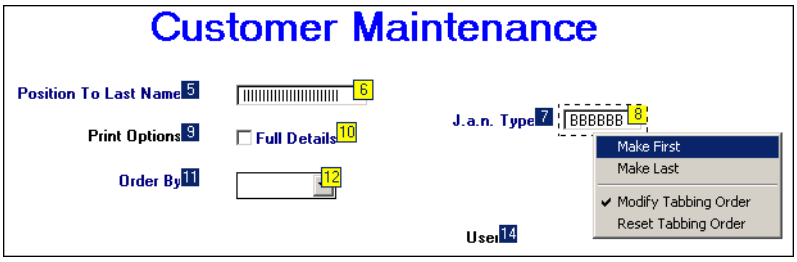
- Making a specific control first or last in tabbing order.
- Using a “drag and drop” mechanism to modify the tabbing order of a pair of controls.
- Resetting the tabbing order in the window or in part of the window.

Making a Control First or Last

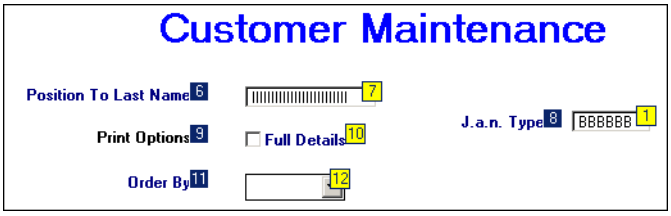
To make a control first or last in the tabbing order:

1 In Modify Tabbing Order mode, put the mouse pointer on the control. An animated frame appears around the control.

2 Right-click and select *Make First* or *Make Last* from the shortcut menu.



In the figure shown above, the selected control is being set to be first in the tabbing order. After this change, ACE recalculates the tabbing order for the remaining controls according to the converter’s ordering mechanism.



Modifying the Tabbing Order of a Pair of Controls

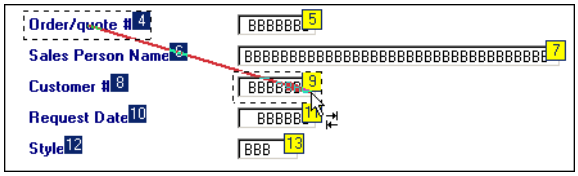
To change the tabbing order of a pair of controls:

1 In Modify Tabbing Order mode, put the mouse pointer on a control. An animated frame appears around the control.

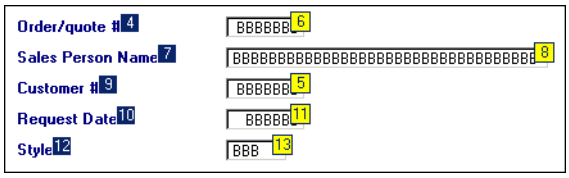
2 Click on the control and drag. A red arrow with its base on the center of the control’s and its tip on the mouse pointer appears.

3 Drag the pointer to the control you want to be next in the tabbing order. The animated frame appears around this second control.

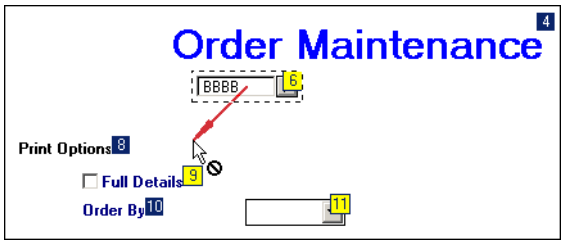
Note the two-arrow icon near the pointer.



4 Release the mouse button. The tabbing order of the second control now follows that of the first control.



5 Repeat steps 1 to 4 for any pair of controls whose order you wish to change.



Note: When you drag the red arrow over the window, if the arrow's tip is over an area that does not contain a control, or over a control that cannot receive a tabbing order number, a small round icon appears near the pointer's head.

Resetting the Tabbing Order for the Whole Window

This restores the default tabbing order. ACE applies the converter's tab ordering mechanism to all controls on the window. Any local modifications made to the tabbing order are lost.

To reset the tabbing order:

1 In Modify Tabbing Order mode, place the pointer anywhere on the window.

2 Right-click and select *Reset Tabbing Order* from the shortcut menu.

-or-

In Design View, select *Design* > *Tabbing Order* > *Reset Tabbing Order*.

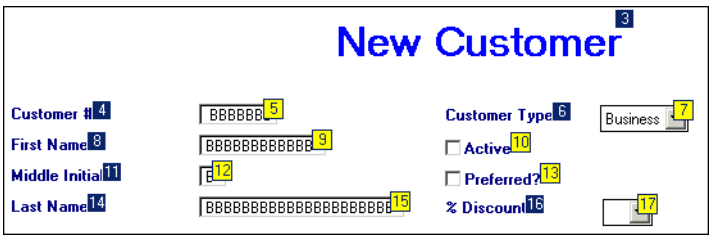
Resetting the Tabbing Order in a Selected Rectangle

This feature uses the converter's specified tab ordering mechanism, or the default mechanism if not specified otherwise, to change the tabbing order of a group of manually selected controls.

The feature is designed to arrange controls in a logical tabbing order, or to arrange a group of controls so that they have a consecutive tabbing order within the group. What this feature does, in effect, is to let you set the tabbing order of the controls in vertically-arranged groups, instead of the default horizontal arrangement.

Consider the following situation. In the window below the controls are arranged in a group on the left that contains personal information, and a group on the right that contains business information.

It would be logical to tab through the fields of one group first and then through the fields of the second group, but this is not the default tabbing order.



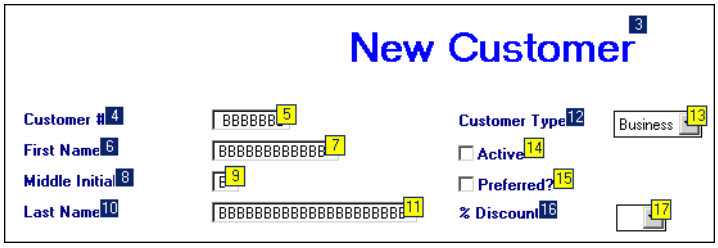
This situation can be changed by selecting the left group of controls and resetting the tabbing order.

Resetting the tabbing order in a selected rectangle applies the converter's ordering mechanism to controls in the selected area. In practice, this means the controls within that selected group are given consecutive tabbing order. The remaining controls are ordered according to the chosen tabbing order mechanism.

To reset tabbing order in a selected rectangle:

1 In Modify Tabbing Order mode, click and drag a rectangle across the group of controls you want to reorder. Click *Yes* in the confirmation message box.

2 The controls in the selected group are given consecutive tabbing order, starting with the control with the lowest tabbing order.



Returning to Design View

To continue with the design changes in the Subapplication, you must exit Modify Tabbing Order mode.

Use one of three ways to leave Modifying Tabbing Order mode:

- Press the *Escape* or *Enter* key.
- Select *Design >Tabbing Order > Modify Tabbing Order*.
- Put the pointer on any area of the window, click the right mouse button, and select *Modify Tabbing Order* from the shortcut menu.

Tabbing Order in Subwindows and Tab Controls

The tabbing order feature is supported in child subwindows and tab controls, but not in popup subwindows.

The following points relate to special aspects of the feature in tabs/subwindows:

- The tabbing order of controls inside a subwindow or a tab is unrelated to the tabbing order in the main window, and is internal to the tab or the subwindow.
- Since tabs and subwindows are in themselves controls, they receive a tabbing order of their own in the main window.

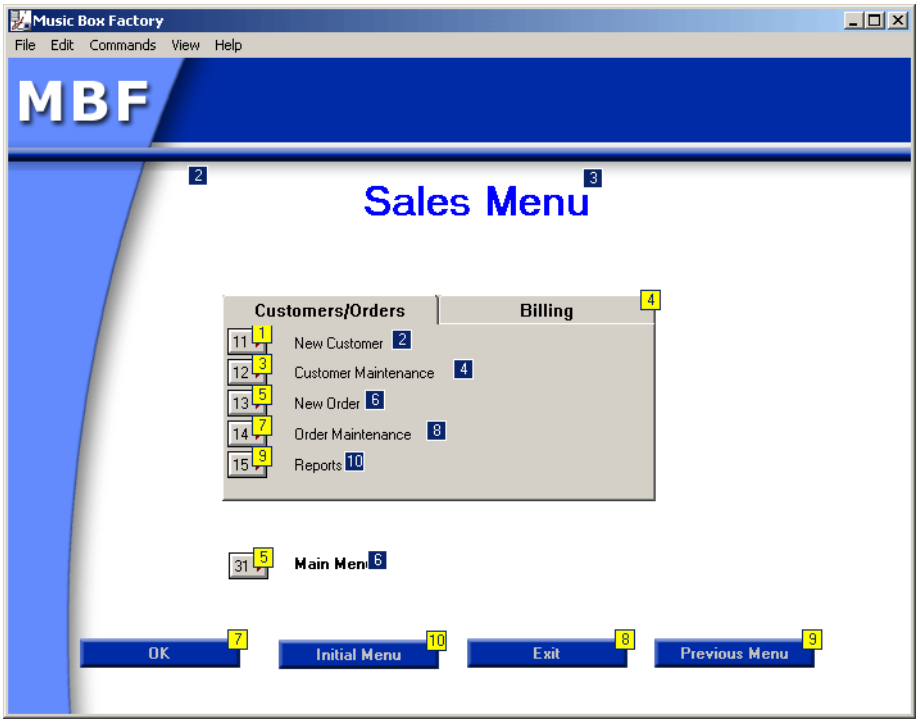


Figure 156. Tabbing order in subwindows

- During runtime you tab through controls in the main window until the tab control or the subwindow receives the input focus. Then controls in the subwindow or the currently selected tab receive input focus according to their internal tabbing order. Once all controls have received input focus, the input focus reverts to the main window.

Limitations

- It is impossible to design the tabbing order in such a way as to make the input focus "jump" from a control on a tab/subwindow to a control on the main window and then "jump" back to another control on the tab/subwindow.
- Resetting the tabbing order in a selected rectangle that includes controls from the main window and from a tab/subwindow has no effect on the tabbing order of the controls in the tab/subwindow, but it does modify the tabbing order of the tab/subwindow itself on the main window.
- When moving controls between subwindows after making tab order changes, the converter puts the controls at the end of the tab order sequence in the target window. It does not keep the tab order as it was in the dragged controls group.
- When the `FrameZOrderBottom` setting in the <applname>.INI file is set to the default value 1, the runtime ignores all tabbing order modifications made to dynamic groups and they receive input focus at the end of the tabbing order.
- When using the *By host screen location* mechanism, the control tabbing order may seem to be inconsistent, since it is set by the order of fields on the host

screen, not by the control's location on the GUI window. The tabbing order for a manually-added control may also seem to be inconsistent, especially when the other controls are relocated on the GUI, since it is calculated by the control's relative location on the window.
To overcome this problem, manually modify the tabbing order in such a way as to add consistency, but only do so after you do not intend to move any more controls on the GUI.

Chapter 13. Methods: Attaching Functionality to the GUI Window

ACE is supplied with predefined methods that handle general aspects of data flow and interaction between the host and the GUI application. Writing new methods or modifying existing methods provide a way to address all interaction between the host application and the GUI window, granting a sophisticated level of functional customization.

This chapter includes the following topics:

- What are Methods?
- Method Types and Method Levels
- Accessing Methods
- Linking User-Triggered Methods to Controls
- Writing Methods

What are Methods?

Methods are routines that give functionality to the GUI application, manage the behavior of the host application and the GUI application, and synchronize between the host screen and the GUI.

Methods are short programs, containing a few lines of code, that instruct ACE to perform actions in a certain order. Methods are linked to triggers. When the trigger is activated, the instructions contained in the method are executed in the order in which they appear in the method definition.

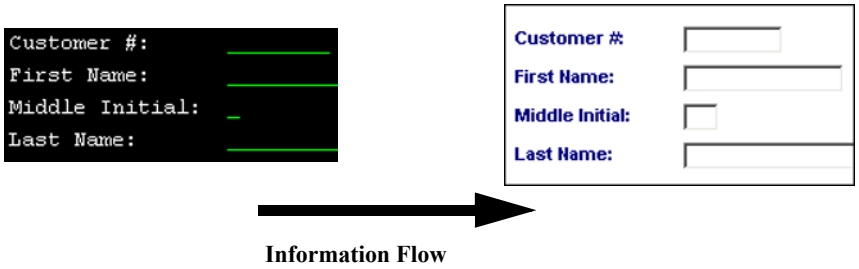
Basic functionality regulated by the use of methods includes:

- Updating fields on the window and on the screen.
- Regulating the flow of data between the host and the window.
- Executing a set of instructions to be performed:
  - As a result of user interaction with the GUI window.
  - When entering or leaving a screen.
  - When a specific event takes place on the host or the window.

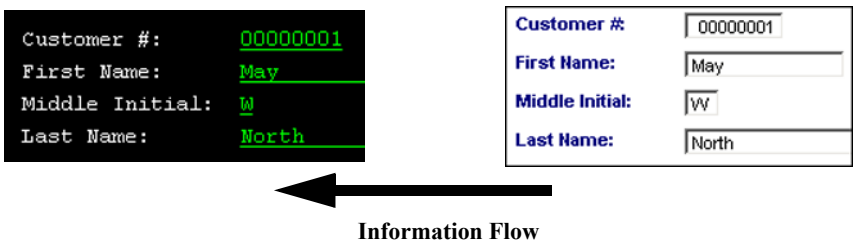
Example 47. Methods

In the GUI window, four input fields are filled in with data. The *Enter* key is then pressed to update the host and move to the next screen.

1. The runtime recognizes the screen and displays the corresponding window.



2. The end user types data in the GUI window input fields.
3. The end user clicks *OK* triggering the *Enter* method, which is attached to the *onClick* event associated with the *OK* button. The method instructs ACE to update the fields on the host with the information typed in the GUI window.



4. The method also informs the host that *Enter* was pressed. This causes the host to move to the next screen. The runtime recognizes the new screen and displays the corresponding window.

Method Types and Method Levels

There are three types of methods in ACE:

- *User-Triggered methods* - Activated by user interaction during the runtime.
- *System-Triggered methods* - Activated when system-dependent events occur during runtime.
- *Message Handling Methods* - Attached to host screen messages. They enable converting messages into a GUI characteristic.

Note: This chapter discusses User-Triggered and System-Triggered methods only. For information on Message Handling methods, see the chapter entitled "Message Handling"in *JIS Interface Server: Advanced Topics*.

Both User-Triggered methods and System-Triggered methods are divided into levels:

- *General Methods* - Application-level methods which are available throughout an Application.
- *Current Library Methods* - Library-level methods which are available only within a specific library.
- *Current Subapplication Methods* - Methods which are available only within a specific Subapplication.

General methods, Current Library methods, and Current Subapplication methods work hierarchically:

- Current Subapplication methods override Current Library methods
- Current Subapplication methods override General methods
- Current Library methods override General methods

Example 48. Method types and method levels

When a specific Subapplication contains a Current Subapplication method and a General method with the same name, the Current Subapplication method is executed.

User-Triggered Methods

User-Triggered methods are activated by the end-user's direct interaction with a control on the GUI window. User-Triggered methods are linked to the *onClick* event that is associated with controls.

ACE provides default User-Triggered methods attached to frequently occurring GUI controls. There are full provisions for modifying predefined User-Triggered methods and writing new ones for enhanced Application functionality.

GUI controls that function as method triggers include:

- Buttons
- Menu items
- Right Mouse Button (RMB) menu items
- Accelerator keys
- Table rows

Examples of User-Triggered Methods

Following are a few examples of User-Triggered Methods provided with ACE. These are not detailed examinations of the instruction lines that comprise each method, but a general explanation of what the method does.

Enter Method

The *Enter* method is a predefined method that is automatically attached by the converter to the *OK* button. This method is linked to the *OK* button in every Subapplication.

Clicking the *OK* button in runtime activates the *Enter* Method.

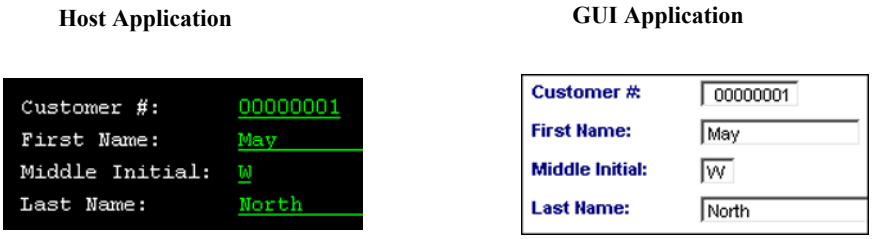


Figure 157. Enter method

During runtime, pressing the *OK* button on the GUI window activates the linked *Enter* method and runs the following routines:

- Updates all fields from the GUI window to the screen.
- Presses the *Enter* key on the host.
- Displays the GUI window corresponding to the new screen on the host.

F_Key Method

The *F_Key* method is a predefined method automatically attached by the converter to buttons that represent *FKey* Pattern Definitions, such as the *Exit*, *Prompt*, and *Cancel* buttons. ACE links the method in every Subapplication that contains buttons representing function keys.

During runtime, pressing these buttons on the GUI window activates the linked *F_Key* method and runs the following routines:

- Updates all fields from the GUI window to the screen.
- Determines which function key is represented by the button that was pressed.
- Presses the correct function key on the host.
- Displays the GUI window corresponding to the new screen on the host.

SelectMenuOption Method

The *SelectMenuOption* method is a predefined method automatically attached by the converter to buttons that represent menu options, such as the numbered buttons in the picture below.

ACE links the method in every Subapplication that contains buttons representing a menu option.

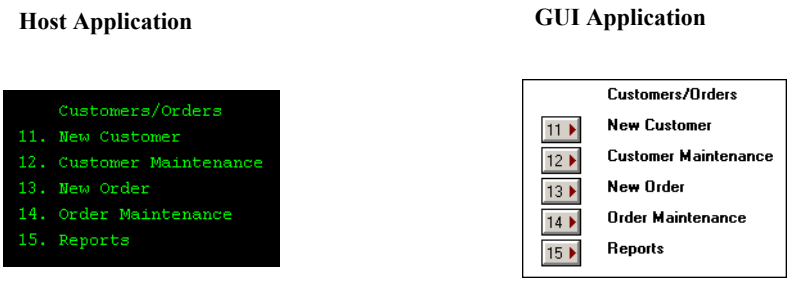


Figure 158. SelectMenuOption method

During runtime, pressing one of the menu option buttons on the GUI window triggers the *SelectMenuOption* method and runs the following routines:

- Updates all fields from the GUI window to the screen.
- Determines which menu option is represented by the button that was pressed.
- Enters the correct number in the menu selection field in the host and presses *Enter*.
- Displays the GUI window corresponding to the new screen on the host.

System-Triggered Methods

System-Triggered Methods are linked to system-dependent events. These system-dependent events exist at the Application or Subapplication level. System-Triggered Methods do not depend upon direct interaction of the end user to trigger the method.

A few examples of System-Triggered Methods are:

- *UserInitSubApplication* - Activated when entering a Subapplication for the first time.
- *UserCloseSubApplWindow* - Activated when the end user closes a window.

System-Triggered Methods are predefined Methods hard-coded in the ACE source code. System-Triggered Methods can be modified by the Application developer, but cannot be deleted.

Example of a System-Triggered Method

UserInitBeforeFirstSubAppl is a System-Triggered Method that is activated once on initial entry to an Application. The default behavior of the method is to do nothing.

The Application developer can use this method to perform specific preparatory operations, at the Application level, before any Subapplications are processed.

For example, this method can be used to read a specific setting from an INI file or write one to it. This requires the Application developer to add the necessary instructions to the *UserInitBeforeFirstSubAppl* method.

Events and System-Triggered Methods

System-Triggered Methods are attached to events. The following diagram shows the main events and which methods they activate:

Figure 159. Flow of events and system-triggered methods

The flow of events in a running Application is illustrated from top to bottom. Each rectangle contains the name of an event. Below the rectangle are the names of the System-Triggered Methods that are attached to this event.

The events in the white rectangles with a dotted outline always occur during the course of an Application or Subapplication.

The events in the gray rectangles with a solid border, in the center of the diagram, occur only under specific circumstances or when the relevant control is used.

The events occur in the following order:

1. Entering an Application
2. Sending a host screen to the server
3. Entering a Subapplication
4. Optional events related to message handling, host refresh, screen and control navigation, and host help request
5. Exiting the Subapplication
6. Exiting the Application

The first three events necessarily happen; their associated methods are always activated.

The optional events occur after entering a Subapplication. The activation of the attached Methods in the course of a Subapplication depends on the type of Subapplication, controls used, and host actions taking place. Any number of the optional events may occur, in no particular order. It is also possible that none of these events occur, in which case the next event that happens, after entering the Subapplication, is exiting the Subapplication.

The last two events always happen; their associated methods are always activated.

Note: When a window refreshes, the *UserRefreshSubApplication* method is activated immediately before the refresh and the *UserAfterRefreshSubApplication* method is activated immediately after the refresh.

System-Triggered Methods Activated in Every Application and Subapplication

Most of these events occur in every Subapplication, activating the linked methods:

Table 15. Activated in every application and subapplication (Sheet 1 of 2)

Method	Description
UserInitApplication	The first method activated when entering an Application, before establishing a connection to the host.

Table 15. Activated in every application and subapplication (Sheet 2 of 2)

Method	Description
UserInitBeforeFirstSubAppl	Activated when entering an Application, before the first Subapplication is shown.
UserPreUpdateInitSubAppl	Activated every time a Subapplication is entered but before information is updated from the screen to the manager. This method is executed before *UserInitSubApplication* and *UserSkipSubApplication*.
UserInitSubApplication	Activated upon every initial entry to a Subapplication that was defined as a regular Subapplication in the *New Subapplication* wizard.
UserDestroySubApplication	Activated every time you exit a Subapplication.
UserCloseSubApplWindow	Activated when clicking the *Close* button to exit a Subapplication or a popup window
UserShouldCloseSubApplWindow	Activated under special circumstances, when exiting a Subapplication or a popup window.
UserDestroyAllSubAppl **UserShouldCloseSubApplWindow**	Activated when clicking the *Close* button to exit an Application.

System-Triggered Methods Activated Under Specific Conditions

The System-Triggered Methods in Table 16 are linked to events that do not necessarily occur in each Subapplication. These methods are specific to certain types of Subapplications, controls, or host actions.

Table 16. Activated under specific conditions (Sheet 1 of 2)

Method	Description
UserShouldWindowBeBuilt **UserSkipSubApplication**	Activated upon entry to a Subapplication that was defined as either *Never Display Window* or *Conditionally Display Window.*
UserRefreshSubApplication	When a host screen change causes the runtime to refresh the GUI window, this method is activated before the refresh.
UserAfterRefreshSubApplication	When a host screen change causes the runtime to refresh the GUI window, this method is activated after the refresh.
PageUp, PageDown, AfterPageUpDown	Activated upon clicking in a table's scrolling bar.
GetToTopOfList, GetToBottomOfList	Activated upon clicking the *Top* and *Bottom* buttons on the GUI.
TableChangedSelection	Activated by changing the selection on a table. Other events also activate this method. See the method's description in "Methods: System-Triggered Methods List" on page 407.
UserMoveToDependentScreen	Only activated in Many-to-One principal Subapplications, when a Dependent screen is reached on the host.
UserIsRealMessage	Activated when the runtime identifies a message on the host.
UserHostMessageHelpRequest **UserMWIRequest**	Activated when clicking the *Help* button and the *Message waiting* button, respectively, on the DIL line (for AS/400 hosts only).

Table 16. Activated under specific conditions (Sheet 2 of 2)

Method	Description
UserAttentionRequest **UserHostHelpRequest**	Activated when selecting the *Attention* menu or Host help, respectively, in the runtime environment window (for AS/400 hosts only).
UserBeforeTabFolder Changed **UserAfterTabFolderChang ed**	Activated when moving the focus in a tab control to a different tab folder. *UserBeforeTabFolderChanged* is activated before the move and *UserAfterTabFolderChanged* is activated after the move.
UserBeforeTableAction, UserAfterTableAction	Activated before or after the processing of every table action. To get the action type, check the value of the user variable _internal_TableAction. Possible action types are: CellChanged, PageUp, PageDown, DoubleClick, PromptClick, MoveToTop, MoveToBottom.

General Methods

ACE is supplied with some predefined General methods that can be modified to suit an Application's specific requirements. New General methods can be written to solve functionality issues and enhance Applications.

General methods can be User-Triggered or System-Triggered.

General User-Triggered methods are linked to events that are associated with specific controls. General User-Triggered Methods are activated by user interaction with a control, such as clicking on a menu item. General Methods are linked to their triggers in the KnowledgeBase or locally in a Subapplication.

General System-Triggered Methods are linked to events that occur during runtime, such as closing a Subapplication window. General System-Triggered Methods are not dependent on user actions. The method-controlled behavior is repeated in every Subapplication when the appropriate event takes place in the runtime.

General User-Triggered Methods

OK buttons, FKey buttons, or menu items are control elements commonly encountered in Subapplications. In the KnowledgeBase, the Representation Definitions for these controls include a General User-Triggered Method that gives these controls their functionality.

Table 17 lists some common controls and their attached General User-Triggered Methods.

Table 17. General user-triggered methods

Control Type	Attached Method
OK button	*Enter* This method updates all fields from the window to the screen, presses the *Enter* key on the host and proceeds to the next screen.
Menu item	*SelectMenuOption* This method types the value of the menu selection option read from the GUI window, into the menu command field and presses the *Enter* key.
Function Key	*F_Key* This method presses a function key and proceeds to the next screen. The value of the function key is determined by the text on the screen.
List Command	*SelectListOption* This method types the value of the list selection option read from the GUI window, into the list selection column according to the selected lines in the Table control.
Prompt	*PromptControl* This method is attached to the prompt control. It presses *F4* with the cursor positioned in the correct field.

General System-Triggered Methods

To modify the behavior of a System-Triggered Method throughout all the Subapplications in an Application, modify the method at the general level.

Example 49. General system-triggered methods

To execute a set of instructions upon the entry of every Subapplication in the Application during the runtime, modify the *UserInitSubApplication* in the *General Methods* list.

System-Triggered Methods and their Use

For a full list of the available System-Triggered Methods and information on how to modify them see Chapter 14 - "Methods: System-Triggered Methods List" on page 407.

Current Subapplication Methods and Current Library Methods

Current Subapplication User-Triggered Methods are linked to their trigger inside specific Subapplications and are not shared by other Subapplications.

Current Library Methods are like current Subapplication Methods in that they are attached to their trigger inside specific libraries. They cannot be activated in other libraries.

There are no predefined Current Subapplication or Current Library Methods.

In some Subapplications a local variable is used or a unique control appears. In this case, writing a Current Subapplication Method is recommended.

For User-Triggered Methods, the controls that act as triggers for Current Subapplication Methods and Current Library Methods are the same as for General Methods.

When working with System-Triggered Methods, General Methods can be modified and used as Current Subapplication Methods. The modification is valid only in the specific Subapplication. In other Subapplications the method behaves like the default.

Example 50. Current subapplication and current library methods

To execute a set of instructions during runtime upon the entry to a specific Subapplication, modify the *UserInitSubApplication* in the *Current Subapplication Methods* list in that Subapplication.

Accessing Methods

Both User-Triggered and System-Triggered Methods can be accessed via a manager dialog box. The manager dialog box displays the currently existing methods in each level: General, Current Library and Current Subapplication, and enables you to perform various actions. The *User-Triggered Methods* manager and the *System-Triggered Methods* manager can be opened either from the *Design* menu in Design View or from the *Define* menu in the KnowledgeBase.

User-Triggered Methods Manager

ACE provides predefined General User-Triggered Methods, which can be accessed via the *User-Triggered Methods* manager. However, there are no predefined User-Triggered Methods at the Current Subapplication and Current Library levels. Therefore, these categories are empty in the *User-Triggered Methods* manager, unless such methods are written by the Application developer.

To access the *User-Triggered Methods* manager:

- In Design View, select *Design* > *User-Triggered Methods*
- In the KnowledgeBase, select *Define* > *User-Triggered Methods*

The following dialog box opens:

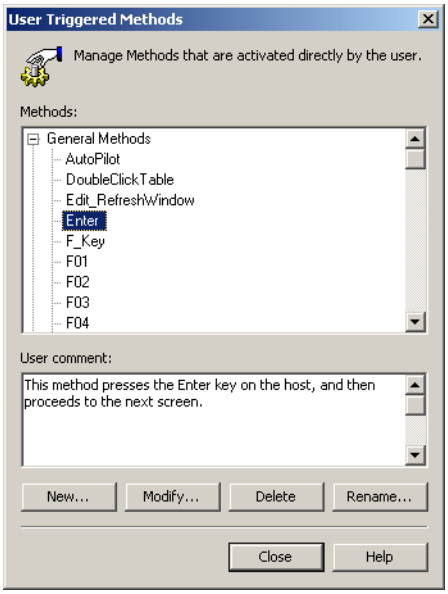


Figure 160. User-Triggered Methods dialog box

The editing options available in the *User-Triggered Methods* manager are:

New	Click to define a new method.
Modify	Click to modify the selected User-Triggered Method.
Delete	Only developer-written methods can be deleted. Methods supplied with ACE are protected and cannot be deleted.
Rename	Opens a dialog box with an edit field where the new name is typed.

Note: To search for a method, click the right mouse button on one of the elements in the methods tree. Select *Find* from the shortcut menu, and type the string to search for. To exit the *Find* dialog box, press the *Esc* key.

For more information, see “Method Operations: User-Triggered Methods” on page 363.

System-Triggered Methods Manager

ACE provides access to predefined System-Triggered Methods via the *System-Triggered Methods* manager. The methods are divided according to their level: General, Current Library and Current Subapplication.

All methods exist in each one of the three levels with the exception of:

- *UserInitApplication* - exists only at the General level because it is valid only at the Application level.
- *UserInitBeforeFirstSubAppl* - exists only at the General and Current Library levels, because it is not valid at the Subapplication level.

To access the System-Triggered Methods manager:

- In Design View, select *Design* > *System-Triggered Methods*
- In the KnowledgeBase, select *Define* > *System-Triggered Methods*

The following dialog box opens:

A red check mark indicates that the predefined functionality of this System-Triggered Method has been modified within the

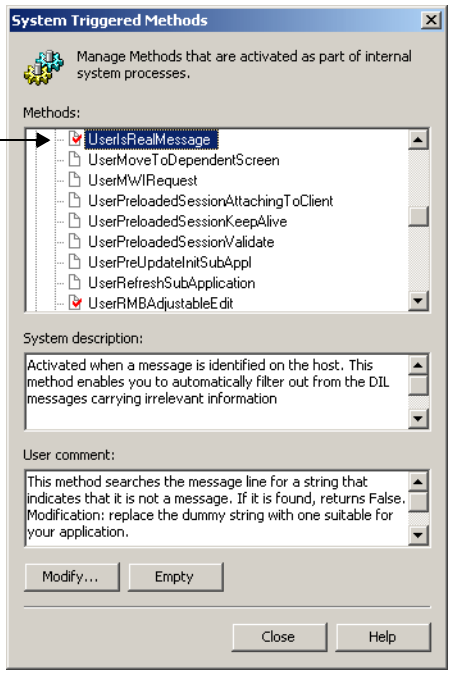


Figure 161. System-Triggered Methods dialog box

A System-Triggered Method is predefined in the source code, but it can be enhanced or overridden by the Application developer, within the context of a specific Application.

In the *System-Triggered Methods* manager, the methods have a small icon to the left of their names - either a blank page, or a page with a red check mark. A blank page indicates that this System-Triggered Method has not been modified within

the context of this Application, and will function as predefined in the source code. This means that when you view such a method it is empty - it does not contain any instructions.

The Application developer can modify the predefined functionality of a System-Triggered Method by adding method instructions to it. Once modified, a red check mark is added to this method's icon. In this manner, you can easily identify the System-Triggered Methods which have been modified by the Application developer.

The editing options available in the *System-Triggered Methods* manager are:

Modify	Click to modify the selected System-Triggered Method.
Empty	Remove all the method lines from the selected method. This means that you revert back to the original predefined functionality of the System-Triggered Method, and the method's icon is once again a blank page

Note: To search for a method, right-click one of the elements in the methods tree. Select *Find* from the shortcut menu, and type the string to search for in the dialog box that opens. To exit the *Find* dialog box, press the *Esc* key.

For more information, see "Method Operations: System-Triggered Methods" on page 365.

UserRMB Methods

UserRMB methods belong to the System-Triggered Methods group. UserRMB methods are activated by a right mouse button (RMB) click on a control. There is a UserRMB method associated with each control type.

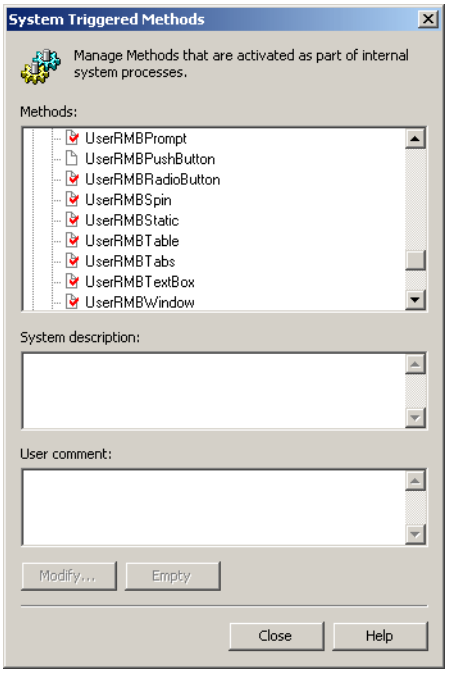


Figure 162. UserRMB methods in System-Triggered Methods dialog box

UserRMB methods can be modified to make the runtime perform any operation when the user presses the right mouse button on that control.

Right-mouse button clicks are usually associated with floating menu items. For this reason, most General UserRMB methods are supplied by default with a predefined implementation that displays the appropriate floating menu item. Therefore, these methods are displayed in the *System-Triggered Methods* manager with a red check mark next to them, denoting their implementation.

> **Note:** The *UserIsRealMessage* method is also provided with an implementation.

The same UserRMB methods do not have a red check mark when they appear in the Current Library or the Current Subapplication list, because the method is empty. To trigger a different method in a specific Subapplication when right clicking on a control, modify the appropriate Current Subapplication or Current Library method.

The implementation of existing General UserRMB methods:

Table 18. General UserRMB method implementation (Sheet 1 of 2)

UserRMBMethod	Implementation
UserRMBAdjustableEdit	Calls the *UserRMBEditable* method.
UserRMBButton	Empty.
UserRMBCheckBox	Calls the *UserRMBWindow* method.
UserRMBComboBox	Calls the *UserRMBEditable* method.
UserRMBDate	Calls the *UserRMBEditable* method.
UserRMBEditBox	Calls the *UserRMBEditable* method.
UserRMBFrame	Calls the *UserRMBWindow* method.
UserRMBGroupBox	Calls the *UserRMBWindow* method.
UserRMBOwnerDrawList Box	Opens the List window menu as a floating menu. Used for backward compatibility.
UserRMBPrompt	Calls the *UserRMBEditable* method.
UserRMBPushButton	Empty.
UserRMBRadioButton	Calls the *UserRMBWindow* method.
UserRMBSpin	Calls the *UserRMBEditable* method.
UserRMBStatic	Calls the *UserRMBWindow* method.
UserRMBTable	Opens the List window menu as floating menu.
UserRMBTabs	Empty.

Table 18. General UserRMB method implementation (Sheet 2 of 2)

UserRMBMethod	Implementation
UserRMBWindow	Opens the Commands window menu as a floating menu.

Note: *UserRMBEditable* is not a System-Triggered Method. It is a User-Triggered Method and can be modified from the *User-Triggered Methods* manager. The supplied *UserRMBEditable* opens the Edit window menu as a floating menu.

User-Triggered Methods vs. System-Triggered Methods

The following table compares User-Triggered Methods and System-Triggered Methods:

Table 19. User- versus System-triggered methods (Sheet 1 of 2)

	User-Triggered Methods	System-Triggered Methods
Activated by user actions	Yes	No
Activated by system events	No	Yes
Write new	Yes	No
Modify	Yes	Yes
Delete	Methods supplied with ACE - No Developer-written methods - Yes	No
Rename	Yes	No

Table 19. User- versus System-triggered methods (Sheet 2 of 2)

	User-Triggered Methods	System-Triggered Methods
Empty	No	Yes

Linking User-Triggered Methods to Controls

User-Triggered Methods can be linked to controls in Design View or in the KnowledgeBase.

In Design View, linking a method to a control is a local operation that affects only the specific control in the specific Subapplication. In this manner, it is possible to link a method of any level - General, Current Library or Current Subapplication. However, whenever possible, it is recommended to link User-Triggered Methods to controls in an automatic manner, via the KnowledgeBase.

In the KnowledgeBase, you can attach a General or Library Level User-Triggered Method to a Representation Definition, as one of its components. When the Representation Definition contains one or more method-triggering components, such as a button, a menu item or an accelerator, the method is automatically attached to each one of these components. This ensures that each occurrence of these controls throughout the Application or library, will trigger the attached method.

Linking Methods to Controls in a Subapplication

In Design View, the *Events* tab in the *Component* dialog box shows which Method is linked to the selected control.

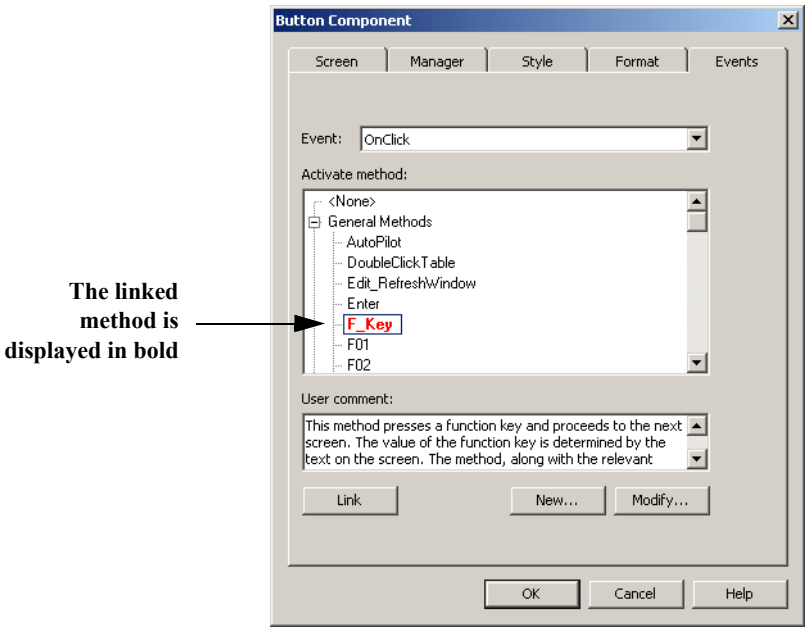


Figure 163. Events tab of a component dialog box

In this dialog box:

- Find out which method, if any, is attached to the control.
- Read a short description of the method.
- Link a method to the control.
- Access the methods manager to write new methods or edit existing methods.

To view the Method linked to a control:

1. Open the control's *Component* dialog box by:

   Double-clicking the control.

   -or-

   Right-clicking the control in the *Window Components* palette and selecting *Modify* from the shortcut menu.
2. Go to the *Events* tab. The method linked to the selected control is displayed in bold red font in the *Activate method* list.

Note: Right-clicking within the *Activate method* list opens a shortcut menu with the following options: *Link*, *New*, *Modify* and *Find*.

To link a method to a control in Design View

1 Open the control's *Component* dialog box by:
   Double-clicking the control.
   -or-
   Right-clicking the control in the *Window Components* palette and selecting *Modify* from the shortcut menu.
2 Go to the *Events* tab and select a method.
3 Click *Link*. The selected method is displayed in bold red font.
4 Click *OK*. The method is now linked to the control.

Note: When the control is a menu item or an accelerator, use the *Subapplication Menu Editor.*

Detaching Methods from Controls

To detach a User-Triggered Method from a control:

1 Open the control's *Component* dialog box by:
   Double-clicking the control.
   -or-
   Right-clicking the control in the *Window Components* palette and selecting *Modify* from the shortcut menu.
2 Go to the *Events* tab. The name of the method attached to the control is displayed in bold red font.
3 Select the *<None>* item at the top of the *Activate method* list.
4 Click *Link*. *<None>* is now displayed in bold red font.
5 Click *OK*. Now there is no method linked to the control.

Note: When the control is a menu item or an accelerator, use the *Subapplication Menu Editor.*

Linking Methods to Controls in the KnowledgeBase

In the KnowledgeBase, General User-Triggered and Current Library User-Triggered Methods can be assigned as Representation Definition components. This is done using the GeneralUTMethod (GUTM) representation component. In the *Style* tab, select the method to assign to the GeneralUTMethod representation component. This method is automatically attached to each of the other components in the Representation Definition. This ensures that each occurrence of these components throughout the Application or the library, will trigger the attached method.

Note: A Current Subapplication User-Triggered Method cannot be a Representation Definition component.

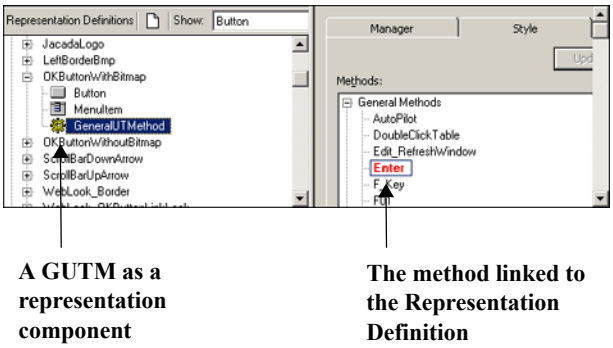


Figure 164. Linking methods to controls in the KnowledgeBase

The figure shows the components of the *OKButtonWithBitmap* Representation Definition in the KnowledgeBase. The representation is comprised of the following components: a Button, a MenuItem, and a GeneralUTMethod. Attaching a method, in this case the *Enter* method, to the *OK* button's Representation Definition, means that all occurrences of the *OK* button control in the Application have the functionality defined by the linked method.

To link a General User-Triggered Method or Current Library User-Triggered Method to a control in the KnowledgeBase

1. In the KnowledgeBase, switch to Representation Definition View.
2. In the lower left panel, right click on a Representation Definition component and select *New*. The *New Representation Component* dialog box opens:

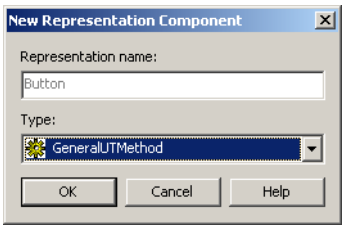


Figure 165. New Representation Component dialog box

3. Choose *GeneralUTMethod* from the *Type* combo box and click *OK*. A General UT Method is added as a component of the selected Representation Definition.
4. Open the *Style* tab in the lower right panel and select a method from the list of General Methods or Current Library Methods.
5. Click *Link*. The selected method is now attached to the GeneralUTMethod Representation Definition component.

Writing Methods

The following topics describe how to write methods:

- The *Define Method* dialog box.
- Method operations.
- Method line types.
- Method line syntax.
- DoMethod method line type.
- Referencing a method line.
- Method examples.

The Define Method Dialog Box

A method is a sequence of one or more instructions. Each instruction is called a method line, and is of a certain type. ACE provides a set of method line types.

To write a new method or modify an existing one you use the *Define Method* dialog box:

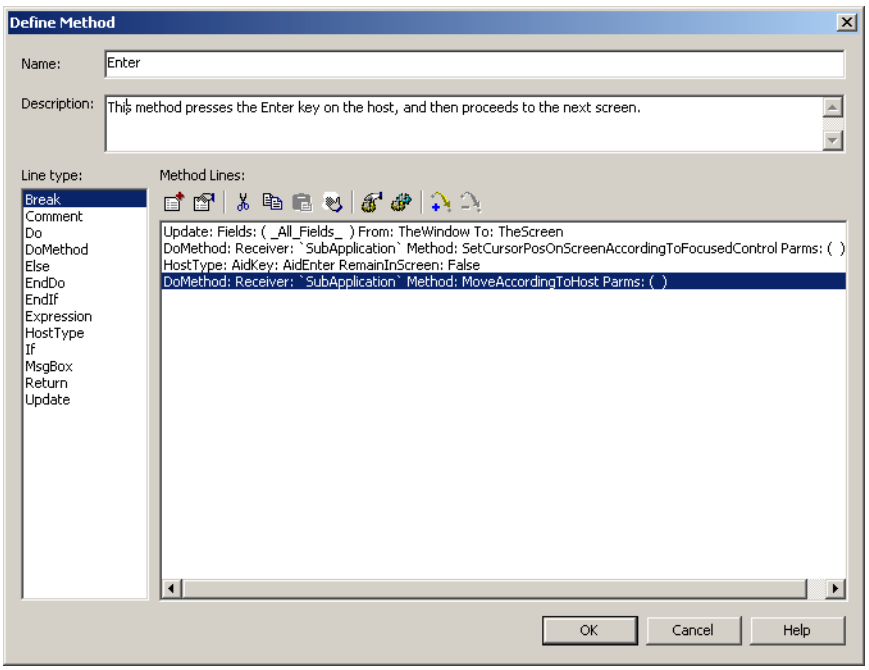


Figure 166. Define Method dialog box

The *Define Method* dialog box contains:

- The *Name* and *Description* of the method.

- A list of the available method *Line types*.

  To add a line to a method, you use the suitable method line type. Most method line types must be configured in a dialog box before they are added to the method. For example, for an If method line type, the condition being tested is specified in the *If* configuration dialog box.

  For more information, see "Method Line Types" on page 366.
- A list of the *Method Lines* that compose this method.

  The toolbar directly above the method lines provides method editing options. These editing options are also available from the shortcut menu that opens when you right click within the *Method Lines* list.

Working with Method Lines

The *Define Method* dialog box contains a toolbar which provides editing options such as cut, copy and paste of method lines, the ability to insert the lines of an existing method into the current method, and to add or remove a reference to a method line.

The existence of selected method lines in the dialog box influences the editing options that are enabled in the toolbar and in the shortcut menu. For example, when there are no method lines selected, you can only add a method line or insert a User or System-Triggered Method into the method. When one or more method lines are selected, it is also possible to perform cut, copy and clear selection operations. When a single method line is selected, it is possible to add or remove a reference to a method line as well.

The selection of method lines also influences the location into which method lines are added, as explained in the following section.

Adding Lines to a Method

When adding lines to a method, the location in which the lines are added depends on the type of operation and on the existence of selected method lines in the dialog box.

The following rules apply:

- When *no* method lines are selected, method lines are added to the *end* of the method.
- When adding a method line:
  - When *one or more* method lines are selected, a new method line is added *below* the *topmost* selected line.
- When pasting method lines or inserting the lines of an existing method:
  - When *one or more* method lines are selected, the method lines are added *above* the *topmost* selected line.

Note: To clear the selection of lines in a method, use the *Unselect* option.

Method Editing Options

Table 20. Method editing options (Sheet 1 of 2)

Icon	Function	How to use
	Add Method Line	To add a method line: **1** Select a line type. **2** Click the icon. The line type's configuration dialog box opens. **3** Enter the necessary information and click *OK*. The new line is added to the method.
	Modify Method Line	To add a method line: **1** Select a method line. **2** Click the icon. The line's configuration dialog box opens. **3** Enter the necessary information and click *OK*. *Note*: Double clicking a method line also opens the line's configuration dialog box.
	Cut	**1** Select one or more method lines. **2** Click the icon. The lines are removed from the method and stored on the clipboard.
	Copy	To copy a method line: **1** Select one or more method lines. **2** Click the icon. The line are copied to the clipboard.
	Paste	To paste a method line: **1** Select a method line. **2** Click the icon. The clipboard contents are pasted into the method.

Table 20. Method editing options (Sheet 2 of 2)

Icon	Function	How to use
	Unselect	Click to unselect selected method lines.
	Insert User-Triggered Method	Use this feature to insert the lines of an existing method into the current method: **1** Click the icon. The *Select User-Triggered Method to Insert* dialog box opens. **2** Select the method and click *OK*. All the lines of the selected method are inserted into the current method. *Note*: A Current Subapplication Method cannot be inserted into another Subapplication's method. A Current Library Method cannot be inserted into another library's method.
	Insert System-Triggered Method	Use this feature to insert the lines of an existing method into the current method: **1** Click the icon. The *Select System-Triggered Method to Insert* dialog box opens. **2** Select the method and click *OK*. All the lines in the selected method are inserted into the current method. *Note*: When selecting a System-Triggered Method, only modified methods appear in the list and are available for insertion.
	Add Reference	"Referencing a Method Line from Another Method Line" on page 394
	Remove Reference	**1** Select a method line. **2** Click the icon. The reference number is removed from the selected line.

Method Operations: User-Triggered Methods

You can perform the following operations on User-Triggered Methods:

- Modify an existing method
- Write a new method
- Rename a method
- Delete a method

Modifying User-Triggered Methods

To modify a method from the User-Triggered Methods manager:

1 In Design View, select *Design > User-Triggered Methods*
The *User-Triggered Methods* manager opens.

2 Expand one of the method categories lists.

3 Double click a method.
-or-
Select a method and click *Modify*
-or-
Right-click a method and select *Modify* from the shortcut menu.
The *Define Method* dialog box opens.

4 Modify the method by adding new method lines or modifying existing ones.
For more information, see "Method Line Types" on page 366.

5 Click *OK* to save your changes.

Modifying User-Triggered Methods Linked to Controls

To modify a method that is linked to a control:

1 Open the control's *Component* dialog box by:
Double-clicking the control.
-or-
In the *Window Components* palette, right-click the control and select *Modify* from the shortcut menu.

2 Go to the *Events* tab.

3 Select the method attached to the control (displayed in bold red font) and click *Modify,* or double-click the method.
The *Define Method* dialog box opens.

4 Modify the method by adding new method lines or modifying existing ones.
For more information, see "Method Line Types" on page 366.

5 Click *OK* to save your changes.

Writing New User-Triggered Methods

To write a new user-triggered method:

1. Open the *User-Triggered Methods* manager from the *Design* menu in Design View, or from the *Define* menu in the KnowledgeBase.
2. Click *New* and select the method level - General, Current Subapplication, or Current Library. The *Define Method* dialog box opens.
3. Type a method name in the *Name* field.
4. Type a description in the *Description* field.
5. Add method lines to the method.

   For more information, see "Writing Method Lines" on page 368.
6. Click *OK* to save the method.

Renaming User-Triggered Methods

To rename a method:

1. Open the *User-Triggered Methods* manager.
2. Select a method from the *Methods* list. Right-click and select *Rename* from the shortcut menu.

   -or-

   Click the *Rename* button.

   A warning is displayed as a reminder to update all references to the method's name. Click *OK*.
3. The *Renamed Method New Name* dialog box opens, displaying the method's current name.
4. Change the method name as desired.
5. Click *OK* to save your changes.

Note: Renaming a method changes only the method name itself. You will need to update all uses of the renamed method, such as in DoMethod lines and representation components of General User-Triggered Methods.

Deleting User-Triggered Methods

To delete a method:

1. Open the *User-Triggered Methods* manager.
2. Select a method.

3 Right-click and choose *Delete* from the shortcut menu.

   -or-

   Click the *Delete* button.

   A warning message is displayed.

4 Click *Yes* to delete the method.

Method Operations: System-Triggered Methods

You can perform the following operations on System-Triggered Methods:

- Modify a method
- Empty a method

Modifying System-Triggered Methods

To modify a System-Triggered Method:

1 Open the *System-Triggered Methods* manager from the *Design* menu in Design View, or from the *Define* menu in the KnowledgeBase.

2 Double click a method

   -or-

   Select a method and click *Modify*

   -or-

   Right click a method and select *Modify* from the shortcut menu.

   The *Define Method* dialog box opens.

3 Modify the method by adding new method lines or modifying existing ones. For more information, see "Method Line Types" on page 366.

4 Click *OK* to save your changes.

Emptying System-Triggered Methods

In the *System-Triggered Methods* manager, a red check mark next to a method name means the method contains method lines. The method lines have either been written by the developer or are implementations provided with ACE (such as the *UserIsRealMessage* and the *UserRMB* methods).

ACE does not permit deleting System-Triggered Methods. However, it is possible to empty a System-Triggered Method by removing all its lines. This means that the functionality of the method reverts back to the original functionality as predefined in the source code, and the icon next to the method name becomes a blank page.

To empty a System-Triggered Method:

1 Open the *System-Triggered Methods* manager.
2 Select a method preceded by a red check mark.
The *Empty* button is enabled.
3 Right-click and select *Empty* from the shortcut menu.
-or-
Click the *Empty* button.

Note: When attempting to empty a General System-Triggered Method, a warning is issued as a reminder that the procedure empties the method for the whole Application and not only for the current Subapplication.

4 Click *Yes* to empty the method.
The red check mark no longer appears next to the method name and the method is empty.

Method Line Types

Method lines are the building blocks of methods. ACE uses several kinds of method line types to provide the functional diversity needed to deal with as many programming situations as possible.

Here you learn how to use each method line type to write syntactically and functionally correct methods.

Note: Incorrectly written methods may cause compilation errors.

First, Table 21 briefly reviews all the method line types. Then, more information on writing methods is provided, followed by a detailed explanation of each method line type.

Table 21. Method Line types (Sheet 1 of 3)

Type	Description
Break	Break out of a loop.
Comment	Insert comment lines to improve the readability of the method lines.

Table 21. Method Line types (Sheet 2 of 3)

Type	Description
Do	Specify the number of times that method lines that appear between the Do and EndDo instructions are executed. Do and EndDo method lines must always be used together.
DoMethod	Activate a DoMethod. DoMethods are predefined methods that can be activated within a method written by the ACE developer. ACE is supplied with a great number of doMethods. For more information, see Chapter 15 - "Methods: DoMethods List" on page 429.
Else	See If.
EndDo	See Do.
EndIf	See If.
Expression	Write an expression which is returned as a value to be used by other method lines.
HostType	Specify actions to be done in the host screen.
If	Create a conditional expression that must have a logical result. Every If must be followed by an EndIf method line at the end of the condition lines. When necessary, an Else line can be inserted between an If and an EndIf line.
MsgBox	Define a message box to be displayed to the user in runtime.
Return	Return one of the following values: • SUCCESS (same as NO VALUE) • FAIL • TRUE • FALSE • A value returned by another method line

Table 21. Method Line types (Sheet 3 of 3)

Type	Description
Update	Update variables between the host and the GUI.

Writing Method Lines

Most of the method lines are defined through a configuration dialog box, where you specify all the details of a line before it is added to a method. The dialog box varies according to the method line type and may require different types of information such as integers, mathematical, logical, text, and may be composed of various items, such as constants, variables, operators, or references to a value returned by another method line. When added to a method, each line has a defined syntax, according to its type. All method lines begin with the name of the method line type.

Note: The Break, Else, EndDo, and EndIf method line types do not need to be configured and are added directly to a method.

The following general rules apply when working with a method line's configuration dialog box:

- Constants can be typed directly into a field.
- Some dialog boxes contain built-in constants, such as various operators. To add a built-in constant to an expression, click the constant with the mouse.
- Some dialog boxes display lists of variables. To add a variable, either select the variable and click the *Add* button or double-click the variable.
- Clicking the *Concatenate* button adds a plus sign (+) to an expression, which serves as a concatenator between strings.
- You can add a reference to a value returned by an existing method line. For more information, see "Referencing a Method Line from Another Method Line" on page 394.
- Erasing an item from an expression is done using conventional options, such as selecting the item to be deleted with the mouse and pressing the *Delete* key.

The following sections describe the various method line types in more detail.

Due to its large scope, the DoMethod line type is discussed in a separate section. For more information, see "DoMethod Method Line Type" on page 383.

Note: When working with method dialog boxes, right-clicking in a dialog box opens a shortcut menu from which you can choose to *OK All, Cancel All*, or open the Help. This option is available only when the initial dialog box is either the *User-Triggered* or *System-Triggered Methods* manager.

Break

A *Break* type method line is used to break out of a loop. It can be used between the Do and EndoDo method line types.

The Break method line type does not require any specifications. The line is added directly to the method.

The syntax of the Break method line is the following:

```
Break:
```

To add a Break line:

1. In the *Define Method* dialog box, double-click the Break line type.
   -or-
   Highlight Break in the *Line type* list and click .

Comment

A *Comment* type method line is used to insert a comment. It can be useful to provide detailed comments on various method Lines, especially when several developers work on the same Application.

A comment line looks like this:

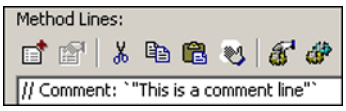


Figure 167. Comment method line type

To add a Comment line:

1. In the *Define Method* dialog box, double-click the Comment line type.
   -or-
   Highlight Comment in the *Line type* list and click .
   The *Comment* dialog box opens.
2. Type the text of the Comment line.
3. Click *OK*.

Do and EndDo

The Do method line is used to execute a set of method lines a specific number of times. The repeated method lines must be surrounded by the Do method line at the top and the EndDo method line at the end.

The EndDo method line is added directly to a method.

An example of syntax of the Do/EndDo method lines is as follows:

```
Do: Times: `5`
Method Line
. . . . . .
. . . . . .
Method Line
EndDo:
```

To add a Do line:

1. In the *Define Method* dialog box, double-click the Do line type.

   -or-

   Highlight Do in the *Line type* list and click .

   The *Do* dialog box opens.
2. Type an integer constant in the *Number of iterations* field

   -or-

   Click the *Reference* button to insert a reference to the value returned by an existing method line. For more information, see "Referencing a Method Line from Another Method Line" on page 394.

   Make sure that the returned value is an integer.
3. Click *OK*.

To add an EndDo line:

1. In the *Define Method* dialog box, double-click the EndDo line type

   -or-

   Highlight EndDo in the *Line type* list and click .

Expression

The *Expression* method line type is used for two purposes:

2. To build an expression and return its value so that other method lines can use that value, for example:

   `#1 = Expression: Expr: `#0 + 1``
3. To build an expression and set a variable's value to that of the expression, for example

   `Expression: Expr: `#0` SetVar: CommandLine`

This is the dialog box for the Expression method line type:

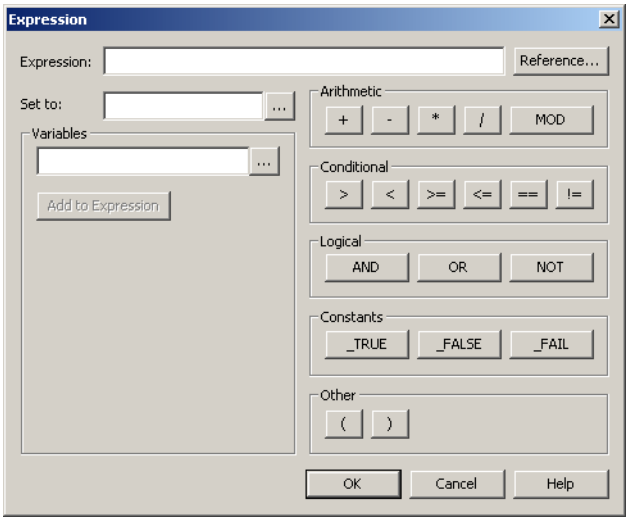


Figure 168. Expression method line type dialog box

To build an Expression:

An Expression is any combination of constants, variables, and operators used to represent a value. Use the following elements to build an expression:

Constants	Type strings or numbers.
Variables	• Add variables. • Add a reference to the value returned by another method line.
Operators	Add arithmetic, conditional or logical operators and parentheses to combine constants and variables in the desired expression.

The Variable Lookup Dialog Box

The *Variable Lookup* dialog box is used to select a variable when setting variables in methods. The *Variable Lookup* dialog box can be accessed from the configuration dialog boxes of the following method line types:

- DoMethod
- Expression
- HostType
- If
- MsgBox
- Update

Click the ... button to open the *Variable Lookup* dialog box:

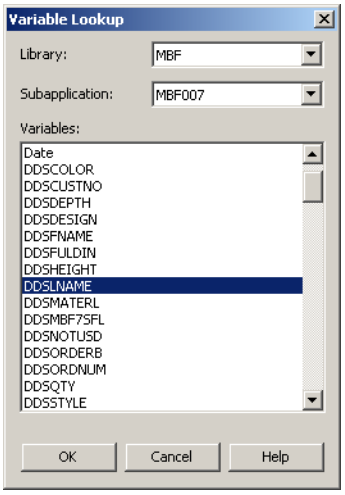


Figure 169. Variable Lookup dialog box

To select a variable from the *Variable Lookup* dialog box:

1. In an Application level method, select a library from the *Library* list and then a Subapplication from the *Subapplication* list. In a library level method, you only need to select a Subapplication.
2. Select the desired variable from the list and click *OK*, or simply double click the variable. The variable name is inserted into the method line type's configuration dialog box.

Note: In Subapplication level methods, the *Variable Lookup* dialog box is not accessible. Instead, a list of the current Subapplication variables is displayed directly in the method line type dialog box.

The Set To Option

Set to contains a list of all the variables in the Manager for the selected Subapplication, the Subapplication's library, or the entire Application. Selecting a variable from *Set to* sets the value of the selected variable to that of the expression. Note, however, that to update this value from the Manager to the Window you need to use the Update method line type. In addition, do not use *Set to* with a list.

To set a variable to the value of an expression:

1. Build the expression using constants, variables, indicators, and operators as described above.

2 For Subapplication methods select the variable from the list of variables in the *Set to* field.

For library and Application level methods select the variable in the *Variable Lookup* dialog box.

The value of the selected variable is set to the value of the expression.

To add an expression line:

1 In the *Define Method* dialog box, double-click the Expression line type. The *Expression* dialog box opens.
2 Build the expression using constants, variables, indicators, and operators as described above.
3 When the expression is complete, click *OK*.

HostType

The *HostType* method line is used to specify actions to be done in the host screen. One or more of the following actions can be carried out:

- Move the cursor to a specified field.
- Type text in the current cursor location.
- Type text in a specified field.
- Press a specified key.

This is an example of a HostType method line, which types the string “1” in the current cursor location and then presses the *Enter* key:

```
HostType: Text: `"1"` AidKey: AidEnter RemainInScreen: False
```

This is the dialog box of the *HostType* method line type:

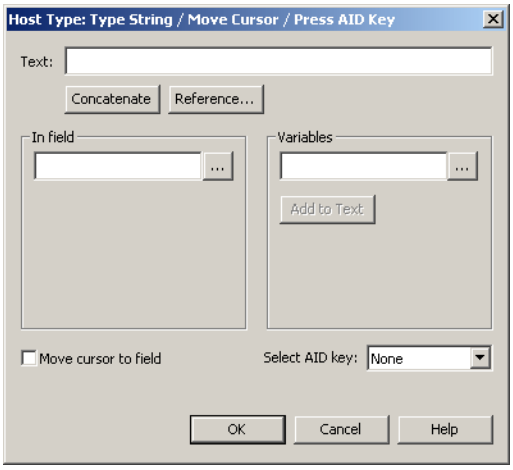


Figure 170. HostType method line type dialog box

Note: In a Current Subapplication method, both *In field* and *Variables* contain list boxes which display all the Subapplication fields and variables, respectively.

The operations specified in the *Host Type* dialog box are carried out in the following order:

- The cursor is moved to the field, if a field is selected.
- The text is typed in the field, or at the cursor's location when no field is selected.
- The AID key is pressed, if specified.

Text

Enter the text to be typed in the host screen. The text may also contain variables or a reference to the value returned by an existing method line.

Text is entered according to the following instructions:

- The *Concatenate* button inserts a plus sign (+) into the *Text* field that serves as a concatenator of strings.
- In Application Level and Library Level Methods, select a variable from the *Variable Lookup* dialog box. and click *Add to Text* to add a variable name to the text field. For more information, see "The Variable Lookup Dialog Box" on page 371.
- In Subapplication Level Methods, select a variable from the *Variables* list. Double-click the variable or click *Add to Text* to add it to the text field.
- Click *Reference* to insert a reference to the value returned by a method line. For more information, see "Referencing a Method Line from Another Method Line" on page 394.

In Field

Select the name of the field in the host screen where the text will be typed.

If no field is specified, the text is typed at the current cursor position.

Move Cursor to Field

When checked, the cursor moves to the field selected in *In field*.

Note: It is also possible to type text in a field without moving the cursor to the field.

Select AID key Select the key to be pressed. The runtime waits until the AID key's action is completed before continuing.

Note: The difference between a variable and a field is that *Variable* refers to the Subapplication Manager and *Field* refers to the *host screen*.

To add a HostType line:

1. In the *Define Method* dialog box, double-click the *HostType* line type.

   The *Host Type* dialog box opens.
2. Insert the required text in the *Text* field.

   Use the *Concatenate* button to insert a plus sign (+) which serves as a concatenator between strings, variables, and references to values returned by other method lines.
3. Select a field from *In field*. If a field is not selected, the text is typed at the current cursor position.
4. Set the *Move cursor to field* check box to move the cursor to the selected field.
5. Select the AID key to be pressed.
6. Click *OK*.

Note: The “RemainInScreen: False” string at the end of the method line is added automatically by ACE. This is an internal parameter which cannot be changed.

If, EndIf, and Else

The *If* dialog box looks identical to the *Expression* dialog box, except that the *Set to* field does not appear in the *If* dialog box. The difference between the If method line and the Expression method line is that the If method line is a conditional expression that must have a logical result.

Every If method line must have an EndIf method line at the end of the condition lines. When necessary, an Else line can be inserted between an If and an EndIf line. Both the EndIf and the Else method line types are added directly to a method when double-clicked.

MsgBox

A *MsgBox* method line displays a message box when the runtime Application is running.

This is an example of a MsgBox method line:

```
MsgBox: MsgBoxType: Exclamation Message: `"You did not select a Navigation
Option"` Caption: `"Select Option"` Default: 1
```

> **Note:** The "`Default:1`" string at the end of the method line is added automatically by ACE. This is an internal parameter which cannot be changed.

This is the dialog box of the MsgBox method line type:

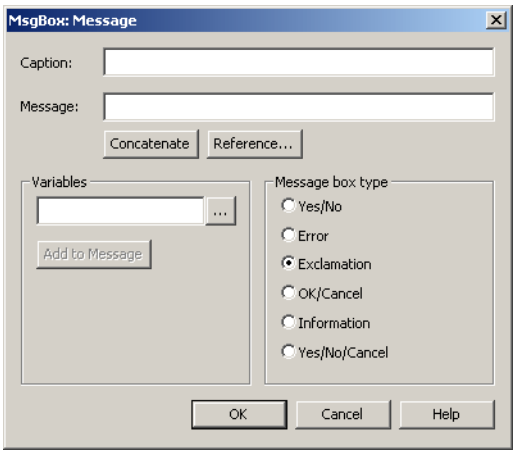


Figure 171. MsgBox method line type dialog box

Enter the caption of the message box in the *Caption* field. The caption must be a string. By default, the caption entered here is the caption of the Subapplication window. When the Subapplication has no window, the Subapplication name is used.

> **Note:** It is recommended to include the Application name in the caption to simplify the identification of a message box when switching between Applications using *ALT+TAB*.

In the *Message* field enter the text to display in the message box, enclosed in quotation marks. The text can also contain a variable, and you can also insert a reference to the value returned by an existing method line.

The following types of message boxes are available:

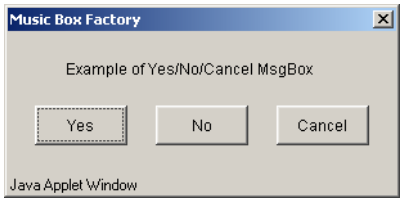
- *Error*

  An Error message box informs the user that an error has occurred. A stop sign icon appears in the message box.
- *Exclamation*

  An Exclamation message box presents the user with a warning message. An exclamation mark icon appears in the message box.

- *Information*

  An Information message box provides the user with information. An icon with the lower case letter 'i' appears in the message box.
- *Yes/No*, *OK/Cancel*, and *Yes/No/Cancel* types:



  When the user clicks one of the buttons, the MsgBox method line returns a value that can be used when referring to this method line in a different method line.

  Each of these message boxes return an integer value depending on which button the user presses during runtime. The value returned also depends on the following setting in the specific.INI file:

```
[MakeExeParms]
DisplayMessageBoxAsInt = 0 or 1
```

Table 22 summarizes all possible return values, sorted according to the type of message box, the value of the INI file setting, and which button was clicked.

Table 22. Return values

Message Box Type	DisplayMessageBox AsInt = 0	DisplayMessageBoxAsInt = 1 (Default value)
Yes/No	Yes - 1 No - 0	Yes - 6 No - 7
OK/Cancel	OK - 1 Cancel - 0	OK - 1 Cancel - 0
Yes/No/Cancel	Yes - 1 No - 0 Cancel - 2	Yes - 6 No - 7 Cancel - 2

When the `DisplayMessageBoxAsInt` setting does not appear in the SPECIFIC.INI file, ACE assumes the default value, which is 1. To use the non-default value, you must add the line `DisplayMessageBoxAsInt=0` in the `[MakeExeParms]` section of the SPECIFIC.INI file.

To add a MsgBox line to a method:

1. In the *Define Method* dialog box, double-click the MsgBox line type.

   -or-

   Select the MsgBox line type and click the *Add Method Line* button.

   The *MsgBox: Message* dialog box opens.

   When necessary, change the text in the *Caption* field. The string must be surrounded by quotation marks (").
2. Insert the message text in the *Message* field, according to the following instructions:
   - A string must be surrounded by quotation marks (").
   - In Application Level and Library Level Methods, select a variable from the *Variable Lookup* dialog box. For more information, see "The Variable Lookup Dialog Box" on page 371.

     Insert the variable by selecting it and clicking the *Add to Message* button. In a Subapplication method you can also double-click the variable. A variable name does not need to be surrounded by quotation marks (").
   - Click the *Reference* button to insert a reference to a value returned by a method line.
   - Click the *Concatenate* button to add a plus sign (+) to the message. This joins different elements such as strings, variables, and references, to values returned by another method line.
3. Select the type of message box.
4. Click *OK*.

Return

The Return method line type is used to cause a method to return a specific value. The configuration dialog box of the Return method line type has two different versions depending on the method type; User-Triggered, System-Triggered, and Message Handling Methods.

User-Triggered and System-Triggered Methods

This is an example of Return method type line for User and System-Triggered Methods:

```
Return: True
```

The *Return: Method Return Value* dialog box looks like this:

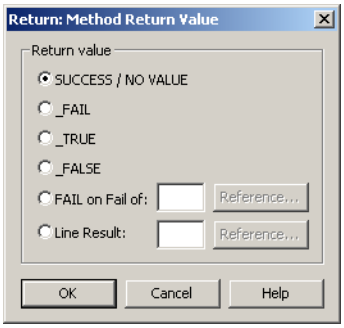


Figure 172. Return method line type dialog box

- *SUCCESS/NO VALUE*

  This is the default return value, when a method does not return a special value.
- *_FAIL*

  The method returns a FAIL value.
- *_TRUE*

  The method returns a TRUE value.
- *_FALSE*

  The method returns a FALSE value.
- *FAIL on Fail of:*

  This option requires a reference to a method line. This option is like *_FAIL* but tests a condition first. The method returns _FAIL only when the referenced line returns _FAIL. When the referenced line returns anything other than _FAIL, the method returns SUCCESS.
- *Line Result:*

  This option requires a reference to a method line. The method returns the return value of the referenced method line.

Method Handling Messages

This is an example of a Return method type line for Method Handling Messages:

```
Return: ReachContinue
```

The *Return: Move Return Value* dialog box looks like this:

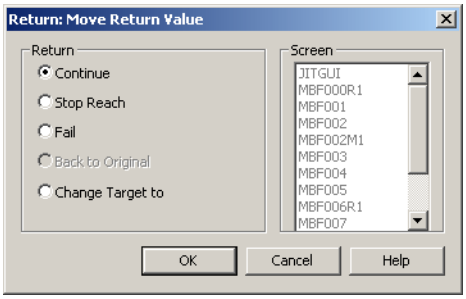


Figure 173. Return: Move Return method line type dialog box

This *Return* dialog box specifies the return value of a move:

- *Continue*
  This return value instructs to continue the current reach.
- *Stop Reach*
  This return value instructs to stop the current reach.
- *Fail*
  This return value indicates that the reach has failed.
- *Back To Original*

For future use.

- *Change Target to*
  This return value is used when the target (destination) screen changes during the method. When you select this option, the *Screen* list becomes active, and you can select the new target screen.

To add a Return line to a method:

1. In the *Define Method* dialog box, double-click the Return line type.
   -or-
   Click the Return line type and click the *Add Method Line* button.
   The *Return* dialog box opens.
2. Select the value to be returned.
3. In the *Method Return Value* dialog box, *FAIL on Fail of* and *Line Result* both require a reference to a method line. To insert the reference, click the *Reference*

button. For more information, see “Referencing a Method Line from Another Method Line” on page 394.

In the *Move Return Value* dialog box, *Change Target to* requires that you select a target screen from the *Screen* list.

4 Click *OK*.

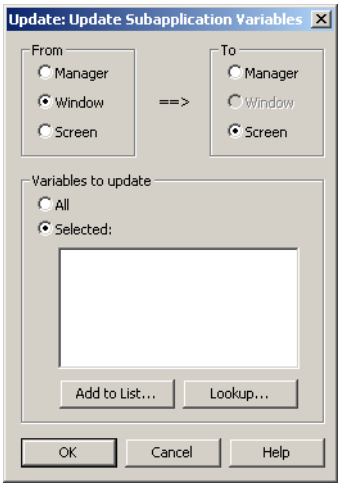
Update

An *Update* method line type updates variables in the specified direction. You can update variables in all possible directions between the Host screen, the Subapplication Manager and the Window.

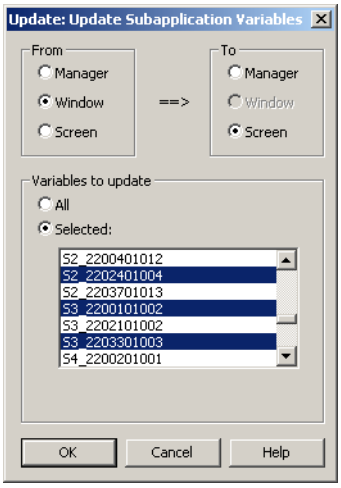
This is an example of an Update method line type which updates the values of all the Subapplication fields from the GUI window to the host screen.:

```
Update: Fields: ( _All_Fields_  ) From: TheWindow To:TheScreen
```

This is the dialog box of the Update method line type:



Application or Library Level



Subapplication Level Method

Figure 174. Update method line type dialog box

- *From, To*

  Use *From* and *To* to specify the direction in which the variables are to be updated.

- *Variables to update*

  For Application or Library Level Methods click *Lookup* to open the *Variable Lookup* dialog box. For more information, see “The Variable Lookup Dialog Box” on page 371. You can select more than one variable. Select the desired variables and click *OK*. This populates the *Variables to update* list.
  For all method levels *Variables to Update* is a multiple selection list from which you select the variables to be updated. Select *All* to update all the variables at once, or select *Selected* to select which variables to update.

- *Add to List*

  For Application or Library Level Methods you can add a variable to the list by clicking *Add to List* and typing the name of the variable in the dialog box that appears:

Figure 175. Add Variable Name to List dialog box

When a Subapplication opens, it automatically updates all the variables from the screen to the window. This is done once. Afterwards, the Application developer is responsible for updating the value of a variable between the screen and the window when necessary.

For example, after entering data in a window, you should update the variables from the window to the screen before leaving the Subapplication.

Note: You can perform various validity checks on the contents of an editable control. This is done in the *Style* tab of the control's *Component* dialog box. The validity checks are executed during an update from the Window to the Manager. When the contents are not valid, the execution of the method is stopped.

Updating Variable Attributes

The runtime is capable of dealing with attributes that change during runtime.

When you want the variable attributes to be updated during runtime, you need to set the *Data Flow* options that update from the Screen to the Manager during runtime This is done in the *Manager* tab of the *Component* dialog box.

When you perform an Update from the Screen to the Manager during runtime, the following attribute information is saved for each field:

- Whether the field is protected
- Whether the field is hidden and protected

In this way, when the attribute of a field changes, the information in runtime is updated.

Note: For some Pattern Definitions, such as *HeaderName*, the runtime data flow options are not selected in order to improve runtime performance, and the data flow is determined at the converter stage. To update the attributes of such a Pattern Definition in runtime, you need to set the suitable *Runtime Data Flow* options in the pattern's Representation Definition.

To add an Update line to a method:

1. In the *Define Method* dialog box, double-click the Update line type.

   -or-

   Select the Update line type and click the *Add Method Line* button.

   The *Update: Update Subapplication Variables* dialog box opens.
2. Specify the direction in which the variables are to be updated, by selecting an option in *From* and *To*.
3. Select one or more variables to update.

   Click *All* to select all the variables in the list.

DoMethod Method Line Type

DoMethods are predefined methods that can be activated within a method written by the Application developer. The DoMethod method line type enables you to insert a line into your method which activates the desired predefined DoMethod.

ACE is supplied with hundreds of DoMethods which perform varied tasks such as: retrieving a variable's content, moving the cursor to a specific location on the host, writing data to and reading data from an **.ini* file, and more. This enables you to write relatively short methods which perform the varied and complex tasks that are necessary to convert your Application.

Note: For a detailed list of DoMethods, see Chapter 15 - "Methods: DoMethods List" on page 429.

DoMethod Receivers

Every DoMethod is executed by one or more receivers. Receiver is a term used in Object Oriented Programming to refer to the object that receives the request to execute a specific method.

The DoMethods in ACE are executed by the following receivers:

- Subapplication
- Screen

- Window
- Application
- System
- Controls
- Existing Method Lines

When adding a DoMethod line to your method, you need to select both the DoMethod and the suitable receiver.

DoMethod Method Line Syntax

In general terms, a DoMethod method line looks like this:

```
DoMethod: Receiver: `[Receiver name]` Method: [DoMethod name] Parms:
([Parameters])
```

Fixed elements are in regular typeface, elements that vary between each DoMethod method line are in bold typeface and placed in square brackets.

The elements of the DoMethod syntax are:

- *DoMethod:* the name of the method line type.
- *Receiver:* a header for the receiver name.
- *Receiver name:* one of the following:
  - Subapplication
  - Screen
  - Window
  - Application
  - System
  - a variable (not for General UTMs)
  - a reference to an existing method line
- *Method:* a header for the DoMethod name.
- *DoMethod name:* the selected DoMethod's name.
- *Parms:* a header for the parameters.
- *Parameters:* the parameter's value, as entered by the developer. When a DoMethod that uses no parameters is selected, the brackets are left empty. See below for more information on parameters.

To insert a DoMethod method line into a method:

1. In the *Define Method* dialog box, double-click the DoMethod method line type

   -or-

   Click the DoMethod method line type and click the *Add Method Line* button. The *DoMethod: Method Activation* dialog box opens.

2 Select the method that you want to add.
For more information, see “Selecting a DoMethod” on page 389.
3 Click *OK*.

DoMethod Parameters

Some DoMethods have parameters. A parameter may be the name of a file, the name of a variable, a string, a number, etc. Parameters enable you to use a single method for different cases.

Example 51. DoMethod parameters

The DoMethod *MoveCursorToRowCol*, executed by the Screen, enables you to move the cursor to a specific location on the host screen. This method has two parameters which enable you specify the row and column values.

The DoMethod: Method Activation Dialog Box

Adding a DoMethod method line opens the *DoMethod: Method Activation* dialog box:

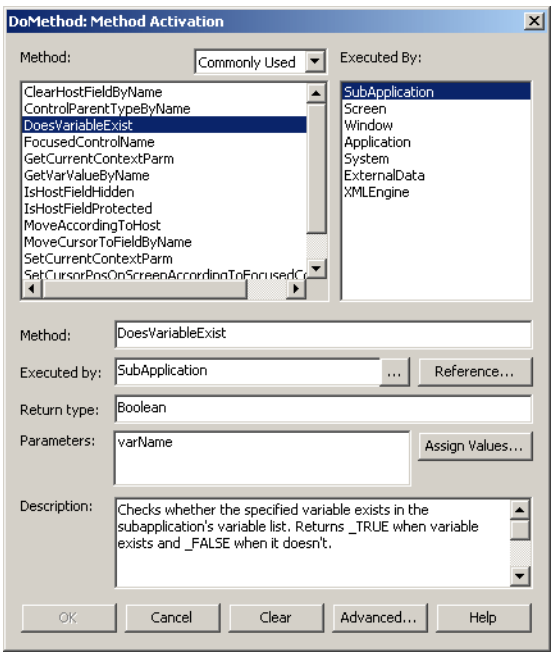


Figure 176. DoMethod: Method Activation dialog box

The elements of this dialog box are:

Table 23. DoMethod: Method Activation dialog box elements (Sheet 1 of 2)

Element	Description
Method	Displays a list of DoMethods. Which DoMethods are listed depends on two factors: **1** Which list type, *Full List* or *Commonly Used*, is currently selected in the combo box. The default setting is *Commonly Used*. **2** Which receiver is currently selected in the *Executed by* list box. **Note:** When no receiver is selected, the entire range of available DoMethods is listed. Otherwise, only DoMethods available to that type of receiver are listed.
DoMethods list type	The DoMethods list type combo box enables you to select which methods to display: • *Full List* - displays a list of all the DoMethods available to the receiver that is currently selected in the *Executed by* listbox. If no receiver is selected, all DoMethods available in ACE appear in the list. • *Commonly Used* - displays the list of commonly used DoMethods available to the receiver that is currently selected in the *Executed by* listbox. If no receiver is selected, all commonly used DoMethods appear in the list. For information on customizing the *Commonly Used* list, see "Customizing the Commonly Used Method List" on page 392.
Executed By	The *Executed by* list box displays the available receivers. Which receivers appear in the list depends on which DoMethod is selected in the DoMethod list box. **Note:** In order for a variable to be displayed in the list, at least one of the Runtime Data Flow check boxes in the Manager tab of the control's Component dialog box must be selected.

Table 23. DoMethod: Method Activation dialog box elements (Sheet 2 of 2)

Element	Description
Method, Executed By, and Return Type	These three fields display the names of the selected DoMethod, the selected receiver, and the method's return type. The content of the *Executed by* field is derived from one of the following sources: • Selecting a receiver from the *Executed by* list. • Clicking the ... button to select a variable. • Clicking the *Reference* button to insert a reference to another method line.
Parameters	The *Parameters* field lists the parameters of the selected DoMethod and their values, if already assigned.
Assign Values	Clicking this button opens the *Method Parameters* dialog box, where you assign values to the method parameters.
Description	The *Description* field displays a short description of what the selected DoMethod does. Use the description to learn about the function of each method.
Clear	Clicking this button clears the selections in the *Method* and the *Executed by* lists. Use this button to switch between viewing methods by selecting a method first, or by selecting the receiver first.
Advanced	Clicking this button opens the *Define Commonly Used Methods* dialog box, where you can customize the commonly used DoMethods list.

Contents of the Method List Box

Clicking a receiver inserts its name in the *Executed by* field and the *Method* list box lists only the DoMethods associated with this receiver.

Existing methods are in themselves DoMethods, usually associated with the Subapplication receiver. The method list box lists among the available DoMethods:

- User-Triggered Methods defined by the user, or provided by the KnowledgeBase (*Enter*, *FKey*, etc.…).
- System-Triggered Methods that are not empty (i.e. have a red check mark near their name in the *System-Triggered Methods* manager).

Note: Since method A can call method B in a DoMethod method line as part of its instructions, and method B can call method A as part of its instructions, there is the possibility of mistakenly writing methods that create infinite loops. Such infinite loops either cause compiler errors or a crash during runtime.

In Subapplication Level Methods, when the currently selected receiver is a variable ("Sub...", or "DDS…"), the method list box displays the DoMethods that can be executed on the control that is related to the variable, such as *Enable*, *Disable*, *HideControl*, *ShowNormal*, etc.

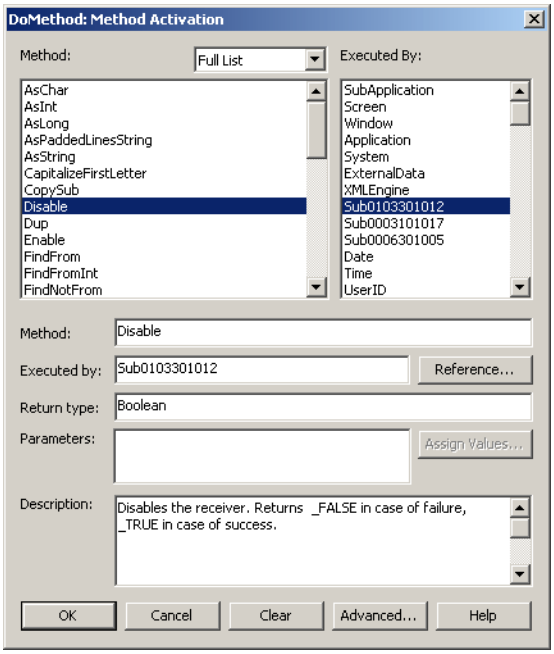


Figure 177. What happens when the receiver is a variable

Selecting a DoMethod

The *DoMethod: Method Activation* dialog box displays a list of DoMethods on the left and a list of receivers on the right.

When the *DoMethod: Method Activation* dialog box first opens, neither a DoMethod nor a receiver is selected. The DoMethods and receivers that appear in the dialog box depend on the method type that is being created or edited. For example, variables can function as receivers in Current Subapplication Methods but not in General Methods.

There are two ways to select the DoMethod that you want to add to the method you are writing:

- By first selecting a DoMethod from the *Method* list, and then selecting a receiver. In most cases only one receiver is available and is automatically selected.
- By first selecting a receiver from the *Executed by* list and then selecting a DoMethod from the *Method* list.

To select a DoMethod:

1. If the DoMethod you want to select is not in the *Method* list, select *Full List* from the list type list box.
2. Choose one of the following procedures:
   Select a DoMethod from the *Method* list, then select a receiver from the *Executed by* list. In most cases only one receiver is available and is automatically selected.

   -or-

   Select a receiver from the *Executed By* list and then select a DoMethod from the *Method* list.

   -or-

   Click the *Reference* button to insert a reference to an existing method line. The reference is inserted into the *Executed by* field.

   For example:

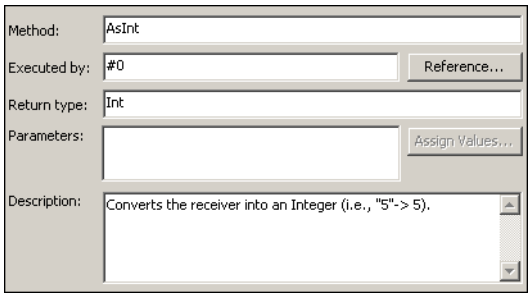


Figure 178. Inserting a reference to another method line

3. Select a DoMethod from the *Method* list.

4 If the selected DoMethod has parameters, the *Assign Values* button is enabled. Press the *Assign Values* button. The *Method Parameters* dialog box opens. See the following section for instructions on assigning parameter values.

5 Click *OK*.

Assigning DoMethod Parameters

A DoMethod may have parameters. In such a case the *Parameters* field displays the name of the parameters needed for the selected DoMethod, and the *Assign Values* button is enabled.

For example, let us examine the *DoesVariableExist* DoMethod, executed by the Subapplication, which checks whether a specific variable exists in the current Subapplication. This method has one parameter, varName, for specifying the variable name:

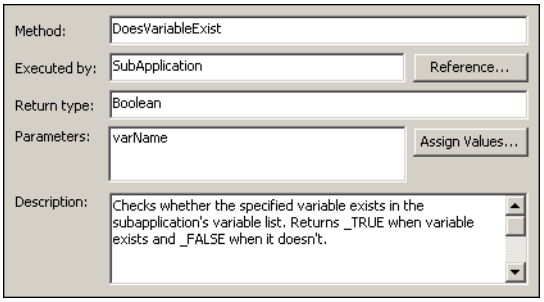


Figure 179. Assigning DoMethod parameters

Pressing the *Assign Values* button opens the *Method Parameters* dialog box:

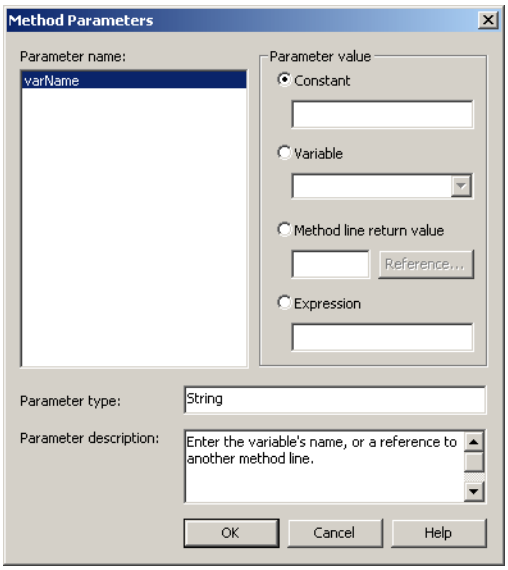


Figure 180. Method Parameters dialog box

Parameter name

Parameter Name lists the names of the parameters that need to be passed to the activated method.

Parameter value

Parameter values can be one of the following types:

- *Constant* - a constant value which you type in.

  The type of the field changes according to the parameter type.
- *Variable* - use the ... button to open the *Variable Lookup* dialog box, in which you select a Subapplication variable.

  If the method is a Current Subapplication Method, instead of the ... button, a combo box containing the Subapplication variables is displayed.
- *Method line return value* - use the *Reference* button to insert a reference to an existing method line. For more information, see "Referencing a Method Line from Another Method Line" on page 394.
- *Expression* - type in the desired expression.

Parameter type

Parameter Type displays the type of the currently selected parameter - boolean, character (char), integer (int), and string:

Table 24. Parameter types

Parameter Type	Field Type
Boolean	A check box with text that changes according to the value: "- True" or "- False".
Character	An edit field that enables you to type only one character.
Integer	A spin control in which you enter the value.
String	An edit field in which you type the string.

Parameter description

Parameter description gives a short description of the parameter that is currently selected in the *Parameter name* listbox.

To assign values to parameters, for each parameter in the *Method Parameters* dialog box:

1. Select the parameter in the *Parameter name* list.
2. Set the parameter's value.
3. When all parameter values have been specified, click *OK*.

> **Note:** When you type a Subapplication name as a parameter value, the Subapplication name must be in upper case letters. When a method has parameters, you cannot click the *OK* button in the *DoMethod: Method Activation* dialog box, before specifying the value of each parameter in the *Method Parameters* dialog box.

Customizing the Commonly Used Method List

The *Commonly Used DoMethod* list is customizable.

1. In the *DoMethod: Method Activation* dialog box, click *Advanced*. The *Define Commonly Used Methods* dialog box opens:

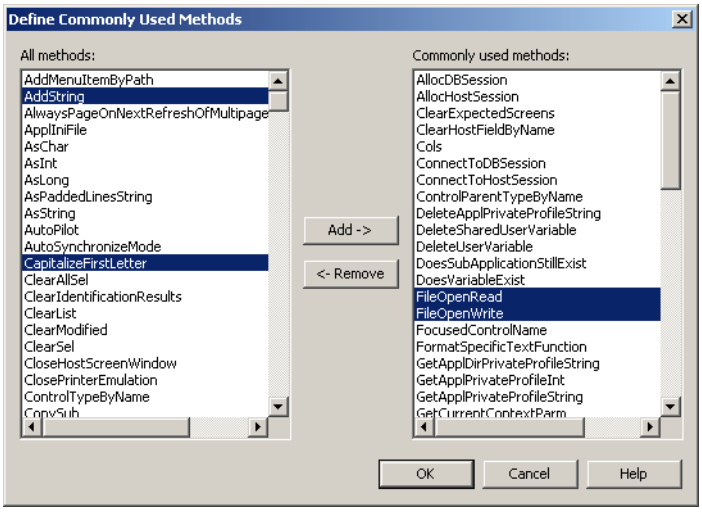


Figure 181. Define Commonly Used Methods dialog box

2. To add methods to the *Commonly used methods* list, in the *All methods* list, select the methods you want to add to the *Commonly used methods* list. Multiple selections are possible.

   Click the *Add* button.

   The methods are added to the *Commonly used methods* list.

3 To remove methods from the *Commonly used methods* list, in the *Commonly used methods* list, select the methods you want to remove from the list.

   Multiple selections are possible.

   Click the *Remove* button.

   The methods are placed back in the *All methods* list.

4 Click *OK* to save your changes.

> **Note:** Methods transferred from one list to the other remain selected after the transfer. Make sure to unselect them by clicking them, before transferring other methods in the opposite direction.

Enter Method: an Example

Now that you are familiar with the various method line types, the following section reviews the method lines in the *Enter* method in detail.

Enter is a General User-Triggered Method. It updates the host with the information typed in the GUI window, presses *Enter* on the host, and displays a new GUI window corresponding to the new screen called on the host.

The *Enter* Method is a Representation Definition component linked in the KnowledgeBase to the *OK* button, giving all *OK* buttons appearing in the Application the same functionality. The event that triggers the method is the user clicking the *OK* button or pressing *Enter* on the keyboard.

This is the way the *Enter* method looks in the *Define Method* dialog box:

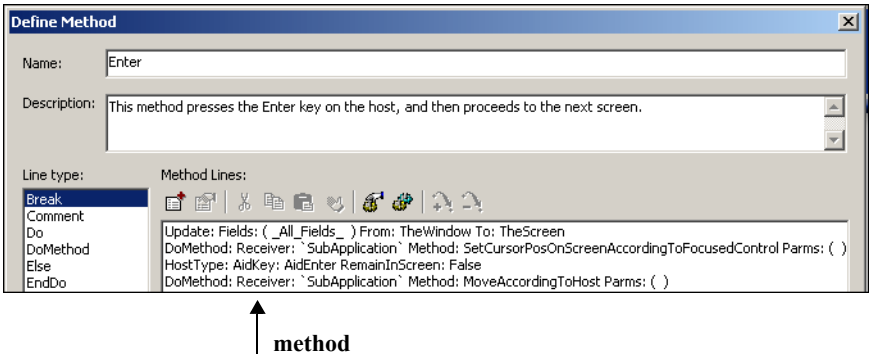


Figure 182. Enter method in the Define Method dialog box

The Method lines comprising the *Enter* method are:

Line 1:

```
Update: Fields (_All_Fields_) From: TheWindow To: TheScreen
```

The first line is an Update type method line. This line instructs the runtime to update all fields from the window to the screen.

Line 2:

```
DoMethod: Receiver: `SubApplication` Method:
SetCursorPosOnScreenAccordingToFocusedControl Parms: (  )
```

The second line is a DoMethod method line. It instructs the runtime to position the host screen cursor on the field which corresponds to the control that is currently in focus on the GUI.

Line 3:

```
HostType: AidKey: AidEnter RemainInScreen: False
```

The third line is a HostType method line. It instructs the runtime to press the *Enter* key on the host.

Line 4:

```
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: (  )
```

The last line is another DoMethod type method Line. It instructs the GUI to display the Subapplication that corresponds to the screen displayed on the host after the *Enter* key is pressed.

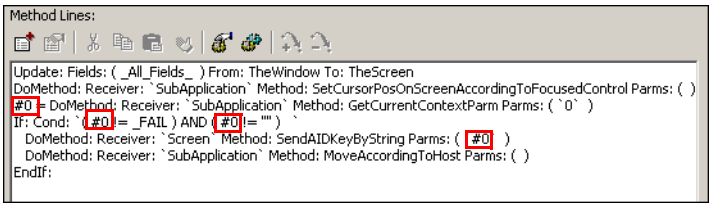
Referencing a Method Line from Another Method Line

Every method line returns a value. A method line can reference the value returned by another method line. At the time of writing the method, the explicit return value of the referenced line is not known. In runtime, the explicit value is used in the execution of the method.

In order to use the value returned by a method line, you first need to add a reference label to that method line. A reference label is added at the beginning of the method line. The label is composed of the # sign followed by a number. The label numbers begin at zero, and by default, increment by one each time. However, the Application developer can specify a different number if desired.

Example 52. Referencing one method line from another

In the following example of the *F_Key* General User-Triggered Method, only one line has a reference label (#0), but the value returned by the referenced line is used several times in the method.

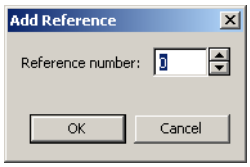


Reference labels to method lines can be added or removed in the *Define Method* dialog box, using the *Add Reference* and *Remove Reference* buttons in the toolbar.

Note: These options are also available from a shortcut menu that opens when you click the right mouse button within the method lines list.

To add a reference label to a method line:

1 Select a method line.
2 Click the *Add Reference* icon. The *Add Reference* dialog box opens:



3 By default, the first available number is displayed. You can enter a different reference number by typing or using the spin box arrows.
4 Click *OK*.
A reference label is added to the method line.

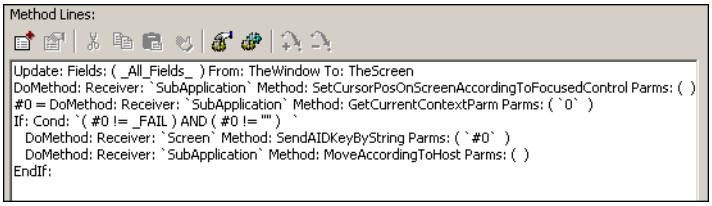
To remove a reference label from a method line:

1 Select a method line.
2 Click the *Remove Reference* icon.
The reference label is removed from the method line.

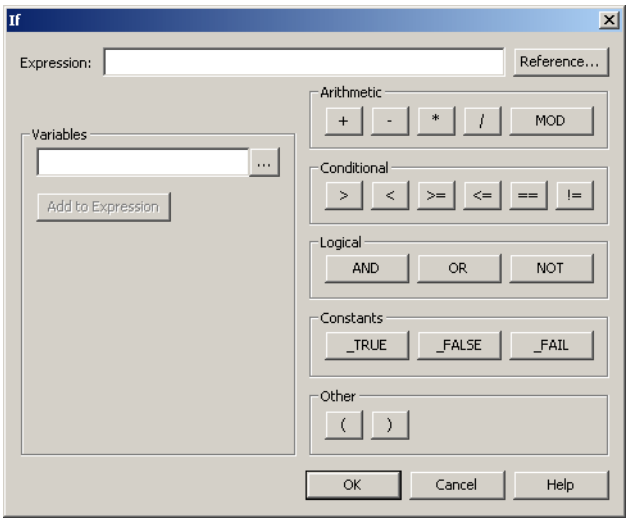
After adding a reference label to a method line, you can refer to the value returned by this method line in another method line. This is done in the method line type configuration dialog boxes. The configuration dialog boxes have a *Reference* button next to fields that can contain a reference to an existing method line. When you click the *Reference* button, a dialog box containing that method's method lines opens, and you can select the method line to reference. You can also add or remove reference labels if necessary.

To reference a method line:

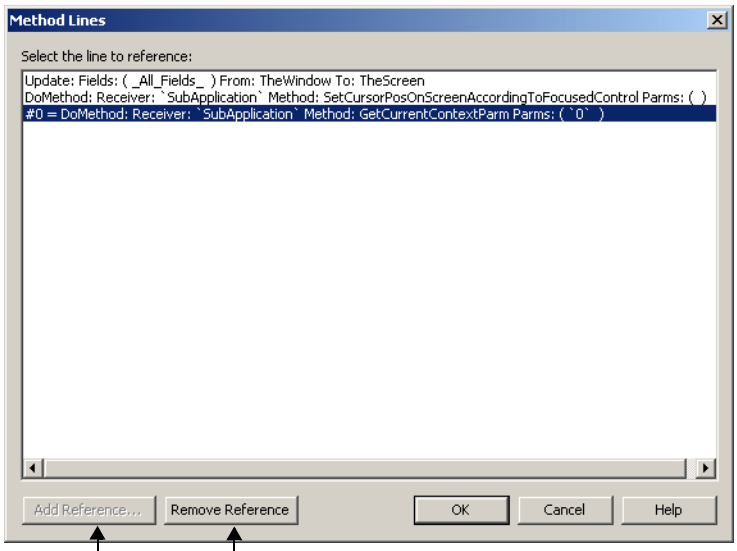
The following instructions refer to the process of writing the *F_Key* method. Let us assume we have already written the first three lines, and have added the reference label #0 to the third line:



1 Open the desired method line type configuration dialog. In this example, the *If* dialog box:



2 Click the *Reference* button. The following dialog box, containing only the method lines, opens:



Add Reference is enabled when a line without a label is

Remove Reference is enabled when the selected line has a reference label.

3 Use the *Add Reference* and *Remove Reference* buttons as necessary.

4 Select a method line with a reference label, and click *OK*.
The *Method Lines* dialog box closes, and the reference is inserted into the configuration dialog box.

In the *F_Key* example, the #0 label is added to the edit field:

Continue editing the method as necessary.

Note: After deleting a referenced method line, make sure that no method lines continue to refer to the deleted label.

Method Examples

This section contains several examples of the use of methods. In each example, first the method's purpose is described. Then, each method line is presented, followed by comments on its function.

Working with Variables

This method takes a value from a list column variable and types it in another field in the screen. It gets the value of the variable using its row and column positions.

DoMethods used:

- *DoesVariableExist*
- *WriteUserVariable*
- *GetHostCursorRow*, *GetHostCursorCol*, and *GetDataFromScreen*

```
DoMethod: Receiver: `SubApplication` Method: DoesVariableExist Parms:
(`"FIELD1"`  )
```

Checks that the variable named `FIELD1` exists in this Subapplication.

```
HostType: Field: FIELD1 MoveCursor: True AidKey: AidNone RemainInScreen:
False
```

Moves the cursor to the variable's field on the host screen.

```
Update: Fields: ( _All_Fields_  ) From: TheWindow To: TheScreen
```

Updates all fields to the host screen.

```
#0 = DoMethod: Receiver: `Screen` Method: GetHostCursorRow Parms: (  )
```

Returns the row position of the cursor on the host screen.

```
#1 = DoMethod: Receiver: `Screen` Method: GetHostCursorCol Parms: (  )
```

Returns the column position of the cursor on the host screen.

```
#2 = DoMethod: Receiver: `Screen` Method: GetDataFromScreen Parms: ( `#0` ,
`#1` , `4` , `_FALSE`  )
```

Returns the data retrieved from the row and column cursor position on the host screen. The length of the data to retrieve, specified in the third parameter, is 4. The last parameter, _FALSE, indicates that the data should be copied without attribute bytes.

```
HostType: Text: `#2` Field: FIELD2 MoveCursor: False AidKey: AidPF10
RemainInScreen: False
```

In the host screen, type the value of FIELD1, returned by the line with the label #2, into FIELD2, and press the *PF10* key.

```
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: (  )
```

Display the next window according to the host screen.

Working with Messages

This method formats a message before displaying it in the DIL (Dynamic Information Line) and in a message box.

DoMethods used:

- *LastMessageGet*
- *FormatSpecificTextFunction*
- *WriteMsgToDIL*

```
#0 = DoMethod: Receiver: `Application` Method: LastMessageGet Parms: (  )
```

Returns the text of the last host message.

```
#1 = DoMethod: Receiver: `Application` Method: FormatSpecificTextFunction
Parms: ( `#0` , `"InitialCapsAlways"` , `""`  )
```

Formats the message text to have initial capital letters.

```
DoMethod: Receiver: `Application` Method: WriteMsgToDIL Parms: ( `#1`  )
```

Writes the formatted message to the DIL.

```
MsgBox: MsgBoxType: Error Message: `#1` Caption: `"Format Error"` Default: 1
```

Writes the formatted message to a message box.

```
Return: ReachContinue
```

Using Conditions

This method updates all the fields to the screen. If the field `"ACCOUNT"` does not contain a value, the user gets an error message.

```
Update: Fields: ( _All_Fields_  ) From: TheWindow To: TheScreen
```

Updates all fields to the screen.

```
#0 = Expression: Expr: `ACCOUNT`
```

Gets the value of the `ACCOUNT` variable.

If `ACCOUNT` does not contain a value, this line returns the value `_FAIL`.

> **Note:** `ACCOUNT` is not surrounded by quotation marks in the expression, and therefore the expression returns the value of the `ACCOUNT` variable, not the "`ACCOUNT`" string.

```
If: Cond: `( #0 !=  _FAIL )`
```

Checks whether `FIELD1` contains a value, i.e. the expression above is not equal to `_FAIL`.

```
DoMethod: Receiver: `SubApplication` Method:
SetCursorPosOnScreenAccordingToFocusedControl Parms: (  )
HostType: AidKey: AidEnter RemainInScreen: False
```

If `ACCOUNT` contains a value, sets the cursor according to the focused control and presses *Enter*.

```
Else:
MsgBox: MsgBoxType: Exclamation Message: `"You Must Enter an Entity"`
Caption: `"ENTITY SELECTION SCREEN"` Default: 1
```

If `ACCOUNT` does not contain a value, displays an error message in a message box.

```
EndIf:
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: (  )
```

Displays the next window according to the host screen.

Reading Values from *.ini Files

This method retrieves a setting from an *.ini file and inserts it into a field.

DoMethods used:

- *GetVarValueByName*
- *GetApplPrivateProfileString*

```
#0 = DoMethod: Receiver: `SubApplication` Method: GetVarValueByName Parms:
(`"ScrHeader"`  )
```

Returns the value of the variable named `ScrHeader`.

```
If: Cond: `#0 !=  _FAIL `
```

Checks whether `ScrHeader` contains a value.

```
#1 = DoMethod: Receiver: `System` Method: GetApplPrivateProfileString Parms:
( `"MySection"` , `"SetHeader"` , `"DEFAULT"`  )
```

If `ScrHeader` contains a value, the method retrieves a string from the <applname>.INI file, according to the following parameters:

- MySection: The name of the section in the *.ini file.
  In the *.ini file this name is surrounded by square brackets.
- SetHeader: The name of the key which contains the string value to be returned by the method line. The format of an *.ini file entry is 'key=value'.
- DEFAULT is the default value to return in case a string is not found.

`Expression: Expr: `#1` SetVar: ScrHeader`

Sets the `ScrHeader` field variable to the string returned by the method line above.

`Update: Fields: ( ScrHeader  ) From: TheSubApp To: TheWindow`

Updates the field's value from the Manager to the Window, i.e. displays it on the GUI.

`EndIf:`

Pressing a Host Key

There are four DoMethods that you can use to "press" a key on the host. These DoMethods are:

- *SendAIDKeyByString*
- *SendKeyByString*
- *SendASpecificKey*
- *SendKeysWithoutReset*

Each of these DoMethods requires you to supply a parameter which specifies the host key to be pressed.

The DoMethods *SendAIDKeyByString* and *SendKeyByString* take string parameters that are based on the name of the key.

The DoMethods *SendASpecificKey* and *SendKeysWithoutReset* take numerical parameters.

Table of Host Keys

The following table lists host keys and the numerical parameter for "pressing" the key with *SendASpecificKey* or *SendKeysWithoutReset*. In addition, the table also specifies the keys that can also be "pressed" by means of *SendAIDKeyByString* and *SendKeyByString* DoMethods, and the string parameters to use.

For many keys, more than one string parameter is allowed. In such cases, the allowed strings are separated by commas.

Table 25. Table of host keys (Sheet 1 of 5)

Host Key	DoMethod	Allowed Strings	Value
Attention	SendAIDKeyByString	Attention, ATTN, Attn	**1777**
Backspace			**1551**
Backtab	SendKeyByString	BACKTAB,BACK, BackTab, Back	**1208**
Clear	SendAIDKeyByString	Clear, CLEAR, clear	**1212**
Command			**1540**
Cursor Down	SendKeyByString	CursorDown, Cursor Down	**1240**
Cursor Left	SendKeyByString	CursorLeft, Cursor Left	**1237**
Cursor Right	SendKeyByString	CursorRight, Cursor Right	**1239**
Cursor Up	SendKeyByString	CursorUp, Cursor Up	**1238**
Cursr Select			**1788**
Delete	SendKeyByString	Del, Delete	**1246**
Dup	SendKeyByString	Dup, DUP	**1553**
End			**1235**
Enter	SendAIDKeyByString	Enter, ENTER, enter	**1213**

Table 25. Table of host keys (Sheet 2 of 5)

Host Key	DoMethod	Allowed Strings	Value
Erase EOF	SendKeyByString	EraseEOF, Erase EOF	**1781**
Escape			**1227**
F1	SendAIDKeyByString	1, 01, F1, F01, PF1	**1312**
F10	SendAIDKeyByString	10, F10, PF10	**1321**
F11	SendAIDKeyByString	11, F11, PF11	**1322**
F12	SendAIDKeyByString	12, F12, PF12	**1323**
F13	SendAIDKeyByString	13, F13, PF13	**1824**
F14	SendAIDKeyByString	14, F14, PF14	**1825**
F15	SendAIDKeyByString	15, F15, PF15	**1826**
F16	SendAIDKeyByString	16, F16, PF16	**1827**
F17	SendAIDKeyByString	17, F17, PF117	**1828**
F18	SendAIDKeyByString	18, F18, PF18	**1829**
F19	SendAIDKeyByString	19, F19, PF19	**1830**
F2	SendAIDKeyByString	2, 02, F2, F02, PF2	**1313**
F20	SendAIDKeyByString	20, F20, PF20	**1831**
F21	SendAIDKeyByString	21, F21, PF21	**1832**
F22	SendAIDKeyByString	22, F22, PF22	**1833**
F23	SendAIDKeyByString	23, F23, PF23	**1834**

Table 25. Table of host keys (Sheet 3 of 5)

Host Key	DoMethod	Allowed Strings	Value
F24	SendAIDKeyByString	24, F24, PF24	**1835**
F3	SendAIDKeyByString	3, 03, F3, F03, PF3	**1314**
F4	SendAIDKeyByString	4, 04, F4, F04, PF4	**1315**
F5	SendAIDKeyByString	5, 05, F5, F05, PF5	**1316**
F6	SendAIDKeyByString	6, 06, F6, F06, PF6	**1317**
F7	SendAIDKeyByString	7, 07, F7, F07, PF7	**1318**
F8	SendAIDKeyByString	8, 08, F8, F08, PF8	**1319**
F9	SendAIDKeyByString	9, 09, F9, F09, PF9	**1320**
Field Exit	SendKeyByString	FieldExit, Field Exit	**1800**
Field Mark	SendKeyByString	FieldMark, Field Mark	**1554**
Field Minus	SendKeyByString	FieldSubtract, Field Subtract, FieldMinus, Field Minus	**1542**
Field Plus			**1543**
Help	SendAIDKeyByString	HELP, Help	**1247**
Hexadecimal			**1545**
Home	SendKeyByString	HOME, Home	**1236**
Insert			**1245**

Table 25. Table of host keys (Sheet 4 of 5)

Host Key	DoMethod	Allowed Strings	Value
Insert Toggle			**1546**
Local Print			**1547**
New Line	SendKeyByString	NewLine, New Line	**1790**
PA1	SendAIDKeyByString	PA1	**1761**
PA2	SendAIDKeyByString	PA2	**1762**
PA3	SendAIDKeyByString	PA3	**1763**
Page Down	SendAIDKeyByString	Dn, DN, Down, Page Down, PageDown, RollUp, Roll Up	**1234**
Page Up	SendAIDKeyByString	Up, UP, Page Up, PageUp, RollDwn, RollDown, Roll Down	**1233**
PC Print			**1550**
Print			**1242**
Reset	SendKeyByString	RESET, Reset	**1794**
ScreenPrint			**1503**
Separator			**1308**
System Request	SendAIDKeyByString	SysReq, System Request, SystemRequest	**1795**

Table 25. Table of host keys (Sheet 5 of 5)

Host Key	DoMethod	Allowed Strings	Value
Tab	SendKeyByString	TAB, Tab	**1209**
Test			**1552**

The numerical parameters to be used with *SendASpecificKey* and *SendKeysWithoutReset* are for the “bare” key. The values for a bare key plus a system key such as “ALT”, “CTRL” or “SHIFT” are calculated as follows:

- ALT+Host_Key add 256 to the bare value.
- CTRL+Host_Key add 512 to the bare value.
- SHIFT+Host_Key add 1024 to the bare value.

You can also “press” more than one system key.

- To press ALT+CTRL add 256+512 to the bare value.
- To press ALT+CTR+SHIFT add 256+512+1024 to the bare value.

Chapter 14. Methods: System-Triggered Methods List

This chapter lists the available System-Triggered Methods and provides the information necessary to modify them.

This chapter describes:

- Return Values

For each System-Triggered Method, the following categories are listed:

Table 26. Categories for system-triggered methods in this section

Category	Description
Activating event	Which event activates the method.
Default return value	This is the value returned by the unmodified method. Modifying the value may necessitate modifying the return value.
Default behavior	This specifies what the unmodified method does. The method may do nothing except return the default return value or it may have the specific behavior mentioned in this category.
Modifying options and uses	This details how to change the method and for what possible uses.

Return Values

System-Triggered Methods are divided in those that must return a value and those that do not return any value. Methods that must return a value, return either:

- A boolean value - TRUE or FALSE.

  -or-
- A SUCCESS/ NO VALUE or FAIL value.

  "FAIL on Fail of:" is like FAIL, but it tests a condition first. It will only return FAIL if a certain condition is met.

When modifying a System-Triggered Method the following rules must be kept:

- For methods that must return a value, specify the value in the method lines. Choose a return value from the Return method line type.
- The return value can be changed, but not the return value type.
- For methods that do not return any value, do not add a Return method line.
- Some System-Triggered Methods must always return the same return value. When applicable, this is indicated in the description of the method's default return value.

An example of a method that must return a value is the *UserAcceptScreen* method.

Method description in the System-Triggered Method manager

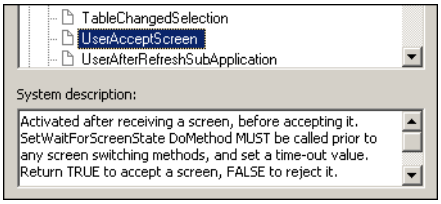


Figure 183. UserAcceptScreen method

AfterPageUpDown

Activating event	This method is activated *after* a Page Up or a Page Down command and *before* the new page is displayed.
Default return value	This method returns FALSE.
Default behavior	The method has no specific behavior. It returns the default return value.
Modifying options	When this method returns TRUE instead of FALSE, the window is refreshed.

GetToBottomOfList

Activating event	This method is activated when the user clicks a *Bottom of List* button (if it exists on the interface).
Default return value	This method returns FALSE.
Default behavior	The method has no specific behavior. It returns the default return value.

Modifying options

Changing the return value to TRUE, executes instructions written in the method.
Instructions necessary to get to the bottom of a list from within the list are usually Application-dependent. This method provides the means to write the required instructions and execute them at the appropriate time.

GetToTopOfList

Activating event

This method is activated when the user clicks a *Top of List* button (if it exists on the interface).

Default return value

This method returns FALSE.

Default behavior

The method has no specific behavior. It returns the default return value.

Modifying options

Changing the return value to TRUE, executes instructions written in the method.
Instructions necessary to get to the top of a list from within the list are usually Application-dependent. This method provides the means to write the required instructions and execute them at the appropriate time.

PageDown

Activating event

This method is activated when the user clicks a list's lower scrollbar.

Default return value

This method does not return a value.

Default behavior

This method presses the *Page Down* key for AS/400 applications and the *F8* key for mainframe applications.

Modifying options Use this method to set the keys to be pressed for scrolling down the list, if they are not *Page Down* or *F6*.

Note: This method is not recommended for defining the scroll keys. Use the *List* wizard instead.

PageUp

Activating event This method is activated when the user clicks a list's upper scrollbar.

Default return value This method does not return a value.

Default behavior This method presses the *Page Up* key for AS/400 applications and the *F7* key for mainframe applications.

Modifying options Use this method to set the keys to be pressed for scrolling the list up, if they are not *Page Up* or *F7*.

Note: This method is not recommended for defining the scroll keys. Use the *List* wizard instead.

TableChangedSelection

Activating event This method is activated when one of the following events occur:

- Upon initial entry into a Subapplication.
- A table gets or loses focus.
- The selection in a table is changed.

Default return value

This method does not return a value.

Default behavior

The method has no specific behavior. It returns the default return value.

Modifying options

Use this method to write instructions that must be executed when the table gets or loses focus or when the selection on the table changes.

The method is used to disable or enable Menu Items and Buttons when certain user conditions are met.

UserAcceptScreen

Activating event

This method is activated immediately after the server receives a screen from the host and before this screen is displayed on the end-user's browser.

Default return value

This method returns either TRUE or FALSE, depending on the current screen sent by the host, and the parameters defined in the *SetWaitForScreenState* DoMethod.

Default behavior

When the method returns TRUE, the current screen is accepted and displayed on the end-user's browser.

If the method returns FALSE, the current screen is ignored.

The JIS Server continues to try to accept a screen, until a timeout is reached. When the timeout is reached, the last screen that was sent by the host is accepted by the server and displayed to the end user.

Modifying options

The *UserAcceptScreen* method is used in conjunction with the parameters of a DoMethod named *SetWaitForScreenState*:

screenList - a list of screens either to be accepted or to be ignored, depending on the value of the 'accept' parameter.

accept - a boolean parameter that indicates whether to accept or ignore the screens listed in 'screenList'.

timeout - the timeout for the server, in milliseconds, after which the last host screen is accepted by the server.

To retrieve the values of the *SetWaitForScreenState* parameters, use the *GetCurrentContextParm* DoMethod. The *GetCurrentContextParm* method has one parameter, parmNumber, which can have one of four values:

- 0 - returns the name of the current host screen
- 1 - returns the list of screens from the 'screenList' parameter
- 2 - returns the value of the 'accept' parameter
- 3 - returns the value of the 'timeout' parameter

UserAfterRefreshSubApplication

Activating event

This method is activated every time the runtime refreshes the Subapplication. It is activated immediately *after* the refresh takes place.

Default return value

This method must return a TRUE value. No other return value is possible.

Default behavior

The method has no specific behavior. It returns the default return value.

Modifying options

UserAfterRefreshSubApplication is the preferred method for changing field values without ACE overwriting the changes when it updates fields.

As ACE acts at all times to keep the GUI in synchronization with the host application, ACE must act when the host application moves from screen to screen and when the screen's identity doesn't change, but there *is* a change to a host field. When a field changes but the screen's identity doesn't change ACE acts by refreshing all the GUI elements that depend on host screen information.

When you override the host information for some control, your changes will be lost whenever ACE refreshes the controls from the host. In this situation, it is important that following a refresh, your override is applied.
Use *UserAfterRefreshSubApplication* to ensure that your override follows a refresh.

Note: Do not use the DoMethod line *RefreshSubApplication* in this method. This would cause the runtime to enter an infinite loop.

UserAfterTabFolderChanged

Activating event

This method is related to tab controls navigation. When the user clicks a tab folder to switch from one tab to another, it is activated immediately *after* the switch.

Default return value

This method does not return a value.

Default behavior

The method has no specific behavior. It returns the default return value.

Modifying options

Use this method to write instructions that must be executed after a tab switch.

UserAfterTableAction

Activating event	This method is activated after the processing of every table action.
Default return value	This method returns TRUE.
Default behavior	The method has no specific behavior. It returns the default return value.
Modifying options	Use this method to write instructions to be executed after one of the following table actions: CellChanged, PageUp, PageDown, DoubleClick, PromptClick, MoveToTop, MoveToBottom. To get the action type, check the value of the user variable _internal_TableAction.

Note: Do not use the *MoveAccordingToHost* DoMethod inside this Method.

UserAttentionRequest

Activating event	This method is activated when the System Attention button is clicked in a runtime environment window.
Default return value	This method does not return a value.
Default behavior	This method displays the System request screen.
Modifying options	Use this method to insert instructions that are called up when the System Attention button is clicked in a runtime environment window.

Note: This method is valid for AS/400 hosts only.

UserBack

Activating event

- The user attempts to close a popup window.
- The *UserShouldCloseSubApplWindow* method was called and returned TRUE.

This method is activated when *both* events occur

Default return value

This method returns SUCCESS/ NO VALUE.

Default behavior

This method sends the correct Fkey needed to close the popup window and refresh. The default values are:

- F12 for an AS/400.
- Clear for CICS.
- F3 for other host operating systems.

Modifying options

Use this method to customize the key that must be sent to the host to close the popup and refresh.

UserBeforeTabFolderChanged

Activating event

This method is related to tab controls navigation. When the user clicks a tab folder to switch from one tab to another, it is activated immediately *before* the switch.

Default return value

This method returns TRUE.

Default behavior

The tab switch is allowed to take place.

Modifying options

When the return value is FALSE, the tab switch does not take place.

Use this method to allow the user to switch tabs only when certain conditions are met.

UserBeforeTableAction

Activating event

This method is activated before the processing of every table action.

Default return value

This method returns TRUE.

Default behavior

The method has no specific behavior. It returns the default return value.

Modifying options

Use this method to write instructions to be executed before one of the following table actions: CellChanged, PageUp, PageDown, DoubleClick, PromptClick, MoveToTop, MoveToBottom.

To get the action type, check the value of the user variable _internal_TableAction.

Note: Do *not* use the *MoveAccordingToHost* DoMethod inside this method.

UserCloseSubApplWindow

Activating event

This method is activated when the user attempts to close a Subapplication window or a popup window.

Default return value

The method returns TRUE.

Default behavior

When the return value is TRUE, nothing happens.

Modifying options

The modifying options for this method make use of another System-Triggered method, the *UserShouldCloseSubApplWindow* method.

If the return value is changed to FALSE, the *UserShouldCloseSubApplWindow* method is called.

If that method returns FALSE: nothing happens - the window does not close.

If the method returns TRUE while the user is trying to close a popup window, the System-Triggered Method *UserBack* is called which determines the proper FKey sent to the host to close the popup and refresh.

If the method returns TRUE while the user is trying to close a regular Subapplication window, the runtime exits. A confirmation message is or is not be displayed, depending on the <ApplName>.ini file setting.

See the description of "UserShouldCloseSubApplWindow" on page 426 for exact details on how to use the method.

UserDestroyApplication

Activating event

This System-Triggered method is called when the session is closed but before the connection to the host is closed and before the data structures of the session are destroyed.

The method is executed in any of the following cases:

- Free Session is called for an XML session.
- An XML session is closed due to idle timeout.
- A Java session is closed by the client.
- All the sessions are closed when the server quits.

If the session was closed as a result of an error or exception, the method is not executed because the session might not be in a stable state.

Default return value

This method does not return a value. Do not use a return with this method.

Default behavior

The method has no specific behavior. It returns the default return value.

Modifying options

This method lets you perform cleanup just before the session is closed. It can be used to perform an orderly logoff from the host, for closing database connections, etc. By default this method does nothing.

Receiver: SubApplication.

Can be defined in the level of Application/Library/SubApplication.

UserDestroySubApplication

Activating event

This method is activated immediately *before* a Subapplication closes. It is not activated when a popup window closes.

Default return value

This method does not return a value. Do not use a return with this method.

Default behavior

The method has no specific behavior. It returns the default return value.

Modifying options

Write instructions in this method that must be executed when the Subapplication closes.

For example, use it for clean-up purposes (e.g. to delete a variable from memory).

UserHostHelpRequest

Activating event

This method is activated when the host *Help* button is clicked in a runtime environment window.

Default return value	This method does not return a value.
Default behavior	This method presses *Help* on the host and moves to the next screen.
Modifying options	Use this method to insert instructions that are called up when the host *Help* button is clicked in a runtime environment window.

Note: This method is valid for AS/400 hosts only.

UserHostMessageHelpRequest

Activating event	This method is activated by a click on the *Help* button in the DIL.
Default return value	This method does not return a value.
Default behavior	This method presses *Message Help* on the host and moves to the next screen.
Modifying options	Use this method to insert instructions that will be called up when the *Help* button is clicked in the DIL.

Note: This method is valid for AS/400 hosts only.

UserInitApplication

Activating event	This is the first method activated when running an Application, before the connection to the server is established.
Default return value	This method does not return a value.
Default behavior	The method has no specific behavior. It returns the default return value.
Modifying options	The method can be used to perform specific preparatory operations on the Application level. For example, set various emulator settings before connecting to the host.

UserInitBeforeFirstSubAppl

Activating event	This method is activated *once* upon initial entry to an Application, before the first Subapplication is processed.
Default return value	This method must return a TRUE value. No other return value is possible.
Default behavior	The method has no specific behavior. It returns the default return value.
Modifying options	The Application developer can use this method to perform specific preparatory operations on the Application level, before any Subapplication is processed. Examples: Read an instruction from an *.ini file, write an instruction to an *.ini file, or to solve screen identification problems by using *SetExpectedScreenForeverByName*.

UserInitSubApplication

Activating event	This method is activated every time a regular Subapplication is entered, but *not* when the Subapplication is refreshed after a popup window closes.
Default return value	This method returns SUCCESS/ NO VALUE.
Default behavior	The method has no specific behavior. It returns the default return value.
Modifying options	Write instructions in this method that must be executed when the Subapplication opens. For example, compose a new window caption from several different fields on the screen.

UserIsRealMessage

Activating event	This method is activated when the runtime identifies a message on the host.
Default return value	The method returns either TRUE or FALSE, depending on whether the message fulfills the conditions written in the method or not.
Default behavior	When the return value is TRUE, the message is displayed in the DIL. When it is FALSE, the message is not displayed.

Modifying options

ACE automatically displays data from the host application's message lines in the GUI window DIL. Occasionally, non-message information is displayed on these lines. This happens primarily when information on a screen is dynamic or information is not processed in the converter. ACE cannot discern between messages and non-message information and therefore displays any information that appears in the message lines in the DIL. When the user scrolls through the messages in the DIL, non-relevant information is received.

Add a method to the Application that distinguishes between message and non-message information. When this method is implemented ACE verifies each message line individually to determine if it is a real message and then saves all messages that are real to a buffer. You can scroll through the messages in the buffer by clicking the buttons on the window DIL. The

method returns TRUE when the message is real and the message then goes through the Message Handling process. If the message is not real, the method returns FALSE and the non-message information is ignored.

UserMoveToDependentScreen

Activating event

This method is activated in a Many-to-One, when the Principal moves between the Dependent screens.

Default return value

This method must return a TRUE value. No other return value is possible.

Default behavior

Moves the host to the Dependent screens.

Modifying options

It is strongly recommended *not* to modify this method. *UserMoveToDependentScreen* is specific to Many-to-One Subapplications. See the chapter entitled "Many-to-One" in *JIS Interface Server: Advanced Topics*.

UserMWIRequest

Activating event	This method is activated by clicking on the Message Waiting Indicator button on the DIL.
Default return value	This method returns SUCCESS/ NO VALUE.
Default behavior	This method performs a *FlatMoveTo* to the messages window.
Modifying options	Use this method to insert instructions that are called up at that particular time.

Note: This method is valid for AS/400 hosts only.

UserPreUpdateInitSubAppl

Activating event	This method is activated every time a Subapplication is entered, but before information is updated from the Screen to the Manager.
Default return value	This method returns SUCCESS/ NO VALUE.
Default behavior	The method has no specific behavior. It returns the default return value.

Modifying options

This method is executed before *UserInitSubApplication* and *UserSkipSubApplication*.

Use this method to do things that cannot be done at the *UserInitSubApplication* stage. For example, reading a client-shared user variable and writing it to an *.ini file so that it takes effect in the current screen.

Note: Do *not* use the *MoveAccordingToHost* DoMethod inside this Method.

UserRefreshSubApplication

Activating event

This method is activated every time the runtime refreshes the Subapplication. It is activated immediately *before* the refresh takes place.

Default return value

This method returns FALSE.

Default behavior

The unmodified method lets the runtime refresh the Subapplication.

Modifying options

The runtime does not refresh the Subapplication when the return value is changed to TRUE. In this case, it is necessary to write the method that refreshes the controls.

If you write new method lines but leave the return value as FALSE, the runtime carries out the instructions and then the refresh.

The *UserRefreshSubApplication* method allows changing field values without ACE overwriting your changes when it updates fields.

Note: When modifying this method, it is recommended to end the method lines with return FALSE, allowing ACE to refresh the Subapplication. Do not use the *RefreshSubApplication* DoMethod in this method.

UserServerDataReady

Activating event	This method is activated after the data is ready at the server just before updating it to the client, both during initialization and during refresh of the Subapplication.
Default return value	This method does not return a value. Do not use a return for this method.
Default behavior	The method has no specific behavior.
Modifying options	Use before the methods *UserInitSubApplication* or *UserAfterRefreshSubApplication* to update control properties before the data is updated to the client. This method allows you to update a control property without having to do an "Update Manager to Window" afterwards.

UserShouldCloseSubApplWindow

Activating event	This method is activated when the user attempts to exit the Application or when the user attempts to close a Subapplication window (popup or MDI client) AND *UserCloseSubApplWindow* returns a FALSE value.
Default return value	The method returns TRUE.
Default behavior	When the return value is TRUE, the Application is exited. A Yes/No message box asking the user whether he wants to exit the Application may or may not be displayed, depending on the <ApplName>.ini setting. Answering Yes closes the runtime.

Modifying options

When the return value is changed to FALSE, the Application does not close. In this case, it is necessary to write the method to give the runtime instructions on how to continue.

There are cases when a runtime window should not close immediately. For example, when a special validity check has to be executed before closure, such as verifying that all fields in a window have been correctly filled.
In such a case, the Application developer adds the required validity check, and the user cannot close the runtime before the check is completed.

UserShouldWindowBeBuilt

Activating event

This method is activated upon entry into a Subapplication of the "Conditionally display window" type.

Default return value

This method returns TRUE.

Default behavior

The unmodified method allows the runtime to build the Subapplication's window.

Modifying options

When the return value is changed to FALSE, the method does not allow the runtime to build the window.

This System-Triggered Method is used for skipping window-display in runtime. See the discussion of skipping window-display at runtime in *JIS Interface Server: Advanced Topics*.

UserSkipSubApplication

Activating event

This method is activated after the Subapplication and its variables have been built.

Default return value

This method does not return a value.

Default behavior

The method has no specific behavior.

Modifying options

This method is used for skipping window-display in runtime.

For a 'Never Display Window' type Subapplication, the method should contain instructions for moving to the next Subapplication, and return TRUE, so that the current Subapplication window is never displayed.

For a 'Conditionally Display Window' type Subapplication, the contents and the return value of this method depend on the value returned by the *UserShouldWindowBeBuilt* method. For more information, see the discussion of skipping window-display at runtime in *JIS Interface Server: Advanced Topics*.

Chapter 15. Methods: DoMethods List

ACE has more than 260 DoMethods you can use in your methods. This chapter presents all the DoMethods by their name, together with a short description and the return type of the DoMethod line type based on the DoMethod.

Return Types

There are several kinds of return types:

- *Void:* the DoMethod has no special return type. This is similar to the Success/No Value return value of a method line.
- *Integer:* the DoMethod returns an integer.
- *Character:* the DoMethod returns a character.
- *String:* the DoMethod returns a string.
- *Boolean:* the DoMethod returns _TRUE or _FALSE.

Table 27. DoMethods (Sheet 1 of 34)

DoMethod name	Description	Return type
AddMenuItemByPath	Adds the window menu or submenu item defined in the parameter, to the bottom of the current floating menu. Returns the position (0-based index) of the menu item added to the current floating menu.	**Integer**
AddString	Adds a string to the list box/combo box and returns its index.	**Integer**
AllocDBSession	Establishes a connection to an external database.	**DBSession**
AllocHostSession	Establishes a connection to the host and allocates a new session	**JcsSession**
AlwaysPageOnNext RefreshOf MultipageLists	Refreshes the host's multipage list even when there has not been any updating, to allow paging. This method supports the multipage table feature.	**Void**
AppendChild	Appends a child node to an element. When appended to an empty document, it is the root element. Use in DOM-based XML methods.	**Boolean**

Table 27. DoMethods (Sheet 2 of 34)

DoMethod name	Description	Return type
AppendTextChild	Inserts the text specified in the parameter into a node. Use in DOM-based XML methods.	**Boolean**
ApplIniFile	Returns the Application's <applname>.ini file name.	**String**
AsChar	Returns the character that corresponds in the ASCII coding system to the receiver's numerical value. In a string starting with a number, only the numerical part is interpreted.	**Character**
AsInt	Converts the receiver into an integer, i.e. "5"-> 5.	**Integer**
AsLong	Converts the receiver into a long number.	**Integer**
AsPaddedLinesString	Returns a formatted string in which new line characters have been introduced in such a manner as to fit the string into a fixed width area.	**String**
AsString	Converts the receiver into a string, i.e. 5 -> "5".	**String**
AutoSynchronizeMode	Checks whether to automatically synchronize on each new screen. Returns _TRUE or _FALSE according to the answer.	**Boolean**
CapitalizeFirstLetter	Capitalizes the receiver's first letter and returns the modified string.	**String**

Table 27. DoMethods (Sheet 3 of 34)

DoMethod name	Description	Return type
ClearAllSel	Deselects all selected items in a multiple listbox. This DoMethod has no return value.	**Void**
ClearExpectedScreens	Clears the buffer of expected screen names. It does not clear expected screens set by *SetExpectedScreenForever ByName*. This DoMethod has no return value.	**Void**
ClearHostFieldByNam e	Clears a field's content on the host. This DoMethod has no return value.	**Void**
ClearIdentification Results	Clears the last identification results. Forces the next identification to execute even without host modifications. This DoMethod has no return value.	**Void**
ClearList	Clears the contents of the listbox/ combobox. This DoMethod has no return value.	**Void**
ClearModified	Clears all the window control's modification bits. This DoMethod has no return value.	**Void**
ClearSel	Deselects a selected item in a multiple listbox. This DoMethod has no return value.	**Void**
CloseHostScreen Window	Closes the host screen window if it is open, otherwise does nothing. This DoMethod has no return value.	**Void**

Table 27. DoMethods (Sheet 4 of 34)

DoMethod name	Description	Return type
ClosePrinter Emulation	Closes the printer emulation session. This DoMethod has no return value.	**Void**
Cols	Returns the number of columns on the host screen.	**Integer**
ConnectToDBSession	Retrieves a previously allocated DB session.	**DBSession**
ConnectToHostSession	Retrieves a previously allocated host session.	**JcsSession**
ControlParentTypeBy Name	Returns the name of the control's parent type (static, textbox, etc.). For example, when a textbox control is within a table, returns table. Returns _FAIL in case of failure.	**String**
ControlTypeByName	Returns the name of the control type (static, textbox, etc.). Returns _FAIL in case of failure.	**String**
CopySub	Inserts part of the source string, defined in the parameters, in the receiver string. Returns the modified string. The parameters define what part of the source string is selected for insertion and where it is inserted in the receiver string.	**String**
CreateElement	Creates a new XML element. Use in DOM-based XML methods.	**XML Element**

Table 27. DoMethods (Sheet 5 of 34)

DoMethod name	Description	Return type
DebugPrint	Writes a message to the JIS Server log for debugging purposes. Messages are only written to the log when the DebugLevel parameter is equal to or lower than the server's debug level.	**Void**
DefaultText	Returns the static default text.	**String**
DeleteApplPrivateProfileString	Deletes a string from the <applname>.INI file. Returns _TRUE when the field is protected and _FALSE when it is not.	**Boolean**
DeleteSharedUserVariable	Deletes a variable from the variables pool shared between client and server. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
DeleteString	Deletes a string from the listbox/combobox. This DoMethod has no return value.	**Void**
DeleteUserVariable	Deletes a variable from the variables pool. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
Disable	Disables the receiver.	**Boolean**

Table 27. DoMethods (Sheet 6 of 34)

DoMethod name	Description	Return type
DoesSubApplicationStill Exist	Checks whether the Subapplication associated with this screen still exists. The Subapplication may no longer exist after operations such as *Terminate* and *MoveAccordingToHost*. Returns _TRUE when the Subapplication exists and _FALSE when it does not.	**Boolean**
DoesVariableExist	Checks whether the specified variable exists in the Subapplication variable list. Returns _TRUE when variable exists and _FALSE when it does not.	**Boolean**
DoubleClickTable	This method reads the default selection character from the INI file - specific to the subapplication or global, and writes it to the list selection column according to the selected lines in the Table control. It should only be attached to a Table control. The default selection character is defined in the [DoubleClickTable] section of the runtime INI file, or, alternatively, in the jacadasv.ini file.	**Void**
Dup	Duplicates the receiver and returns the duplicated string.	**String**
DuplicateNode	Adds a brother node of the JIS Name type specified in the parameter to a document. Use in template-based XML methods.	**Boolean**

Table 27. DoMethods (Sheet 7 of 34)

DoMethod name	Description	Return type
Enable	Enables the receiver. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
ExecuteMethod	In an external JcsSession, executes an ACE method that has no parameters on the JIS Server. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
ExecuteQuery	Executes an SQL statement. Returns either the row count for INSERT, UPDATE or DELETE, or 0 for SQL statements that return nothing. Returns -1 in case of failure.	**Int**
FileOpenRead	Builds an input stream object from a file. Use this method to read the content of a file.	**JSInput Stream**
FileOpenWrite	Maps a file on the disk as an output stream object in order to write the output to a file.	**JSOutput Stream**
FindFrom	Checks whether a character exists in a string. Returns the location (0-based index) of the first occurrence of the character, or -1 if the character is not found.	**Integer**
FindFromInt	Similar to the *FindFrom* method but not recommended for use.	**Integer**

Table 27. DoMethods (Sheet 8 of 34)

DoMethod name	Description	Return type
FindNotFrom	Finds where the receiver string has a character different from a given character. Returns the position (0-based index) of the character's first occurrence. If the search is unsuccessful, returns the null position at the end of the string.	**Integer**
FindSubSet	Returns the position, within a given receiver, of the specified string's starting point.	**Integer**
FindSubSetInt	Looks for and returns the start position (0-based index) of a specific sub-string in the receiver. If the sub-string is not found, returns -1.	**Integer**
FlatMoveToNextFull Screen	Performs a flat *MoveToNextFullScreen*.	**Boolean**
FlipCheck	Toggles the check state of the checkbox. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
FocusedControlName	Returns the variable's name of the control in focus. Works only on variables with checked Runtime Data Flow arrows in the Subapplication manager.	**String**
FormatSpecificText Function	Formats a string according to a given text formatting function. Returns _FAIL in case of failure.	**String**
Free	Frees a previously allocated JcsSession on a JIS Server. This DoMethod has no return value.	**Void**

Table 27. DoMethods (Sheet 9 of 34)

DoMethod name	Description	Return type
GetActive	Returns the number of the active tab (first tab's number is 0).	**Integer**
GetApplDirPrivateProfileString	Returns a string taken from the specified *.ini file. The *.ini file must reside in the Application's directory.	**String**
GetApplPrivateProfileInt	Returns a number taken from the <*ApplName*>*.ini* file.	**Integer**
GetApplPrivateProfileString	Returns a string taken from the <*ApplName*>*.ini* file.	**String**
GetAttribute	Returns the value of a node's attribute. The attribute's name is specified in the parameter. Use in DOM-based XML methods.	**String**
GetAutoReconnectToHostMode	Returns the value of the AutoReconnectToHost flag of the current running session.	**Boolean**
GetCellText	Returns the value of a table cell, according to the row and column variable name. You can also indicate whether to run formatting functions from the manager to the window. Returns NULL if the cell is out of the table boundaries.	**String**
GetCheck	Returns the check state of the checkbox, _FALSE when it is unchecked, _TRUE when it is checked.	**Boolean**

Table 27. DoMethods (Sheet 10 of 34)

DoMethod name	Description	Return type
GetChildNodes	Returns a list of the child XML elements of a node. Use in DOM-based XML methods.	**XML Element List**
GetClientIPAddress	Using code extension, enables to retrieve the client's IP address from the server side. Not supported in XML, HTTP communication, and JIS Connects.	**String**
GetColIndex	Returns the currently selected item's column index (0 based). Returns -1 when the selected item is a row.	**Integer**
GetCount	Returns the number of items in the listbox/combobox.	**Integer**
GetCurrentContextParm	Returns the current context parameter for the GeneralUTMethod. Returns _FAIL in case of failure.	**String**
GetCurrentHostScreenName	Returns the current host screen's name.	**String**
GetCurrentScreenName	In a JcsSession, returns the name of the current screen of the session.	**String**
GetCurrentText	Returns a string built from the item currently selected in the listbox/combobox, followed by the list of all items in the listbox/combobox separated by semi-columns, e.g. when Spring is the selection, the method returns: `Spring;Winter;Spring;Summer;Fall;.`	**String**

Table 27. DoMethods (Sheet 11 of 34)

DoMethod name	Description	Return type
GetDataFromScreen	Returns data taken directly from the host screen.	**String**
GetDocumentElement	Returns the root element of an XML document. Use in DOM-based XML methods.	**XML Element**
GetElementsByTag Name	Returns a list of the elements that have the tag name specified in the parameter. Use in DOM-based XML methods.	**XML Element List**
GetExternalInputStrea m	Returns an InputStream object from an external data source.	**JSInput Stream**
GetExternalOutput Stream	Returns an output stream object in order to send data from the server to the client.	**JSOutput Stream**
GetHandle	Returns the current DBSession's handle. You can use this handle to retrieve a DBSession instance representing this session by calling *ConnectToDBSession*.	**String**
GetHandle	Returns the current JcsSession's handle. You can use this handle to retrieve a JcsSession instance representing this session by calling *ConnectToJcsSession*.	**String**
GetHostCursorCol	Returns the 0-based value of the column in which the host cursor is located. Returns -1 in case of failure.	**Integer**

Table 27. DoMethods (Sheet 12 of 34)

DoMethod name	Description	Return type
GetHostCursorRow	Returns the 0-based value of the row in which the host cursor is located. Returns -1 in case of failure.	**Integer**
GetHostFieldColByName	Gets the column location of a given field on the host. If the field is in a list, the location of the list column currently in focus is returned. When the DoMethod fails to retrieve the location, it returns -1.	**Integer**
GetHostFieldHelpContext	Returns the help context of the given host field.	**String**
GetHostFieldRowByName	Gets the row location of a given field on the host. If the field is in a list, the location of the list column currently in focus is returned. When the DoMethod fails to retrieve the location, it returns -1.	**Integer**
GetIntByColumnIndex	Gets the value of a column (specified by its index) in the current row as a Java int. Returns the column value; if the value is SQL NULL, returns 0.	**Int**
GetIntByColumnName	Gets the value of a column (specified by its name) in the current row as a Java int. Returns the column value; if the value is SQL NULL, returns 0.	**Int**
GetLastError	Returns a string that represents the exception that has been thrown by the last ExternalData, JcsSession or DBSession method.	**String**

Table 27. DoMethods (Sheet 13 of 34)

DoMethod name	Description	Return type
GetLength	Returns the number of elements in a node list. Use in DOM-based XML methods.	**Int**
GetLinkText	Returns the text of the link.	**String**
GetMethodParameter Value	Returns a string containing the value of the specified parameter.	**String**
GetMsgFieldHostRow	Returns the line (0-based) of the DIL message field on the host. Returns -1 in case of failure.	**Integer**
GetNextWord	Returns the word (delimited by blanks) that follows the word in which the position defined by the parameter is.	**String**
GetNodeAttribute	Returns the value of a node's attribute. The attribute is specified in a parameter. Use in a template-based XML method.	**String**
GetNodeValue	Returns the text of a node. Use in template-based XML methods.	**String**
GetNodeTextValue	Returns the text of a node. Use in DOM-based XML methods.	**String**
GetNumOfNonProtecte d BytesOnHost	Returns the number of non-protected bytes from the host location specified in the parameters.	**Integer**

Table 27. DoMethods (Sheet 14 of 34)

DoMethod name	Description	Return type
GetPasswordFromUser	Opens a dialog box with a textbox for user input, which is displayed as a password. When the user presses OK, the method returns the string typed by the user. When the user presses Cancel, the method returns _FAIL.	**String**
GetRMBPressedControl	Returns the last control on which the RMB was pressed.	**GUI Object**
GetRowCount	Returns the number of rows in the table.	**Integer**
GetRowIndex	Returns the row of the currently selected item (0 based index). Returns -1 when the selected item is a column.	**Integer**
GetSelCount	Returns the quantity of selected items in a multiple-selection listbox/table.	**Integer**
GetSelIdx	Returns the index of the currently selected item.	**Integer**
GetSelList	Returns an IntVec with numbers of items selected in a multiple listbox.	**IntVec**
GetSelString	Returns the string selected from a single-selection listbox/combobox.	**String**
GetSessionID	Returns the SessionID as a string which can be up to 10 digits long, and can contain a leading minus sign. (Internally, the ServerID is stored as a Java integer, in 32-bit two's complement format.)	**String**

Table 27. DoMethods (Sheet 15 of 34)

DoMethod name	Description	Return type
GetSessionType	Returns the session type as defined in the ServerConfiuration.xml file.	**String**
GetSharedUserVariable	Returns the value of a variable from the variables pool shared between client and server. When the variable does not exist, returns an empty string.	**String**
GetString	Returns the combobox/listbox string that occupies the index specified in parameter.	**String**
GetStringByColumnIndex	Gets the value of a column (specified by its index) in the current row as a Java string. Returns the column value; if the value is SQL NULL, returns null.	**String**
GetStringByColumnName	Gets the value of a column (specified by its name) in the current row as a Java string. Returns the column value; if the value is SQL NULL, returns null	**String**
GetTableCell	Returns the value of a table cell specified by table name, row number, and column name. Returns null if cell is not found.	**String**
GetTagName	Returns the name of an element's tag. Use in DOM-based XML methods.	**String**
GetText	Returns the text of the selected item in the combobox.	**String**

Table 27. DoMethods (Sheet 16 of 34)

DoMethod name	Description	Return type
GetTextFromUser	Opens a dialog box containing *OK* and *CANCEL* buttons, for user text input. Returns the string typed by the user, or _FAIL in case the user clicks *CANCEL*.	**String**
GetTextNum	Returns the number (0 based) of the selected radio button.	**Integer**
GetTextOfMessage	Returns the text of a host message.	**String**
GetTimeOutLimit	Returns the maximum amount of time (in milliseconds) the runtime waits for the host. This corresponds to the TimeOutLimit setting in the *<ApplName>.ini* file.	**Integer**
GetTopIndex	Returns the top visible line/row index in the listbox/table (retrieval all only).	**Integer**
GetTransitionListSize	Many2Many - gets the transition list size.	**Integer**
GetTransitionType	Many2Many - gets the transition type.	**String**
GetUserVariable	Returns a variable's value from the variables pool. When the variable does not exist, returns an empty string.	**String**
GetVarValueByName	Returns the value of a variable that belongs to a Subapplication. Returns null if the variable does not exist. Returns an empty string if the variable was not updated.	**String**

Table 27. DoMethods (Sheet 17 of 34)

DoMethod name	Description	Return type
GetVar	In a JcsSession, returns the value of a variable specified in the parameter. Returns null if the variable is not found.	**String**
GetXMLElementByName	Returns a DOM-based element, from a node in a template-based XML document. The method is used to bridge between the JIS template interface and the DOM interface.	**XML Element**
GetXmlLastError	Returns a description of the last error to occur during the execution of the method. Use in DOM-based or template-based XML methods.	**String**
HasFocus	Returns whether the receiver has focus or not.	**Boolean**
HideControl	Hides the receiver.	**Boolean**
HideWindow	Hides the receiver, which can be either a window or a subwindow. This DoMethod has no return value.	**Void**
HostMessageHelpRequested	Performs a help message as was pressed on the question mark.	**Void**
IgnoreHostChangesForSynchronize	Notifies the JIS runtime not to synchronize upon the last change on the host (useful for cases of spontaneous changes on the host).	**Void**
InLogon	Returns whether the current screen is a logon screen or not (TRUE/FALSE).	**Boolean**

Table 27. DoMethods (Sheet 18 of 34)

DoMethod name	Description	Return type
InsertString	Adds a string to the listbox/ combobox at the specified index.	**Void**
InTheScr	Checks whether the host screen has changed or not after the execution of the last method line. Returns _TRUE when the screen is the same and _FALSE when it has changed.	**Boolean**
InTheScrWithoutMsgs	Checks whether the host screen has changed or not after the execution of the last Method line, without analyzing messages. Returns _TRUE when the screen is the same and _FALSE when it has changed.	**Boolean**
IsCellProtected	Checks whether the cell is protected or not. Returns _TRUE when the cell is protected and _FALSE when it is not.	**Boolean**
IsDefaultPushButton	Returns whether the button is the default button or not.	**Boolean**
IsDependent	Returns whether the Subapplication is a dependent Subapplication (Many to Many) or not.	**Boolean**
IsEnabled	Returns whether the receiver is enabled.	**Boolean**
IsFolded	Returns whether the table is currently in fold status.	**Boolean**
IsFoldedOnHost	Returns whether the list is currently in fold status on the host.	**Boolean**

Table 27. DoMethods (Sheet 19 of 34)

DoMethod name	Description	Return type
IsHostFieldHidden	Checks whether the host field is hidden. Returns _TRUE when the field is protected and _FALSE when it is not.	**Boolean**
IsHostFieldProtected	Checks whether a host field is protected on the host. Returns _TRUE when the field is protected and _FALSE when it is not.	**Boolean**
IsHostKeyboardLocked	Checks whether the host keyboard is locked and is not in the X-SYSTEM state. Returns _TRUE when these conditions are met and _FALSE when they are not.	**Boolean**
IsHostPopup	Checks whether the current screen is a host popup screen.	**Boolean**
IsHostScreenWindowOpen	Returns whether the host screen window is open or not.	**Boolean**
IsJavaGUI	Checks whether the current runtime is a Java runtime. Returns _TRUE when it is and _FALSE when it is not.	**Boolean**
IsJavaServer	Returns whether this is a Java runtime.	**Boolean**
IsModified	Returns whether the edit field was modified.	**Boolean**
IsPopup	Returns whether the Subapplication is a window popup Subapplication.	**Boolean**

Table 27. DoMethods (Sheet 20 of 34)

DoMethod name	Description	Return type
IsPrincipal	Returns whether the Subapplication is a principal Subapplication (Many to Many).	**Boolean**
IsScreenInTransitionList	Many2Many - returns whether the given screen is in the transition list.	**Boolean**
IsSessionInUse	Returns whether the session is connected to a client, or not, i.e. is a pooled session.	**Boolean**
IsVBGUI	Checks whether the current runtime is a VB runtime. Returns _TRUE when it is and _FALSE when it is not.	**Boolean**
IsVisible	Returns whether receiver is visible.	**Boolean**
Item	Returns an XML element contained in a list of nodes. The element is specified by its index in the list. Use in DOM-based XML Methods.	**XML Element**
LastMessageClear	Clears the last host message received by the Application.	**String**
LastMessageGet	Returns the last host message received by the Application.	**String**
LeftJustify	Returns a formatted copy of the receiver in which leading blanks have been removed (i.e. left justification).	**String**
Length	Returns the receiver's length.	**Integer**
LibraryName	Returns the library's name.	**String**

Table 27. DoMethods (Sheet 21 of 34)

DoMethod name	Description	Return type
LibraryNameBySubappl Name	Gets the library name of a given Subapplication, or "" if unknown.	**String**
MenuCheckByPath	Puts a check mark on a menu item. This DoMethod has no return value.	**Void**
MenuDeleteByPath	Deletes a menu item. This DoMethod has no return value.	**Void**
MenuDisableByPath	Disables a menu item or a popup. This DoMethod has no return value.	**Void**
MenuEnableByPath	Enables a menu item or a popup. This DoMethod has no return value.	**Void**
MenuToggleCheckBy Path	Toggles a menu item check on or off. This DoMethod has no return value.	**Void**
MenuUnCheckByPath	Removes a check mark from a menu item. This DoMethod has no return value.	**Void**
MoveAccordingToHost	Displays the GUI window according to the host application's behavior. When the host screen changes, moves to the corresponding Subapplication. When the host screen does not change, stays on same Subapplication.	**Void**
MoveCursorToFieldBy Name	Moves the cursor to the specified host field. This DoMethod has no return value.	**Void**

Table 27. DoMethods (Sheet 22 of 34)

DoMethod name	Description	Return type
MoveCursorToRowCol	Moves the cursor to a certain location on the host, specified by a line and a column number. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
MoveCursorToTableField ByColNameAndRow Number	Moves the cursor to the field with the given name.	**Void**
MoveToDependent Screen	Many2Many - moves to a certain dependent screen on host.	**Boolean**
MoveToNextFullScreen	Moves to the current full screen on host.	**Void, a SubApp**
Name	Returns the Application's name.	**String**
NewDocumentFrom Template	Builds a new XML document based on the template specified in the parameter. Use in template-based XML methods.	**XML Template Document**
NewXMLTreeDocument	Creates a new empty XML document. Use in DOM-based XML methods.	**XML Tree Document**
Next	Moves the cursor down one row from its current position. Returns _TRUE if the new current row is valid; _FALSE if there are no more rows.	**Boolean**
NextNode	Gets to the next node that has the same JISName, as specified in the parameter. Use in template-based XML methods.	**Boolean**

Table 27. DoMethods (Sheet 23 of 34)

DoMethod name	Description	Return type
NotifyOnMessages TreatmentRestart	For multi-scrollable messages screens only. Informs the runtime on AID changes on the host, using a message handling method.	**Boolean**
NumberOfMsgsOn Screen	Returns the number of messages on the host screen.	**Integer**
NumberOfRadioButto ns	Returns the number of radio buttons in a group.	**Integer**
OpenPrinterEmulation	Opens a printer emulation session. This DoMethod has no return value.	**Void**
OpenPrinterEmulation ByHandler	Opens a printer emulation session, using printer extension classes from the specified handler name. This DoMethod has no return value.	**Void**
ParseDocumentBy Template	Parses a document read from the input stream, according to the specified template, and inserts JISNames into the nodes. Use in template-based XML Methods.	**XML Template Document**
PositionCursorOnLast MsgField	Positions the cursor on the DIL message field with a message on the host.	**Boolean**
PrinterEmulationLU Name	Returns the LUName of the latest printer emulation session. If there was no printer emulation session in the current Application run, the method returns _FAIL.	**Void**
PrintHostScreenToLog	Prints the current contents of the host screen to the session's log.	**Void**

Table 27. DoMethods (Sheet 24 of 34)

DoMethod name	Description	Return type
PrintString	Prints a string into the OutputStream.	**Void**
PutTableCell	Sets the value of a table cell specified by table name, row number, and column name. Returns the previous value of the cell, or null in case of failure.	**String**
PutVar	In a JcsSession, sets a variable to the specified value. Returns the previous value of the screen field, or null if the screen field is not found.	**String**
ReadToString	Transforms an InputStream into a string. This function empties the input stream.	**String**
ReadXMLTreeDocument	Builds a DOM-based XML document from the input stream. Use in DOM-based XML methods.	**XML Tree Document**
RefreshCurrentSubApplication	Refreshes the current Subapplication window from the screen. This DoMethod has no return value.	**Void**
RefreshFormatHandler	Reloads the format information from the application.ini file.	**Void**
RefreshFromHost	Refreshes the host screen.	**Boolean**
RefreshFromHostWithoutMsgs	Refreshes the host screen without checking for messages.	**Boolean**
RefreshSubApplication	Refreshes the Subapplication window from the screen.	**Void**

Table 27. DoMethods (Sheet 25 of 34)

DoMethod name	Description	Return type
RefreshSubApplication WithAllDependents	Refreshes the Subapplication window from the screen, and from all dependent screens when relevant. This DoMethod has no return value.	**Void**
ReloadApplIniFile	Reloads the *<ApplName>.ini* file into memory.	**Boolean**
RemoveNode	Removes a node and all its child nodes from a document. Use in template-based XML methods.	**Boolean**
Replace	Replaces part of the receiver with part of a given string.	**String**
ReplaceString	Replaces a string in the listbox at the specified index.	**Void**
Reset25thLineError	Clears a 25th line error. For AS/400 only.	**Void**
ResetTableContent	Clears the content of a table.	**Void**
RightJustify	Returns a copy of the receiver right justified.	**String**
Rows	Returns the number of rows on the host screen.	**Integer**
RTLanguage	Returns the host application language.	**String**
SaveHostScreen	Saves the current host screen to a file. You must provide the file name with a .PNL extension.	**Void**

Table 27. DoMethods (Sheet 26 of 34)

DoMethod name	Description	Return type
SelectString	Selects a specified string if possible, and returns its index.	**Integer**
SendAIDKeyByString	Presses an AID key specified by its name on the host. This DoMethod has no return value.	**Void**
SendASpecificKey	Presses a key specified by its ID number. This DoMethod has no return value.	**Void**
SendKeyByString	Sends the host a non-AID key specified by its name on the host. This DoMethod has no return value.	**Void**
SendKeysWithoutRese t	Sends a special key to the host without prior reset.	**Boolean**
Set	Fills a string with a certain character.	**Void**
SetAttribute	Adds an attribute to a node and sets its value. If the attribute already exists, its value is changed to the new value. Use in DOM-based XML Methods.	**Boolean**
SetAutoReconnectTo HostMode	Sets the value of the autoReconnectToHost flag for the current running session.	**Void**
SetAutoSynchronize Mode	Determines whether to automatically synchronize on each new screen. Returns the previous value.	**Boolean**

Table 27. DoMethods (Sheet 27 of 34)

DoMethod name	Description	Return type
SetCellText	Sets the text of a table cell, according to the row and column variable name. Updates the host only if needed. You can also indicate whether to run the formatting functions from the Window to the Manager. Returns _TRUE in case of success, _FALSE in case of failure. For example, if the cell is out of the table boundaries.	**Boolean**
SetCheck	Checks/unchecks the checkbox.	**Integer**
SetCurrentContextParm	Sets the current context parameter for the next GeneralUTMethod. This DoMethod has no return value.	**Void**
SetCurrentMenuFromFloating	Loads a specified floating menu and sets it to be the current menu. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
SetCurrentMenuFromWindow	Retrieves a menu from the window according to the given path, and sets it to be the current menu.	**Void**
SetCurSel	Sets the current selection to index (single selection listbox).	**Void**
SetCursorPosOnScreen AccordingToFocusedControl	Sets the cursor position in the screen on the area that is equivalent to the currently focused control. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**

Table 27. DoMethods (Sheet 28 of 34)

DoMethod name	Description	Return type
SetCursorPosOnScreen ByGivenControl	Sets the screen cursor to the position that corresponds to the control specified in the parameter. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
SetCurTableSel	Sets the current selection to row and column.	**Integer**
SetExpectedScreenByName	Sets the expected screen. When more than one Subapplication can match the current screen, this method indicates the name of the one that should be called up. This DoMethod has no return value.	**Void**
SetExpectedScreenForeverByName	Sets the expected screen until the session terminates. *SetExpectedScreenByName* overrides this method. This DoMethod has no return value.	**Void**
SetFocus	Enables and sets the focus to the receiver.	**Void**
SetFocusfromScrToWin	Sets the focus on the GUI controls according to the position of the cursor on the screen. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
SetFocusOnRMBPressedControl	Moves the focus to the last control on which the RMB was pressed.	**Void**
SetFocusOnWindow	Sets the focus to the receiver (Subapplication) and sets the focus on the control according to the host.	**Void**

Table 27. DoMethods (Sheet 29 of 34)

DoMethod name	Description	Return type
SetLabel	Sets the label text of a checkbox component.	**Void**
SetLimitText	Sets the limit on the number of characters permitted in the prompt.	**Void**
SetModifiedFlag	Sets the control's MODIFIED flag to ON or OFF.	**Boolean**
SetMultiPageMultiplier	Sets the value used to multiply the number of pages displayed in the table, to increase the number of list pages read from the host. Returns the previous value.	**Integer**
SetMultSel	Sets one selected item in a multiple listbox.	**Void**
SetNodeAttribute	Sets the value of a node's attribute. If the attribute already has a value, it is replaced by the new value. Use in a template-based XML method.	**Boolean**
SetNodeValue	Inserts the text specified in the parameter, into a node identified by its JISName. Use in template-based XML methods.	**Boolean**
SetNonResponseMode	Determines whether the host runs in Non-Response mode or runs normally, depending on the parameter value (true or false). This DoMethod returns the previous value of the assumption.	**Boolean**
SetSubApplFocus	Sets the focus to the receiver (Subapplication).	**Void**

Table 27. DoMethods (Sheet 30 of 34)

DoMethod name	Description	Return type
SetText	Sets the text of the control's edit field.	**Object**
SetTextNum	Sets the selected radio button by its number (0 based).	**Void**
SetTimeOutLimit	Sets the maximum amount of time (milliseconds) to wait for the host.	**Void**
SetTopIndex	Sets the top visible line index/row index in the listbox/table (Retrieval All only).	**Integer**
SetTransitionType	Many2Many - sets the transition type (for private use).	**Void**
SetURL	Sets a receiver's URL.	**Void**
SetVarValueByName	Sets a Subapplication's variable value.	**Boolean**
SetWaitForScreenState	Sets the amount of time the runtime should wait until the screen(s) specified in the parameters list arrive from the host, before it sends whatever screen is currently present.	**Void**
ShowHostScreen Window	Shows the host screen window.	**Void**
ShowNormal	Sets the receiver to its normal (i.e. unhidden) state. This DoMethod has no return value.	**Void**
Sleep	Suspends execution of the application for the specified number of milliseconds.	**Void**

Table 27. DoMethods (Sheet 31 of 34)

DoMethod name	Description	Return type
StrDelete	Removes a sub-string from the receiver.	**String**
StringToInputStream	Returns an input stream object built from a string specified in the parameter.	**JSInput Stream**
SubStr	Returns a sub-string from the receiver.	**String**
SuffixFrom	Gets a suffix of the receiver from a certain position.	**String**
SwitchDateFormats	Changes the formats.INI file settings in the [DateFormat] section that are used to format date fields. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
SynchronizeByHost Screen	Synchronizes the host screen and the window. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
Terminate	Terminates the receiver (Subapplication).	**Void**
TerminateBack	Terminates the receiver (Subapplication) and refreshes topmost Subapplication.	**Void**
TerminateBackUpTo	Terminates Subapplications up to the receiver; refreshes topmost Subapplication.	**Void**
TerminateIfScreen Changed	Terminates the receiver (Subapplication) if screen has changed.	**Void**

Table 27. DoMethods (Sheet 32 of 34)

DoMethod name	Description	Return type
TerminateUpTo	Terminates Subapplications up to the receiver (Subapplication).	**Void**
ToggleFoldMode	Toggles between folded and unfolded mode.	**Integer**
ToLower	Converts the receiver to lower case.	**String**
TotalExit	Causes the Application to exit. Equivalent to selecting *File > Exit* in the menu bar. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
ToUpper	Converts the receiver to upper case.	**String**
ToXMLTreeDocument	Converts a template-based XML document to a DOM-based XML document.	**XML Tree Document**
TrackLeftMBCurrent Menu	Displays the current floating menu. Use only on left mouse triggers.	**Void**
TrackRightMBCurrent Menu	Displays the current floating menu. Use only on right mouse triggers. This DoMethod has no return value.	**Void**
TrtString	Replaces all characters in "fromStr" with those in "toStr" of the same index.	**String**

Table 27. DoMethods (Sheet 33 of 34)

DoMethod name	Description	Return type
TypeStrWithoutCheck	Types a string in the current host cursor location and ignores any consequences on the host. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
TypeStrWithoutCheck AndWait	Types a string in the current host cursor location and waits for screen response. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
UpdateAFormatHandl er	Reloads the format values specified in the *.ini file.	**Boolean**
ValidateDependent ScreenByName	Many2Many - validates that the current screen is a dependent screen and moves there if it is not.	**Boolean**
VariableParentNameB y Name	Returns the parent variable name or _FAIL.	**String**
VecGet	Gets one character from the receiver.	**Character**
VecPut	Assigns one character in the receiver.	**String**
ViewHostScreenWindo w	Equivalent to selecting *View* > *HostScreen* in an Application runtime main menu (i.e. toggles the host screen view). This DoMethod has no return value.	**Void**

Table 27. DoMethods (Sheet 34 of 34)

DoMethod name	Description	Return type
WriteApplDirPrivate ProfileString	Writes a string to the INI file specified in the parameters. The file MUST reside in the Application's directory. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
WriteApplPrivateProfi le String	Writes a string to the <applname>.INI file. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
WriteMsgToDIL	Writes a message to the DIL. There is no return value to indicate failure if the DIL does not exist.	**Void**
WriteSharedUser Variable	Writes a variable to the variables pool shared between client and server. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
WriteUserVariable	Creates a variable in the variables pool and writes a value in it. Returns _FALSE in case of failure, _TRUE in case of success.	**Boolean**
WriteXMLTemplate Document	Writes a template-based XML document to the output stream. Use in template-based XML methods.	**Boolean**
WriteXMLTreeDocum ent	Writes a DOM-based XML document to the output stream. Use in DOM-based XML methods.	**Boolean**

Chapter 16. Runtime Screen Identification View

Runtime Field Information View and Runtime Identification View, are used to specify the information that ACE needs to identify and process each of the screens during runtime. In Runtime Screen Identification View, the Application developer provides the information that helps the runtime identify each of the various screens.

This chapter describes:

- How Runtime Screen Identification Works
- The Runtime Screen Identification Screen
- Fixed and Variable Characters in a Screen
- Message Areas in a Screen
- The Screen Fingerprint
- Undoing Your Changes
- Automatic Identification Definitions
- Comparing a Captured Screen Image and a Screen in Runtime Screen Identification View

How Runtime Screen Identification Works

When you enter Runtime Screen Identification View the screen appears as follows:

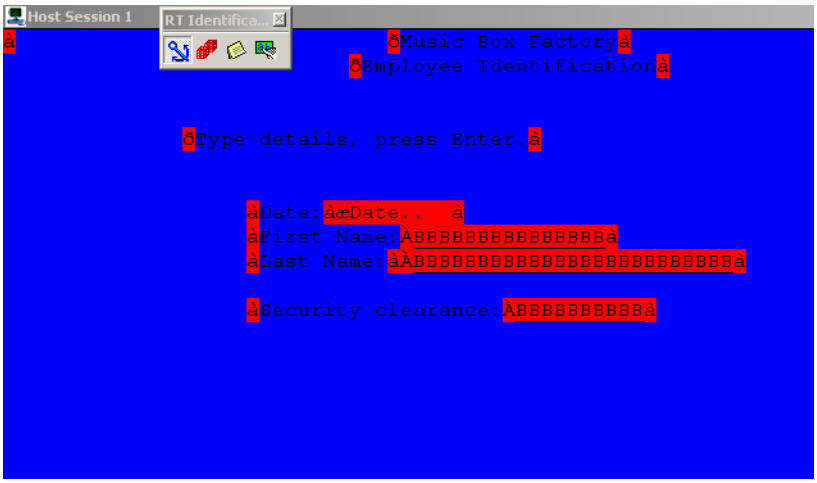


Figure 184. Runtime Screen Identification view

Runtime Screen Identification View prepares your screen for operation in the runtime Application. In runtime, whenever ACE encounters a new screen, it must identify the correct window to display. It does this by performing the following two processes:

- Elimination.
- Verification.The color markings in Runtime Screen Identification View create a transparency for every host application screen which is color coded. During runtime, specific sections of the screen that are being identified are compared with the transparencies. Comparing only specific sections of the screen, and not the whole screen, saves a considerable amount of time and is sufficient to eliminate most of the transparencies.

At the conclusion of the elimination process, ACE verifies that the selected screen is the correct screen to display, based on the screen fingerprint.

The *Fingerprint* of a screen completes the identification by verifying which screen, from the group of screens that were not eliminated, matches the current screen exactly. Because the fingerprint of a screen is unique to a screen, it serves to differentiate between the screens that were not eliminated.

Runtime Screen Identification View presents defaults for the location and dimension of a screen's variable patterns. Since the contents of a screen may vary during runtime, it is important to indicate any pattern that may change as 'variable', and to specify a correct range for the variations in location or dimension. Runtime Screen Identification View provides the Application developer with tools to improve the defaults if necessary.

The Runtime Screen Identification Menu

The following menu is enabled in Runtime Screen Identification View.

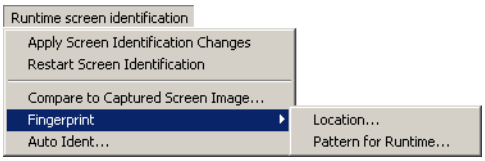


Figure 185. Runtime Screen Identification view menu options

Apply Screen Identification Changes	Apply any changes made to the screen.
Restart Screen Identification	Restart the identification ignoring all changes made to the screen.

Compare to Captured Screen Image	Compare a captured screen image to the currently displayed Subapplication. Areas where inconsistencies are found are colored pink.
Fingerprint	*Location:* Specify the exact location of the Fingerprint. *Pattern for Runtime:* Select a Pattern Definition to be used as a Fingerprint.
Auto Ident...	Set an identification type to a specific Identification Definition.

The Runtime Screen Identification Screen

This section describes the runtime screen identification screen.

The Color-coding Scheme

Identifying a screen is a combination of two processes: *elimination* and *verification*. The color-coding of the host screen in Runtime Screen Identification View is related to both these processes.

In Runtime Screen Identification View, every character in the screen is marked in one of five colors: blue, red, yellow, green or pink. The colors represent the following:

Blue	Characters that are fixed and do not change during runtime.
Red	Characters that are variable and may change during runtime.
Yellow	A line that may contain a message during runtime.
Green	A sequence of one or more characters that are unique to a screen and serve as a Fingerprint to identify a screen during runtime.
Pink	Areas in which inconsistencies have been found between a captured screen image from the runtime Application and a Subapplication.

The following schematic illustrates how a screen may look in Runtime Screen Identification View:

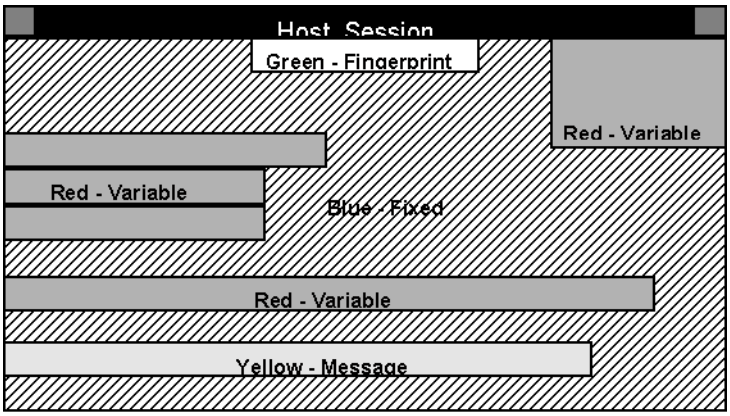


Figure 186. Schematic illustration of how a screen may look in Runtime ID view

In Runtime Screen Identification View, ACE displays the color-coded host screen. You can modify the color coding manually when necessary. You can also create Identification Definitions to automate the process.

Fixed and Variable Characters in a Screen

To identify a screen during runtime, ACE must know which characters of a screen are fixed and which characters may vary during runtime. The fixed characters can be used for comparison during the elimination process.

A screen is composed of character cells. A character cell is a section of the screen that can contain one unit of information (such as a character or an attribute). The number of character cells in a screen depends upon the number of its rows and columns. The number of character cells in a row is equal to the number of columns in the screen, and the number of character cells in a column is equal to the number of rows in the screen.

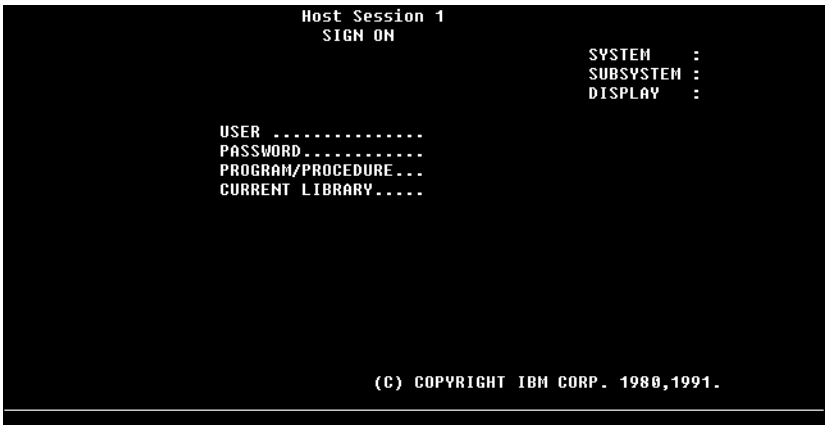
A fixed character is colored blue. The contents of a fixed character cell do not change during runtime.

A variable character is colored red. The contents of a variable character cell may change during runtime.

In Runtime Field Information View, only the Meaningful Patterns in the screen are classified as fixed or variable. Runtime Screen Identification View completes the fixed or variable markings *for the whole screen*, not only the Meaningful Patterns. It is important to mark any area in the screen that is variable, so that such an area does not prevent the identification of a screen during runtime.

Example 53. Fixed and variable characters

Let us take as an example the following SIGN ON screen:



In addition to Meaningful Patterns such a `USER`, `PASSWORD`, etc., this screen also contains the following pattern:

```
(C) COPYRIGHT IBM CORP. 1980, 1991.
```

This is not a Meaningful Pattern. However, such a pattern may change in content or in location. Therefore, it is important to mark such a pattern as variable.

If this pattern is marked as fixed, once it changes, the runtime will no longer be able to identify that this is the same SIGN ON screen.

However, when the pattern is marked as variable, the runtime will be able to continue identifying the screen correctly, even if the pattern changes.

The Application developer can make any necessary changes to the fixed and variable markings.

To mark fixed characters in a screen:

1. In the *RT Identification* floating toolbar, click .
2. Holding down the left mouse button, drag the cursor across the characters in the screen that you want to mark as fixed. A frame indicates the marked characters. When you release the mouse button, the framed area is colored blue.

Note: To cancel all your markings, select *Runtime Screen Identification* > *Restart Screen Identification*. You can also cancel an incorrect color marking by coloring over it. Each character in the screen can have only one color.

To mark variable characters in a screen:

1 In the *RT Identification* floating toolbar, click .
2 Holding down the left mouse button, drag the cursor across the characters in the screen that you want to mark as variable. A frame indicates the marked characters. When you release the mouse button, the framed area is colored red.

Note: To cancel all your markings, select *Runtime Screen Identification* > *Restart Screen Identification*. You can also cancel an incorrect color marking by coloring over it. Each character in the screen can have only one color.

It is important to mark the fixed and variable characters correctly, in order to help the runtime to identify a screen. Incorrect markings might obstruct the Runtime Screen Identification rather than help it. It is especially important that no variable characters be marked as fixed.

Message Areas in a Screen

A screen usually has specific lines in which messages appear. These are the messages that were defined using Message Definitions in Analysis View. The Application developer must mark any line in the screen in which a message can appear, because this information is important for the identification of a screen during runtime.

A message line is colored yellow. The maximum range of a message should be marked, but do not include screen attributes in a message line. Not more than one line can be marked at a time. In order to mark a message several lines high, each line should be marked separately.

To mark a message line in a screen

1 In the *RT Identification* floating toolbar, click .
2 Holding down the left mouse button, drag the cursor across the characters in the screen that you want to mark as a message line. A frame indicates the marked characters. When you release the mouse, the framed area is colored yellow.

   Make sure that *no screen attributes* are included in the message line.

Note: To cancel all your markings, select *Runtime Screen Identification* > *Restart Screen Identification*. You can also cancel an incorrect color marking by coloring over it. Each character in the screen can have only one color.

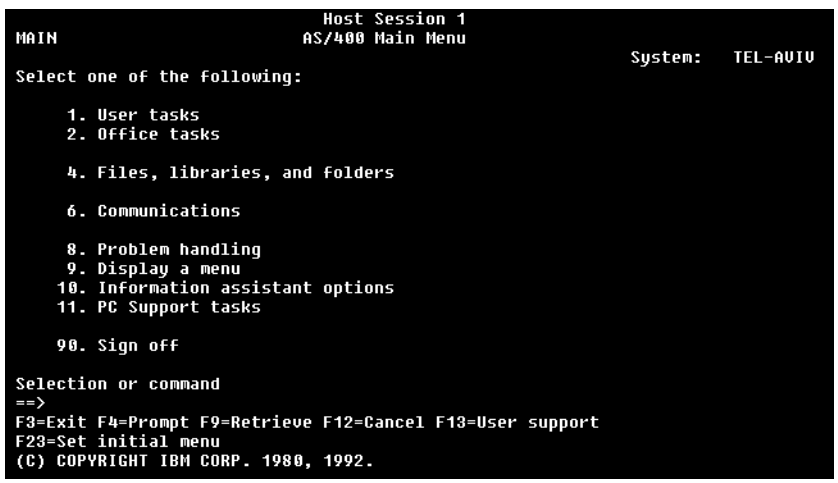
The Screen Fingerprint

In addition to fixed, variable and message characters, a screen can also have a *Fingerprint*.

The screen fingerprint is a sequence of one or more characters that are unique to a screen. The fingerprint is used during runtime to verify the identification of a screen.

Example 54. The screen fingerprint

The heading of a screen is usually unique to that screen and therefore can serve to verify the identification of the screen. Consider the following screen:



You could use the header of this screen as its fingerprint:

```
AS/400 Main Menu
```

The fingerprint of a screen is colored green. A screen can have only one fingerprint consisting of no more than one row.

To mark the fingerprint of a screen:

1. In the *RT Identification* floating toolbar, click .
2. Holding down the left mouse button, drag the cursor across the characters that are the fingerprint of the screen. A frame indicates the marked characters. When you release the mouse button, the framed area is colored green.

Note: To cancel all your markings, select *Runtime Screen Identification* > *Restart Screen Identification*. You can also cancel an incorrect color marking by coloring over it. Each character in the screen can have only one color.

The Location of the Fingerprint

The Location option provides flexibility in the location of the fingerprint. The location of the fingerprint string may sometimes vary in a screen. Use the *Location* option to define the range, in columns and in rows, in which a fingerprint can appear in a screen. When the range is specified, even if the location of the fingerprint varies, the runtime will be able to identify the screen correctly.

To define a location range for a fingerprint:

1 From the *Runtime Screen Identification* menu, select *Fingerprint* > *Location*.

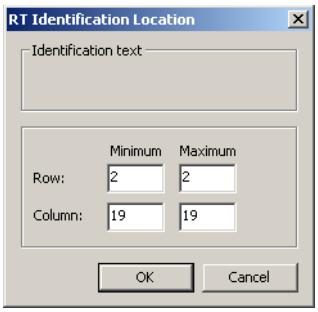


Figure 187. RT Identification dialog box

2 Type the *Minimum* and *Maximum* values in *Row* and in *Column* to define a location range for the fingerprint. (*Row* and *Column* numbering begin at zero.)

3 Click *OK*.

Note: When marking a fingerprint make sure that no characters are variable. If a fingerprint contains variable characters, the Runtime Screen Identification will not be able to correctly identify the screen. A fingerprint helps to identify a screen and is therefore unique. However, you can have the same fingerprint for more than one screen, although you should keep this number to a minimum.

Using a Pattern Definition as a Fingerprint

Another option is using a Pattern Definition to be recognized in runtime as a fingerprint. From the *Runtime screen identification* menu, select *Fingerprint > Pattern for Runtime*. Enter the Pattern Definition name in the *Fingerprint Pattern Definition for Runtime* dialog box.

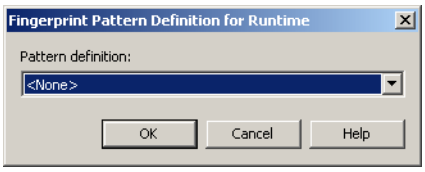


Figure 188. Fingerprint Pattern Definition for Runtime dialog box

The selected Pattern Definition must be matched on the screen in runtime for the Subapplication to be identified. It is not related to any specific location on the screen in ACE.

Undoing Your Changes

Whenever you change the Runtime Screen Identification View markings, ACE remembers what you did and allows you to undo the changes.

The undo feature differs from the Restart Screen Identification feature in that when you restart, you lose all modifications, even if the changes were made in a previous ACE session. The undo feature only acts on changes made in the current Subapplication session.

The Behavior of Undo Changes

The undo changes feature has the following behavior:

- ACE maintains a record of all the changes you make. Undo acts to reverse the most recent change. Once you have undone a change, a further undo reverses the next most recent change, and so on.
- Each change that is undone is added to a record of changes to redo. When you have undone several changes, redo performs the most recently undone operation.
- Each change that is redone is added to the record of changes to undo. When you have redone several changes, undo reverses the most recently redone operation.
- A new change clears the redo list, but does not affect the undo list.
- Once you close a given Subapplication, either by opening a different Subapplication, a different Application or library, or closing ACE completely,

that Subapplication's undo/redo memory is lost–when you next open that Subapplication there are no operations to undo/redo.

- Undoing some operations requires an Apply. An Apply is a general updating of the Subapplication, including the effects of any KnowledgeBase changes. If a particular undo operation requires an Apply, you are asked whether or not you wish to proceed with the undo.

Performing an Undo

You can undo operations in Runtime Screen Identification View in any one of three ways:

From the Toolbar

Click the toolbar *Undo* button. Before you undo, the button's ToolTip indicates which operation will be undone.

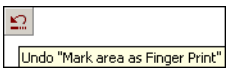


Figure 189. Undo from the toolbar

From the Edit Menu

From the *Edit* menu select *Undo*: The operation that will be undone appears beside the menu item.

Figure 190. Undo from the edit menu

From the Keyboard

Press *Ctrl+Z*.

Performing a Redo

You can redo operations in Runtime Screen Identification View, in any one of three ways:

From the Toolbar

Click the toolbar *Redo* button. Before you redo, the button's ToolTip indicates which operation will be redone.

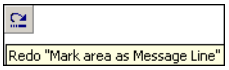


Figure 191. Redo from the toolbar

From the Edit Menu

From the *Edit* menu select *Redo*. The operation that will be redone appears beside the menu item.

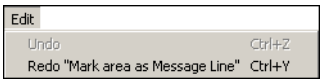


Figure 192. Redo from the Edit menu

From the Keyboard

Press *Ctrl+Y*.

Automatic Identification Definitions

You can automate the runtime screen identification by creating *Identification Definitions*. An Identification Definition is a Pattern Definition that is only used in Runtime Screen Identification View. You can specify the Identification Definitions for each type of area in the screen: Fixed, Variable, Message, Fingerprint.

To define an Identification Definition, from the *Runtime Screen Identification* menu select *Auto Ident*. The *Automatic Identification Definition* dialog box opens:

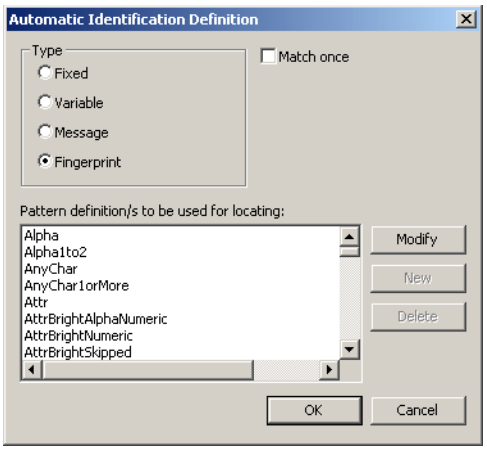


Figure 193. Automatic Identification Definition dialog box

Specify the Pattern Definitions to be used for locating each type of area in the screen.

Type	The type of area for which you are specifying the Identification Definition.
Match once	When set, the Identification Definition recognizes only the first character sequence that satisfies it.
Pattern Definitions To Be Used For Locating	This list contains all the Pattern Definitions. This is a multiple selection list box, in which you select the Pattern Definitions to be used for locating the Type specified above. To match a multiple selection, hold down *CTRL* while selecting the Pattern Definitions.
Modify, New	Use these buttons to modify an existing Pattern Definition, or create a new one.

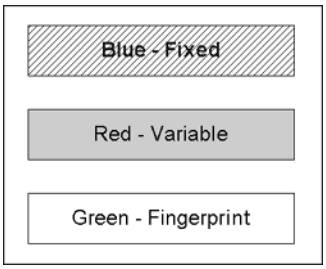
Example 55. Automatic identification definitions

The following is an example of how to modify the default Runtime Identification markings of a screen, and use the fingerprint Location option.

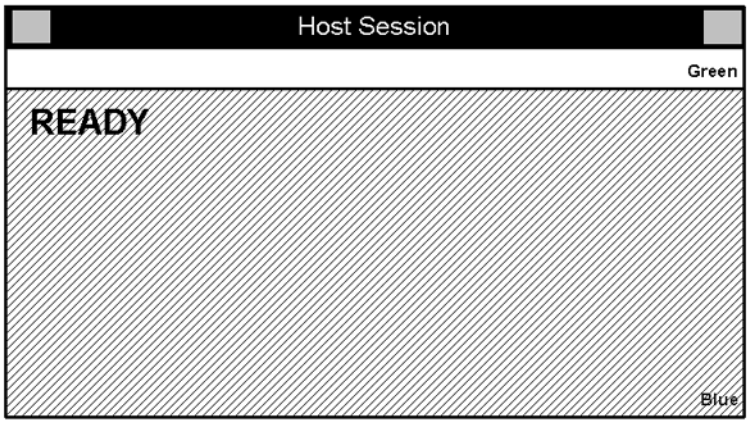
Let us take as an example a READY screen. The string "READY" is the fingerprint of this screen.

The "READY" string always begins at the first column (column zero) of a screen (including the attribute that appears to the left of the string), however, the row number can vary.

This example uses schematic drawings of a screen, to simplify the distinction between the different color markings. The color key is:



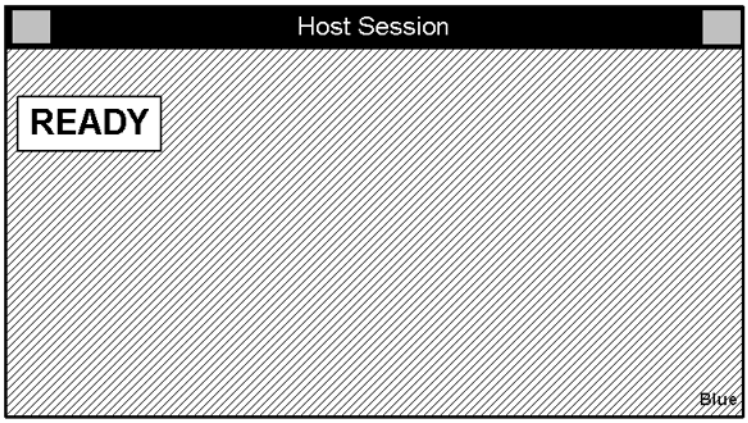
The following drawing presents the default identification markings of the READY screen:



Except for the first line, which is marked as the fingerprint, the whole screen is marked as fixed.

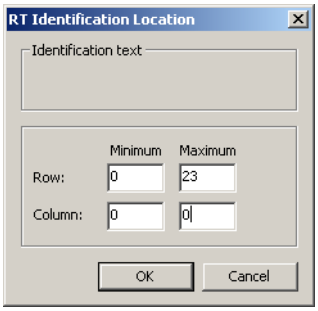
These default identification markings must be modified.

First, mark the READY string as the fingerprint using the *Fingerprint* icon:

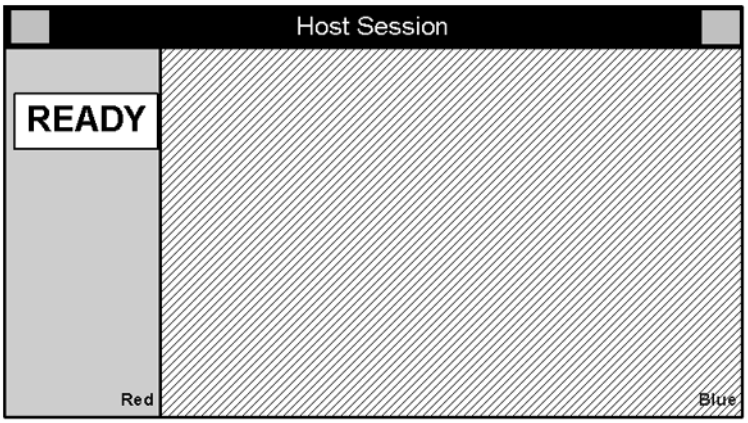


The location of the READY string may vary in its row number. The READY string always begins at column zero of a screen, but the row number may vary. Therefore, it is necessary to define a location range for the fingerprint.

To define a location range for a fingerprint, from the *Runtime Screen Identification* menu choose *Fingerprint* > *Location*. In the *RT Identification Location* dialog box, modify the value under *Row, Maximum* to 23, so that the dialog box looks like this:



Click the *OK* button. The screen now looks like this:



Because the location of the READY string may vary between row zero and row 23, this entire section of the screen is marked as variable.

Finally, since a READY screen contains information that is variable, it is necessary to mark the rest of the screen, which is marked as fixed, as variable too. To do this, use the *Variable* icon. The result is a completely red screen, except for the green Fingerprint, as shown below:

Comparing a Captured Screen Image and a Screen in Runtime Screen Identification View

In runtime, when there is an inconsistency between the host screen image and the screen image that you have converted, the host screen rather than the converted GUI window is displayed. This process is called bleedthrough.

ACE contains a feature that enables you to compare two screen images. Thus, if your runtime Application reverts to bleedthrough rather than displaying the converted window, you can capture the displayed host screen image, save it, and compare it to the PNL file that your Subapplication is based upon.

Note: The Compare to Captured Screen Image feature may be used solely with the File or DDS emulator. If it is necessary to edit the screen image, use the Screen Image Editor. See "Editing Screen Images" on page 43.

Capturing a Host Screen Image in Runtime

To capture the screen image in runtime, from the *Application* menu select *Emulator > Save Host Screen Image*.

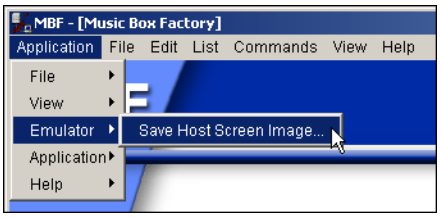


Figure 194. To capture a host screen image during runtime

Provide a name for the new panel in the *Save Panel As* dialog box.

Comparing Two Screen Images

Screen images can be compared in Runtime Screen Identification View. ACE uses the Runtime Screen Identification View color codes in runtime to identify and call up the appropriate screen. Bleedthrough occurs when there is a discrepancy between the runtime screen and the screen as identified in Runtime Screen Identification View.

To compare a host screen image and a Subapplication, use the *Compare to Captured Screen Image* option as follows:

1. In ACE, in Runtime Screen Identification View, call up the PNL file that your Subapplication is based upon.
2. From the *Runtime Screen Identification* menu, select *Compare to Captured Screen Image*. The *Select Next Screen* dialog box opens.

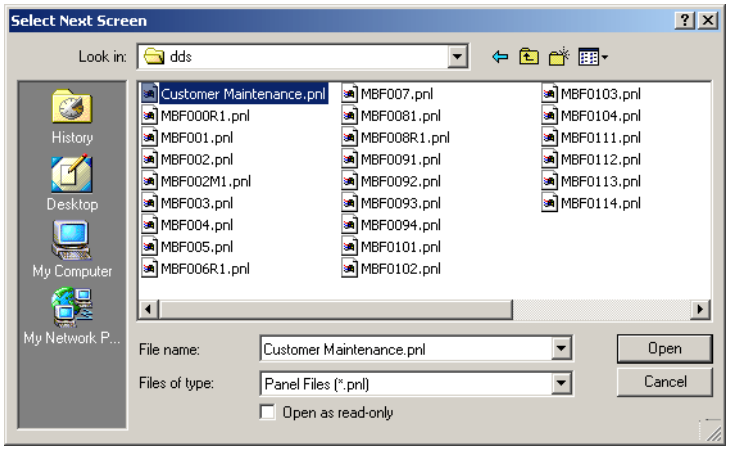


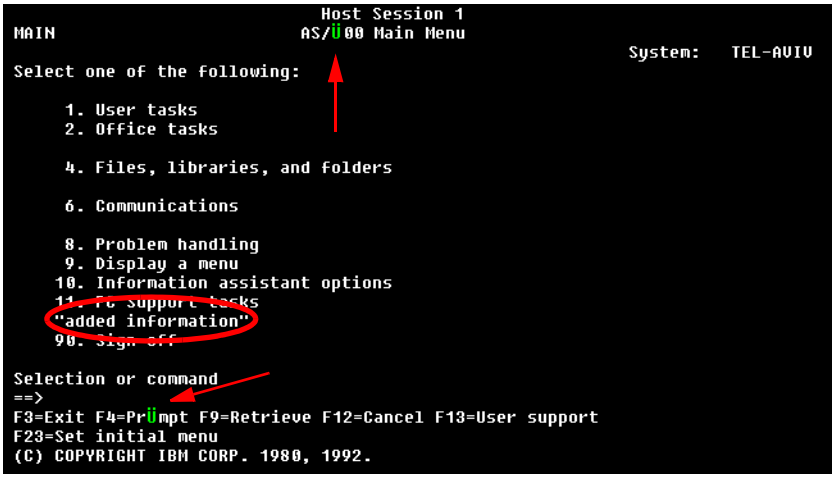
Figure 195. Select Next Screen dialog box

3 Select the panel that you saved from the screen presented in runtime.

The Compare to Screen Image option, in effect, superimposes the selected panel over the Runtime Screen Identification screen. If the information on the captured screen image and the Subapplication is identical, the user will not see any differences between how the Runtime Screen Identification screen is presented prior to the comparison. However, any inconsistencies which exist between the captured screen image and the Subapplication receive a pink representation on the Runtime Screen Identification screen.

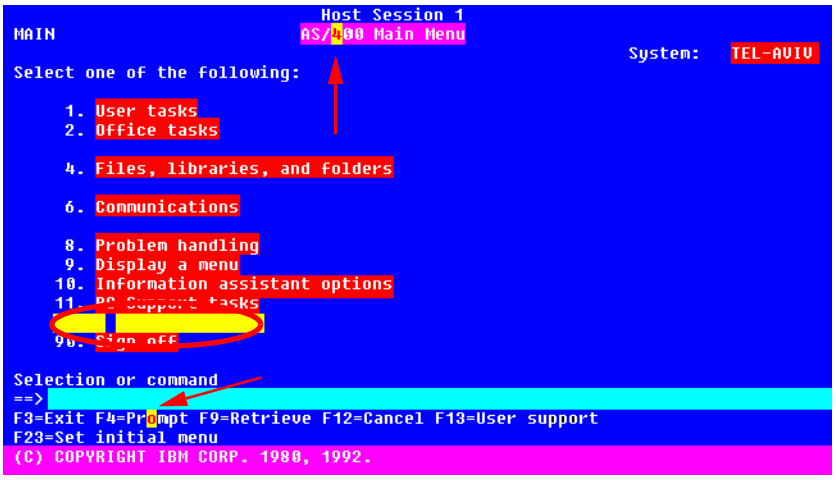
Example 56. Comparing two screen images

The following screen has undergone minor alterations. These alterations, however slight, will effect ACE's ability to recognize and call up this screen during runtime.



When bleedthrough occurs, capture the screen in runtime using the *Save Host Screen Image* option. Then, in ACE, call up the corresponding Subapplication, to compare it to the captured screen image.

In Runtime Screen Identification View, when the Subapplication is compared to the captured screen image, the inconsistencies are color coded pink.



ACE properly identifies the changed screen in runtime if the changed areas are marked as variable (red) on the new screen in Runtime Screen Identification View. To do this, select the *Variable* icon, and drag the cursor to outline a rectangle over the pink portions. The area becomes red, and ACE perceives data in that area as variable, thereby allowing for alternative data to appear in that section.

Chapter 17. Runtime Field Information View

Runtime Field Information View and Runtime Identification View, are used to specify the information that ACE needs to identify and process each of the screens during runtime. Runtime Field Information View defines the information for the meaningful patterns on the screen.

This chapter describes:

- How Runtime Field Information Works
- The Runtime Field Information Screen

How Runtime Field Information Works

When you enter Runtime Field Information View your window appears as follows:

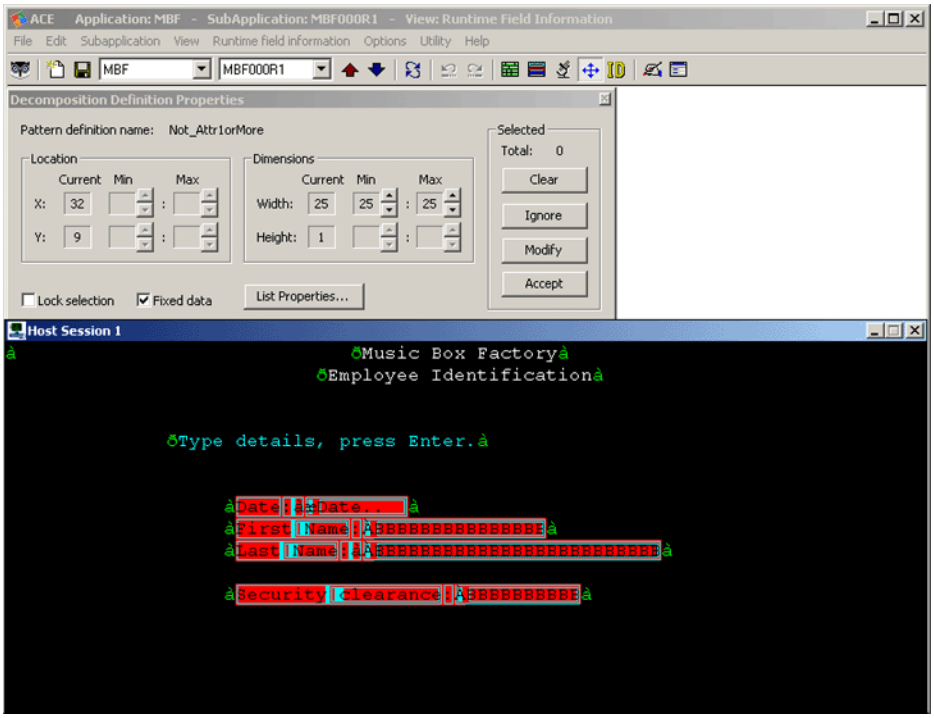


Figure 196. Runtime Field Information view

Runtime Field Information View prepares your screens for operation in the runtime Application. In runtime, whenever ACE encounters a new screen, it must identify the correct window to display.

It does this by performing the following two processes:

- Identification

  The runtime Application identifies a screen by determining which converted screen matches the current screen. This process is called Runtime Identification and is discussed in Chapter 16 - "Runtime Screen Identification View" on page 465.

- Decomposition

  After identifying the current screen, the screen is decomposed in order to locate the Meaningful Patterns that were recognized during the analysis. This process is called Runtime Field Information.

Variable Patterns

During analysis, ACE analyzes a single specific appearance of a screen. The contents of the analyzed screen are fixed and do not change. During runtime, however, the contents of a screen may change. As a result of changes in content, the location or dimensions of the patterns may vary.

Example 57. Variable patterns

To demonstrate how the contents of a screen may vary during runtime, let us examine the following pattern:

```
Line 1 of 5
```

Such a pattern can also appear in the following form:

```
Line 1 of 15
```

And also in the following form:

```
Line 11 of 15
```

The *dimension* of this pattern is variable and the range is between 11 and 13 characters. In addition, if we assume that the pattern is always right justified in a screen, then the *location* of the beginning of the pattern also changes according to its dimension:

```
Line 1 of 5
Line 1 of 15
Line 11 of 15
```

The above is an example of a single pattern that may vary during runtime in content and consequently may vary in dimension or location.

During runtime, ACE may retrieve information from the screen, or write information to the screen. Therefore, it is important to indicate which Meaningful Patterns may vary during runtime and specify the range of the variation.

The Runtime Field Information Screen

Runtime Field Information View presents default specifications for the location and dimensions of a screen's variable patterns. The defaults are based upon the analysis of a specific instance of the screen, and the Pattern Definitions that composed it. Since the contents of a screen may vary during runtime, it is important to indicate any pattern that may change as variable, and to specify a

correct range for the variations in location or dimension. This View provides the Application developer with tools to improve the default specifications, when necessary.

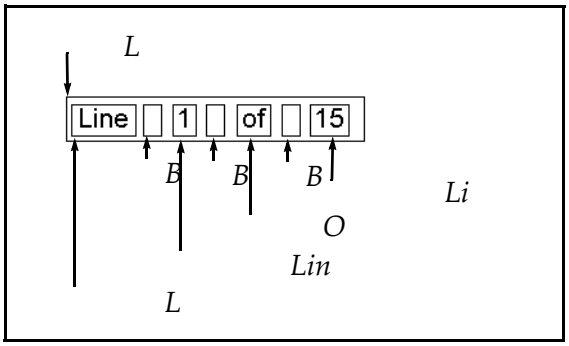
Runtime Field Information View opens by presenting ACE's suggested decomposition of a screen's Patterns. The Application developer can modify the decomposition if necessary, by specifying that a pattern is variable or fixed, or by modifying the location or dimension range. These specifications enable the runtime to refer to the appropriate patterns on the screen during runtime.

Example 58. The runtime field information screen

Let us refer again to the following pattern:

```
Line 1 of 15
```

The analysis recognizes such a pattern in the following manner:



As a result of the above analysis, the default break down is:

The *LineRangeFrom* Pattern Definition is fixed and its dimension range is Min:1 - Max:1

The *LineRangeOf* Pattern Definition is fixed and its dimension range is Min:2 - Max:2.

However, these default specifications need to be modified. Both Pattern Definitions should be marked as variable, not fixed, and in both cases the dimension range should be modified to Min:1 - Max:2.

The Color-Coding Scheme

In Runtime Field Information View, the screen patterns which were recognized during analysis are colored according to their runtime decomposition significance.

- Red and light blue indicate the *fixed* patterns.
- Dark blue indicates the *variable* patterns.

Example 59. The color-coding scheme

A fixed string pattern such as `Password` is colored in red and light blue, whereas the input field that follows it is colored in dark blue, because its contents are variable.

> **Note:** In Runtime Field Information View, only the Meaningful Patterns that may vary at runtime are colored, and the rest of the screen remains black

The options available in Runtime Field Information View are similar to those available in Analysis View. The Pattern Definitions are marked on the screen using the same conventions as in Analysis View. Rectangles are used to represent the hierarchical structure of the Pattern Definitions. The only difference is in the use of colors.

Selecting a Pattern Definition in the screen is done in the same way as in Analysis View, using the mouse.

- The screen decomposition can be displayed in different levels using the Display Level option.
- The *Decomposition Definition Properties* dialog box is very similar to the *Pattern Definition Properties* dialog box of Analysis View.

The Runtime Field Information Menu

The *Runtime Field Information* menu is accessed from the menu bar. It contains the following options:

Figure 197. Runtime Field Information menu options

Apply Field Information Changes	Apply any changes made to the screen.
Restart Field Information	Restart the decomposition, ignoring all changes made to the screen.

The Decomposition Definition Properties Dialog Box

The *Decomposition Definition Properties* dialog box contains information regarding the Pattern Definitions that are decomposed at this stage.

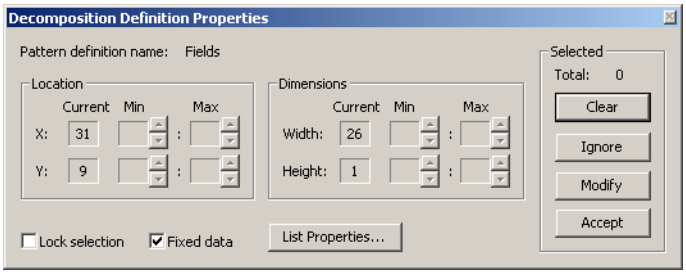


Figure 198. Decomposition Definition Properties dialog box

- To select a Pattern Definition, click it with the mouse. For every Pattern Definition selected, the *Selected Total* is incremented by one.
- Clicking the *Clear* button, clears the selected Pattern Definitions one by one, starting with the latest selection.
- The *Accept* button is used to accept changes in *Location* or *Dimensions*.
- The *Ignore* button is currently not used.

Fixed Data

Fixed Data is a checkbox that indicates whether the contents of a pattern should be fixed or variable during runtime. Use this checkbox to change the status of a field to/from fixed/variable.

- When set, the *Fixed Data* check box indicates that the contents of the selected Pattern Definition are fixed. For example, a string pattern such as "USERID" is fixed and does not change during runtime.
- When cleared, the *Fixed Data* check box indicates that the contents of the selected Pattern Definition are variable. For example, the contents of a USERID pattern may vary during runtime:

```
USERID ===> ABCDEFGH
USERID ===> MYUSERID
USERID ===> SOMEBODY
```

As you move the mouse across the screen, the *Fixed Data* check box changes according to the Pattern Definition the mouse is pointing at. The application developer can set or clear the check box for a selected Pattern Definition, when appropriate.

Lock Selection

Lock Selection enables you to lock onto the last Pattern Definition that was selected on the screen. Locking a selected Pattern Definition allows you to modify the location and dimensions of the selected Pattern Definition through the *Location* and *Dimensions* arrow keys.

When the *Lock Selection* check box is cleared, the *Location* and *Dimensions* arrow keys are disabled.

- When *Lock Selection* is set, the information displayed in the *Decomposition Definition Properties* dialog box refers to the last selected Pattern Definition.
- When *Lock Selection* is cleared, the information displayed in the *Decomposition Definition Properties* dialog box is influenced by the movement of the mouse on the screen. The information displayed refers to the Pattern Definition the mouse is currently pointing to.
- When the mouse points at an area in the screen that is not matched with any Pattern Definition, the *Decomposition Definition Properties* dialog box displays information that refers to the last Pattern Definition the mouse pointed to.

Location

Location contains the range of the locations a Pattern Definition may assume, in columns and in rows:

- The *X: Current* box contains the column number of the first character in the Pattern Definition. The *Min* and *Max* boxes contain the minimum and maximum column numbers at which the pattern can begin.
- The *Y: Current* box contains the row number of the Pattern Definition's current location. The *Min* and *Max* boxes contain the minimum and maximum row numbers at which the pattern can begin.

Note: Row and Column numbering begin at zero

Dimensions

Dimensions contains the range of the dimensions of a Pattern Definition in columns and in rows.

- The *Width: Current* box contains the current dimensions of the pattern in columns. The *Min* and *Max* boxes contain the minimum and maximum dimensions of the pattern in columns.
- The *Height: Current* box contains the current dimensions of the pattern in rows. The *Min* and *Max* boxes contain the minimum and maximum dimensions of the pattern in rows.

List Options

The *List Options* button is related to a list-type Pattern Definition. For details, see the chapter about lists in *JIS Interface Server: Advanced Topics*.

Modifying the Range of a Pattern Definition's Location

The location of a Pattern Definition varies according to the lower level Pattern Definitions that compose it.

Example 60. Modifying the range of a pattern definitions' location I

Let us refer again to a pattern that can have three different dimensions:

```
Line 1 of 5
Line 1 of 15
Line 11 of 15
```

When this pattern appears right justified in a screen, the location at which this Pattern Definition begins is not fixed, but varies.

A Pattern Definition's location also depends upon the location or dimensions of other Pattern Definitions that appear before it on the screen.

Example 61. Modifying the range of a pattern definitions' location II

Let us assume a screen contains the following line:

```
City ===> San Diego     ZipCode ===> 92107
```

The screen is designed in such a manner that the ZipCode pattern always begins five characters after the last character of the City pattern. Therefore, this line can also have the following form:

```
City ===> Huntington Beach     ZipCode ===> 92648
```

In such a case, the location of the ZipCode pattern depends on the dimension of the City pattern.

It is important to note that:

- ACE does not enable you to modify the location range when it is not appropriate. In such a case, after setting the *Lock Selection* check box, the *Location Min* and *Max* boxes are empty.
- It is not possible to change the location range of one of the lower level Pattern Definitions that compose certain compound Pattern Definition types. For example, it is not logical to change the location range of a Pattern Definition that is one of the components of a Horizontal Group, a Vertical Group, a Horizontal Iteration or a Vertical Iteration. The components of these compound Pattern Definition types appear together in a specific sequence, and the location and dimension of each component depend upon the location and dimension of the other components. Therefore, you may only modify the location range of the compound Pattern Definition itself.

To modify the range of a Pattern Definition's location:

1. Select the Pattern Definition to be modified.
2. Set the *Lock Selection* check box. The arrow keys beside the *Min* and *Max* boxes are activated.
3. In *Location,* use the *Min* and *Max* arrow keys to change the range of rows or the range of columns.
4. Confirm your changes by clicking the *Accept* button.
5. If you have changed your mind, you can at any stage select *Restart Field Information* from the *Runtime Field Information* menu to disregard all your modifications and return to the original default decomposition.

Modifying the Range of a Pattern Definition's Dimensions

The dimensions of a Pattern Definition depend upon its type. A basic Pattern Definition type, such as String, matches patterns that are fixed and do not change in dimension.

Example 62. Modifying the range of a pattern definition's dimensions I

The contents of a String type Pattern Definition, such as `USERID` and `PASSWORD`, are fixed and do not change their dimensions.

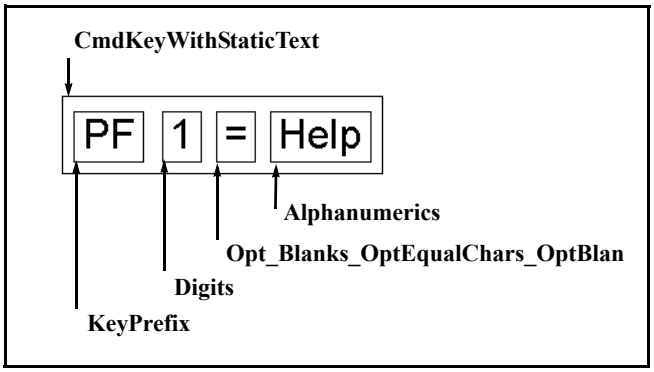
In contrast, a compound Pattern Definition type is composed of other Pattern Definitions of various types, that may vary in dimension. Therefore, the dimensions of a higher level Pattern Definition depend upon the dimensions of the lower level Pattern Definitions that compose it.

Example 63. Modifying the range of a pattern definitions' dimensions II

The dimensions of a Horizontal Group type Pattern Definition depends upon the dimensions of its components, and therefore varies accordingly. For example, let us examine the *CmdKeyWithStaticText* Pattern Definition, that matches patterns such as:

```
PF1=Help
```

This Pattern Definition is a Horizontal Group of the following Pattern Definitions: *KeyPrefix*, *Digits*, *Opt_Blanks_OptEqualChars_OptBlanks*, and Alphanumerics, as shown in the following diagram:



The contents of *Alphanumerics*, for example, may vary:

```
PF6=Forward
PF8=ScrollDown
```

Therefore, the dimensions of *CmdKeyWithStaticText* depend upon the dimensions of the Pattern Definitions that compose it.

In Runtime Screen Information View, it is not possible to change the dimension range of certain compound Pattern Definition types.

Example 64. Modifying the range of a pattern definitions' dimensions III

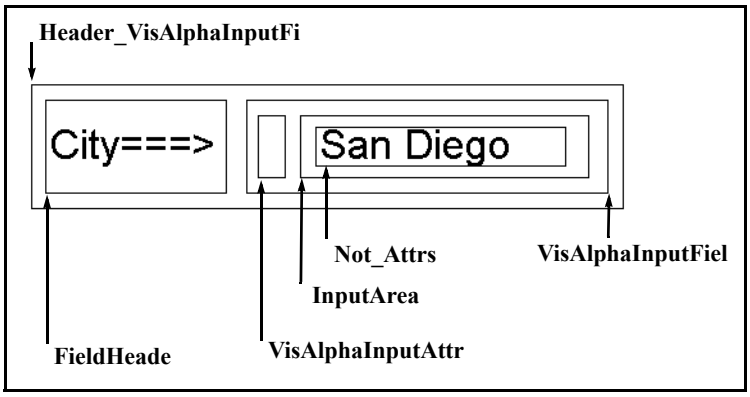
It is not logical to specify a dimension range for a Pattern Definition of Horizontal Group type, because its dimensions depend upon the dimension of the Pattern Definitions that compose it (see previous example).

On the other hand, you can specify a dimension range for a Pattern Definition of Horizontal Iteration type.

For example, let us examine the following example of a pattern:

```
CITY ===> San Diego
```

As a result of the analysis, the name of the city, San Diego, is matched with the Pattern Definition *Not_Attrs*, which is a Horizontal Iteration of the *Not_Attr* Pattern Definition. *Not_Attr* is a Character Set of any character that is not a screen attribute.



The name of the city may change during runtime, causing the dimensions to vary. Therefore, it is possible to specify a dimension range for the *Not_Attrs* Pattern Definition, whereas it is not possible to do so for the *Header_VisAlphaInputField* Pattern Definition.

To modify the range of a Pattern Definition's dimensions:

1. Select the variable Pattern Definition for which you would like to modify the dimension range.
2. Set the *Lock Selection* check box. The arrow keys beside the *Min* and *Max* boxes are activated.

3 Use the *Dimension* arrow keys to change the range of rows or the range of columns in the *Min* and *Max* boxes.
4 Confirm your changes by clicking the *Accept* button.
5 If you have changed your mind, you can at any stage select *Restart Field Information* from the *Runtime Field Information* menu to disregard all your modifications and return to the original default decomposition.

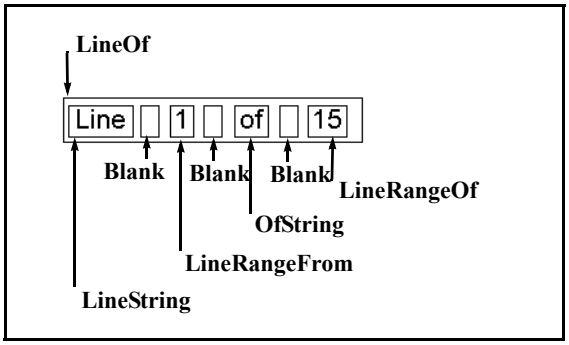
Example 65. Modifying a decomposition

The following example demonstrates how to modify a decomposition.

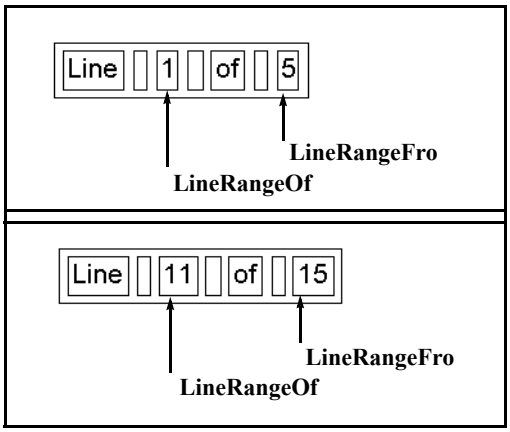
We shall refer once again to the following pattern, and demonstrate what should be done in the *Decomposition Definition Properties* dialog box:

```
Line 1 of 15
```

The analysis recognizes such a pattern in the following manner:

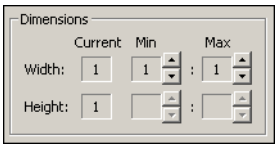


In this pattern, the LineString and OfString Pattern Definitions match patterns that do not change in content. In contrast, the patterns matched by the LineRangeFrom and LineRangeOf Pattern Definitions may vary in their content, as can be seen in the following two examples:

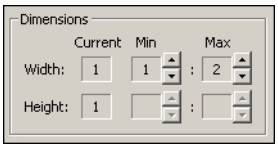


Therefore, it is necessary to inform ACE of these variations.

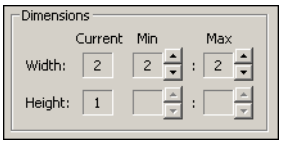
The default decomposition of the LineRangeFrom Pattern Definition is as follows:



The current width dimension is 1, and the dimension range is Min:1 - Max:1. However, the dimensions of LineRangeFrom may vary between Min:1 - Max:2. Therefore, use the arrows to increase the *Max* value to 2 (*Lock Selection* must be set) and to obtain the following result:

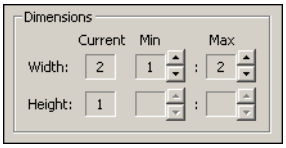


The default decomposition of *LineRangeOf* also needs to be modified:

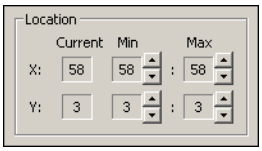


The current width dimension is 2, but the dimensions of *LineRangeOf* may vary between Min:1 - Max:2.

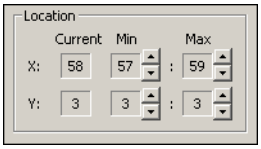
Therefore, use the arrows to decrease the *Min* value to 1 (*Lock Selection* must be set) and to obtain the following result:



Finally, when the *LineOf* Pattern Definition appears right justified in a screen, then the location of the beginning of the pattern also changes according to its dimensions. In such a case, you also need to modify the default decomposition of *LineOf*:



The *Location* range values need to be modified: the *Min* value should be decreased by one, and the *Max* value should be increased by one, to obtain the following result:



Note: Click *Accept* to confirm your changes.

Chapter 18. Multiple Subapplication Features

The features in this chapter discuss Subapplications as a group. These features enable you to sort Subapplications according to categories, extract information from a group of Subapplications, or perform certain tasks on a group of Subapplications.

This chapter describes:

- Using Batch Processing
- Query

When you have learned to work in Batch mode you will be able to apply changes or restart Subapplications of a selected group in certain views.

Reading about the Query feature teaches you to create groups of Subapplications according to properties that you specify.

Read about Log changes to learn how to find Subapplications that have been affected by KnowledgeBase modifications.

Using Batch Processing

Batch processing is a wizard-driven feature that applies KnowledgeBase modifications to Subapplications. After initial setup, batch processing does not routinely require user intervention, however, certain rare processing errors may still require attention.

Batch processing can be run on an entire Application or on a library, and on Subapplication subsets within the Application or library.

Using Subsets

The use of Subapplications subsets targets only selected Subapplications for batch processing.

Example 66. Using subsets

Two developers are working a single Application. Each developer is responsible for certain types of Subapplications. To run a batch process on developer A's Subapplications, save a subset containing the Subapplications assigned to developer A.

Subapplication subsets can be saved in the *Batch Processing* wizard and in the *Query* dialog box.

Accessing Batch Mode

The *Batch Processing* wizard can be entered directly by choosing *Batch* from the *Subapplication* menu. The *Batch Processing* wizard is also available by clicking the *Batch* button in the *Query* dialog box.

Implementing Batch Processing

To set up batch processing:

1. Select Subapplications for processing. The selection step has an option to save the selected Subapplications as a subset. The Subapplications available for selection are the Subapplications that are contained in the Application or library that is currently open.
2. Configure the processing options for each ACE View. There are wizard steps for each ACE view. The options are to apply or discard all local modifications that were made to Subapplications. Applying or discarding local modifications in the selected Subapplications is handled through the wizard's *View Processing Options* steps.

Note: All menus and toolbars are disabled during batch processing.

Processing an Application or a library

To process an application or library:

1. In the *Subapplication* menu, select *Batch* to start the *Batch Processing* wizard.

   -or-

   Press *Batch* in the *Query* dialog box.

   The *Batch Processing* wizard appears. Press *Next*.
2. Select the Subapplications to include in the Batch:

   Press *Select All*

   -or-

   Select a Subapplication from the Subapplication list.

   To select more than one Subapplication, hold down the *Ctrl* key.

   -or-

   Select a Subapplication subset from the *Select subset* combo box.

   Optional: Save the selected Subapplications as a subset by clicking *Save Subset* and entering a subset name in the *Save Subset As* dialog box.
3. Choose one of the options in the *View Processing Options* wizard steps.

   The default choice is *Reapply local modifications*.
4. Enable or disable the Batch log.

   If *Log batch process* is checked, the *Log file name* edit box and the *Browse* button are enabled. Enter a file name in the edit box or choose a file name and location using the *Browse* button.
5. Enable or disable the Subapplications log.

   If *Log subapplication changes* is checked, the *Options* button is enabled. Pressing *Options* opens the *Log Options* dialog box where you can specify the log file and enter any other necessary information.
6. Press *Finish*. Batch processing begins. The *Batch Process* dialog box is displayed.

The Batch Processing Dialog Box

A dialog box, similar to the *Generate Runtime* dialog box, is displayed while the batch process is running.

The *Batch Process* dialog box has three buttons:

- *Pause* - Pauses the batch processing. When the process is paused, this button becomes *Resume*. Press *Resume* to restart batch processing.
- *Break* - Terminates batch processing.
- *Close* - Closes the *Batch Process* dialog box. This button is enabled only when batch processing is completed.

The *Batch Process* dialog box remains open at the end of the process; the message `-Batch process completed-` is displayed.

The processing record can be viewed in the *Batch Process* dialog box. To scroll through the processing record use the *Page Up*/*Page Down* buttons or the scroll bar.

The Log Files

Running a batch on the Application after having made global modifications to the KnowledgeBase may produce undesired changes in some Subapplications. The log files show which Subapplications have been affected by the KnowledgeBase modifications and the discrepancies.

The batch feature offers two log options: Log batch changes and Log Subapplication changes.

The Batch Changes Log

The Batch log records messages generated during batch processing.

Each time a batch is run, new information is added to the log file. The first information written to the log file is a header containing the date and time of the batch process.

Three types of messages appear or are logged during batch processing:

- Error messages appear in the *Batch Process* dialog box and are logged in the Batch log file.
- Subapplication processing messages are displayed in the converter's message line in the *Batch Process* dialog box and are logged in the Batch log file.
- Processing stage messages appear only in the converter's message line.

Enabling Batch log files:

1. In the *Log Batch Process* step of the *Batch Processing* wizard, check the *Log batch process* checkbox. The *Next* button is disabled until a file name is entered in the *Log file name* edit box.
2. Type a file name in the edit box and click the *Browse* button to open the *Save As* dialog box and to set the file location, or use the *Browse* button to select a file.
3. Press *Next*. An error message is displayed if the file name is illegal, the location is not found, or if the operator does not have permission to write the file.

Note: The Batch log file and file location entered in this step remain the default for each batch process in the Application.

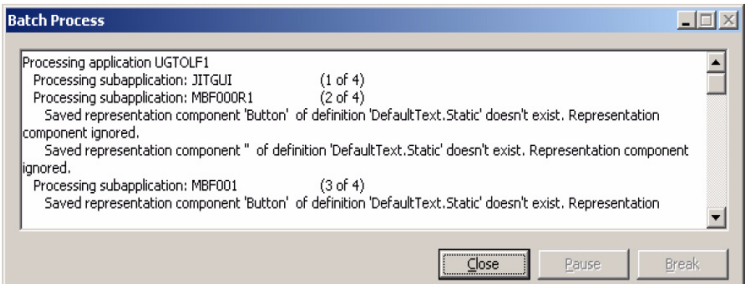


Figure 199. Batch process dialog box

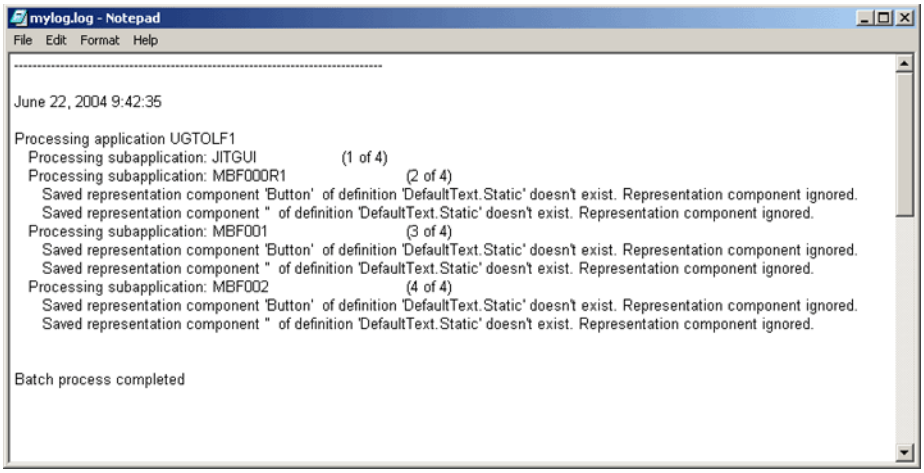


Figure 200. Extract from a batch file log

An example of a batch processing during which errors occurred is shown above in the dialog box and corresponding log file extract.

The Subapplication Log

When the *Subapplication Log* option is enabled, ACE compares the Subapplication before and after the KnowledgeBase modifications have been applied to it during batch processing. The results are written to a *.log file.

```
[MBF000R1] Change found: Different trigger of control SimpleButton
[MBF000R1] Change found: Different trigger of control SimpleButton
[MBF000R1] Change found: Different trigger of control SimpleButton
[MBF001] Change found: variable Sub0501501015: Control format
[MBF001] Change found: variable Sub0601501009: Control format
[MBF001] Change found: variable Sub0701501016: Control format
[MBF001] Change found: variable Sub0801501014: Control format
[MBF001] Change found: variable Sub0901501012: Control format
[MBF001] Change found: variable Sub1001501017: Control format
[MBF001] Change found: variable Sub1101501021: Control format
[MBF001] Change found: variable Sub1401501010: Control format
[MBF001] Change found: Different font for control SimpleWindow
[MBF001] Change found: Different rectangle of control SimpleButton
[MBF001] Change found: Different rectangle of control SimpleButton
[MBF001] Change found: Different rectangle of control SimpleButton
[MBF001] Change found: Different rectangle of control SimpleButton
[MBF001] Change found: Different rectangle of control SimpleButton
[MBF001] Change found: Different rectangle of control SimpleButton
[MBF001] Change found: Different rectangle of control SimpleButton
[MBF001] Change found: Different type of control (SimpleStatic<>SimpleButton)
[MBF001] Change found: Different rectangle of control SimpleStatic
```

Figure 201. Extract from a Subapplication log file

The figure above shows an extract of a Subapplication log file. The file contains the names of the Subapplications where changes have been found and the nature of the changes. The highlighted line indicates that a change in the properties of a control was detected in the Subapplication [MBF001] before and after batch processing.

Defining Logging Parameters

Changes are logged according to pre-defined parameters. The parameters are defined in the *Log Options* dialog box.

Set the parameters to reflect the changes that are likely to occur in the Subapplications due to modifications made to the KnowledgeBase. During batch processing, ACE logs only the changes that comply with the preset parameters.

To access the *Log Options* dialog box:

1. In the *Log Subapplication Changes* step, set the *Log subapplication changes* checkbox. The *Options* button is enabled.
2. Click *Options*. The *Log Options* dialog box opens.

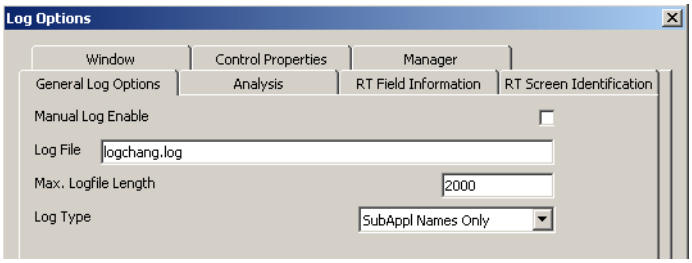


Figure 202. Log Options dialog box

In this example, ACE logs the difference in the number of controls and in the number of templates generated by Representation Definitions before and after running a batch on the assigned Subapplications.

The Log Options Tabs

Clicking a tab opens dialog boxes for various parameters. Each parameter is represented by a checkbox entry.

General Log Options Tab

Manual Log Enable	When checked, manual modifications to the Subapplications are recorded in the log file.
Log File	Assign a name for the log file.
Max. Logfile Length	Maximum number of lines the log file can contain.
Log Type	*SubAppl Names Only* - Logs only the names of Subapplications in which changes have been detected. *Modification Details* – Logs detailed information about the changes in Subapplications.

Analysis Tab

Screen Pattern Representations	Logs the number of screen representation patterns modified.
Representation Definition	Logs whether the selected (red) Pattern Definition in the Representation Definition structure tree has been modified.

RT Field Information Tab

Number of Decomposition Definitions	Logs the differences in the number of decomposition definitions.
Definition Name	Logs modifications in definition names.
Location and Dimension	Logs modifications in location and dimension.
Extended Information	Logs modifications in extended information.

RT Screen Identification Tab

Color Change	Logs modifications in color.
Popup Location and Size	Logs modifications in popup location and size.
Popup Border Definition	Logs modifications in popup border definition.

Window Tab

Number of Controls	Logs differences in the number of controls.
Number of Menus	Logs differences in the number of menus generated by Representation Definitions.
Number of Templates	Logs differences in the number of templates generated by Representation Definitions.

Number of GUTMs	Logs differences in the number of General User-Triggered Methods generated by Representation Definitions.
Number of Accelerators	Logs differences in the number of accelerators generated by Representation Definitions.
Validity Check	Logs modifications in validity checks.
Trigger	Logs modifications of triggers attached to controls generated by Representation Definitions.

Control Properties Tab

Control Class (Type)	Logs control class (type) modifications.
Location and Dimension	Logs modifications of a control's location and dimension.
Control Style	Logs control style modifications.
Control Caption	Logs control caption modifications.
Background color	Logs background color modifications.
Text color	Logs text color modifications.
Font	Logs control font modifications.

Manager Tab

Number of Representation Variables	Logs the number of representation variables.
Representation Definition Name	Logs modified Representation Definition names.

Screen to Manager Update	Logs whether Runtime Data Flow check boxes have been checked/unchecked.
Manager to Window Update	Logs whether Runtime Data Flow check boxes have been reconfigured.
Window Formats	Logs formats used on the window's controls.
Save to INI	Logs whether Save to INI parameters have changed.
Variable Name	Logs variable name modifications.
Hierarchy	Logs variable hierarchy.

Query

The Query feature is a tool used to identify and group Subapplications by their properties. This is useful in organizing batch processing or generating a runtime only for Subapplications with a particular property.

To access the Query option, select *Query* from the *Subapplication* menu. This opens the *Query* dialog box.

The Query Dialog Box

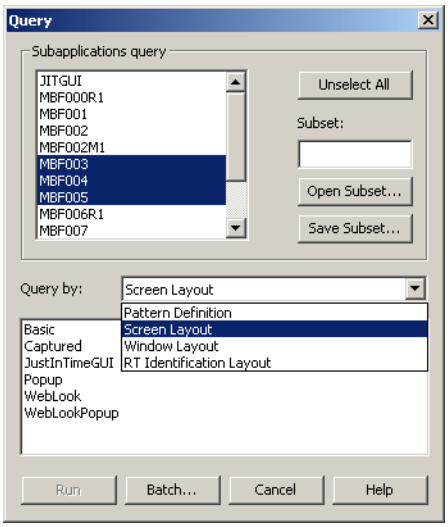


Figure 203. Query dialog box

Selecting Subapplications

The *Subapplications query* box lists all the Subapplications in the current Application. You can select the Subapplications to query either individually from the list, or you can use the *Select All* button to highlight all the Subapplications.

The query is run on the highlighted Subapplications in the list.When no Subapplications are selected, the button reads *Select All*. If you click *Select All* the query is run on all the Subapplications. If any Subapplication is highlighted, the button reads *Unselect All.*

After the query is run, only the Subapplications that meet the conditions of the query remain highlighted.

Query by Conditions

Once the Subapplications are chosen, a condition for the query must be selected.

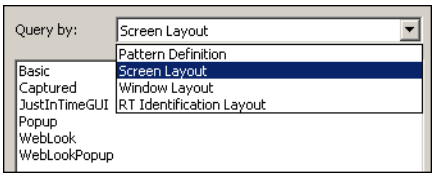


Figure 204. Query by option

The Subapplications can be queried for four separate types of criteria. Select one of these criteria from the *Query by* drop-down list.

Pattern Definition	Which of the selected Subapplications contain a specific Pattern Definition.
Screen Layout	Which of the selected Subapplications were analyzed using a specific screen layout.
Window Layout	Which of the selected Subapplications use a specific Window Layout.
RT Identification Layout	Which of the selected Subapplications use a specific Runtime Identification Layout.

Once a category has been chosen, a list of the category's definitions appears in the list box below. For example, if *Window Layout* was selected, the list of available window layouts appears.

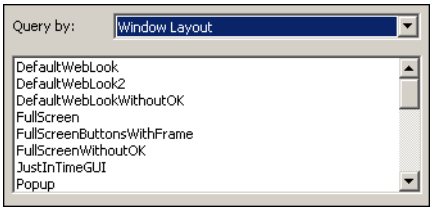


Figure 205. List of queried conditions

Note: If more than one layout or definition is highlighted, ACE selects all Subapplications which conform to one of the selected layouts or definitions.

Performing a Query

Click the *Run* button to run a query using the conditions selected.

ACE then unselects the Subapplications that do not meet the criterion, and leaves those Subapplications that meet the criterion selected. For example, if you query for the screen layout "Default", only the Subapplications with that screen layout remain highlighted.

When queried for Pattern Definitions, ACE searches each Subapplication. When querying Screen, Window, and Runtime Screen Identification Layouts, ACE queries only the information in the `[Subapplications]` section of the specific INI file. Therefore, to use this feature for querying layouts, you must first be sure

that the layout information is in the INI file. For information regarding entering layout information in the INI file, see the chapter about writing layouts to an *.ini file in *JIS Interface Server: Advanced Topics*.

Creating a Subapplication Subset

The selected Subapplications can be saved as a subset for future use. The subset that is generated from a query can be saved with a logical name. Further queries can be run, a batch can be performed, or a runtime can be created from this subset.

Saving a Subset

To save the queried Subapplications under a logical name, in the *Query* dialog box, click the *Save Subset* button.

Clicking the *Save Subset* button opens the *Save Subset As* dialog box:

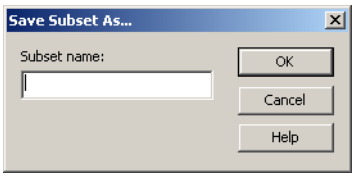


Figure 206. Save Subset As dialog box

Enter a logical group name in the field provided and click *OK*. The group name is saved to the `[Partial MakeExe Groups]` section of the Specific.ini file.

Calling Up a Subset

To call up a saved subset, click the *Open Subset* button:

The *Open Subset* dialog box opens, listing all of the existing subsets. Select the desired group and click *OK*.

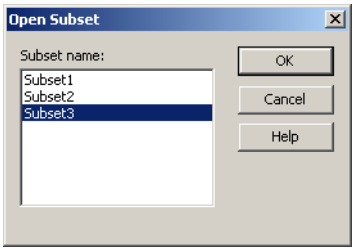


Figure 207. Open Subset dialog box

ACE returns to the *Query* dialog box. The Subapplications that are part of the chosen subset are highlighted.

Querying Subapplications Conforming to Multiple Conditions

To select Subapplications which meet more than one condition, a query must be run for each separate condition. Remember, a query for layouts searches text in the INI file whereas a query for Pattern Definitions searches each Subapplication. Therefore, be sure to initially query layouts, because this is the faster process.

To query Subapplications for both a Screen Layouts and Pattern Definitions, do the following:

1. Run a query using a Screen Layout condition.
2. Query the highlighted Subapplications for the Pattern Definition. It is not necessary to select the Subapplications again.

The Subapplications that meet both conditions are highlighted in the list box. You can either run a batch or a generate runtime on this group, or save it under a logical name.

Chapter 19. The Runtime Application

The converted host application has to be compiled and prepared for distribution. This chapter includes the information you will require to bring the converted application to a fully operational state on the end user's computer. It also includes an explanation about the runtime application interface.

This chapter describes:

- How Does the Compiler Work?
- Configuring the Generate Runtime Process
- Configuring Your Test Subset
- Producing the Executable File
- Suppressing Error Messages While Compiling
- Running the Generated Application
- About the Runtime Application
- System Setup
- Demos and Testing

Generating the Runtime

Use the *Generate Runtime* wizard to compile Subapplications and create an executable runtime. Compile all the Subapplications in the Application or configure a subset of Subapplications that may be in a smaller runtime version.

The *Generate Runtime* wizard is accessible from the *File* menu in all ACE Views.

How Does the Compiler Work?

ACE is based upon a proprietary object oriented language named GAL (Graphical Application Language). GAL is used internally only. The Application developer does not come in contact with GAL.

At the end of the conversion process, the Generate Runtime command creates an executable file by generating the GAL code and compiling and linking. Most of the GAL code is generated from information obtained during the conversion process, such as the graphic display information obtained and the methods defined in Design View.

In addition to the conversion information, there are several built-in GAL sources which are also used during compilation and linkage. The following diagram demonstrates the compilation process:

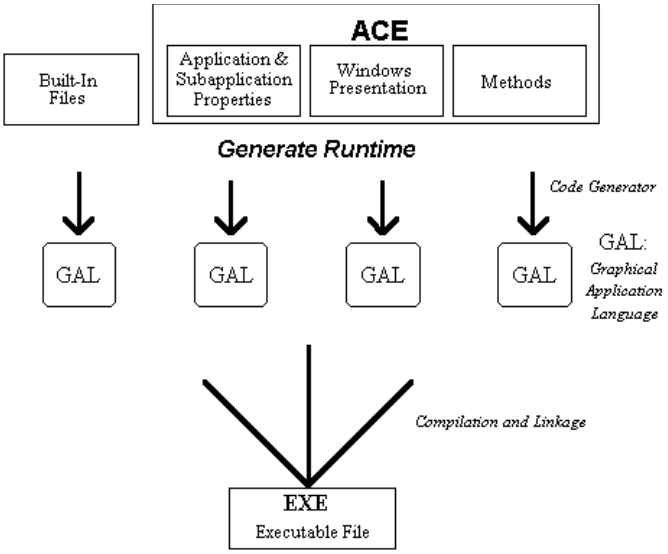


Figure 208. Diagram of the compilation process

Configuring the Generate Runtime Process

This section describes how to select individual Subapplications for inclusion in the runtime and determine whether or not they need to be recompiled. For additional information about libraries, see the section about creating a runtime in *JIS Interface Server: Getting Started with the Automated Conversion Environment*.

To generate the runtime:

1. From the *File* menu select *Generate Runtime*. This activates the *Generate Runtime* wizard.
2. The *Generate Runtime* wizard leads you through the steps necessary to create a runtime suitable to the platform you are using.
3. In the *Select Subapplications to Include in Runtime* step, choose:

All	To create an executable file including all the Subapplications in your Application.

Subset To create an executable file consisting of a user-defined subset of Subapplications. Click *Select* to open the *Subapplication Subsets* dialog box.

Generate Runtime Including all the Subapplications

When you choose *All*, the wizard allows you to further specify your choice. To save time during the *Generate Runtime* process, you may wish to process only those Subapplications that have changed since the last compilation. If such is the case make sure *Only new and modified* is selected.

ACE automatically discards the unchanged Subapplications.

Figure 209. Subapplications to Process option

Note: When compiling an Application for the first time, the *Generate Runtime* wizard skips the *Select Subapplications to Include in Runtime* step and all Subapplications are included in the compilation. The *Select Subapplications to Include in Runtime* step only becomes available after the first successful runtime generation.

Generating Runtime Using Subsets

ACE enables you to create a runtime Application from a subset of all the Subapplications in an Application. This is very useful for testing a specific subset of Subapplications in runtime.

Generating runtime using subsets is a time-saving feature. If you are creating a runtime version for a subset of Subapplications, where some of them have been modified, you can save time by performing the Generate Runtime only on those that were modified.

Creating a Test Subset

Generating runtime using subsets enables you to define a subset of Subapplications to be used for creating a runtime Application, and to save the subset for future use. Thus you can build a small Application made up of a subset of Subapplications, save this subset and recompile this subset at another session of ACE, without redefining the subset.

Configuring Your Test Subset

1. In the *Generate Runtime* wizard, in the *Select Subapplications to Include in Runtime* step, select *Subset*. The *Select* button is enabled.
2. Press the *Select* button. The *Subapplication Subsets* dialog box is displayed.

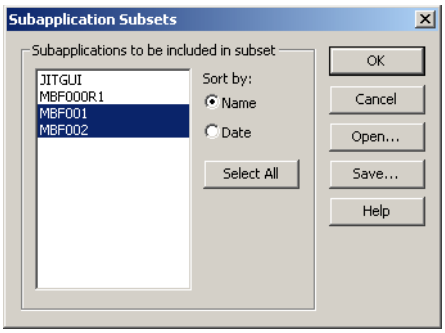


Figure 210. Subapplication Subsets dialog box

The *Subapplication Subsets* dialog box displays all the Subapplications in the current Application. You can display the Subapplications sorted either by date or by name.

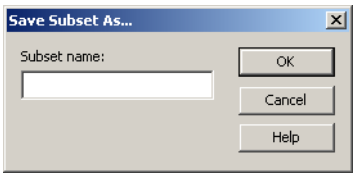
Select an individual Subapplication from the list by clicking on it, or select all of the Subapplications using the *Select All* button.

Note: Click *Select All* if you are selecting most of the Subapplications. Then unselect the Subapplications that are not to be included.

At the bottom of the screen, you can select the names of the first and last Subapplication of the runtime version. Only the Subapplications selected in the partial list are displayed in the combo box.

To save a subset:

1. Select the Subapplications you wish to include in the subset.
2. Click *Save*. The *Save Subset As* dialog box opens.

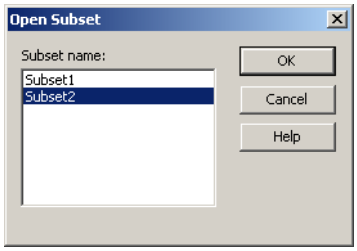


3. Type the subset name and click *OK*.
4. In the *Subapplication Subsets* dialog box, click *OK*.

Note: It is not necessary to save your configuration, however if you do not, you will have to reconfigure the subset again if you change the configuration.

To open an existing subset

1. In the *Subapplication Subsets* dialog box, click *Open*.
2. The *Open Subset* dialog box opens containing a list of all the subsets defined for this Application.



Note: Only subsets defined for this Application are listed

Subsets in the Converter INI File

The parameters defined in the Generate Runtime function are recorded in two groups in the specific.ini file: `[MakeExe Parms]` and `[Partial MakeExe Subsets]`. If you wish, you can change the parameters of your subset by editing the entries in these group of this *.ini file.

[MakeExe Parms]

The first time you create a partial runtime your system creates this group title in the `specific.ini` file.

[Partial MakeExe Groups]

When you save a subset, your system creates this group title in the `specific.ini` file.

Producing the Executable File

Before compiling your first application, set the Runtime Generation Options. To do this, see the sections entitled "Setting Runtime Generation Options" in *JIS Interface Server: Java Client User's Guide* or *JIS Interface Server: XHTML Client User's Guide*.

The Generate Runtime Command

The *Generate Runtime* command causes an application generator to create code, compile and link it, in order to produce an executable file, which is given the name of the Application: <ApplName>.exe.

To produce the executable file:

1. From the *File* menu, select *Generate Runtime*.
2. Go through the steps of the *Generate Runtime* wizard. When you press *Finish*, ACE starts the compilation process and the *Generating the Runtime* dialog box is displayed:

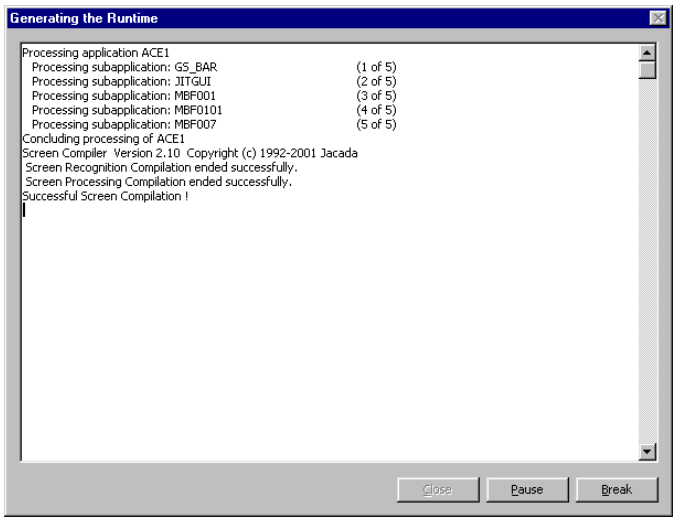


Figure 211. Generating the Runtime dialog box

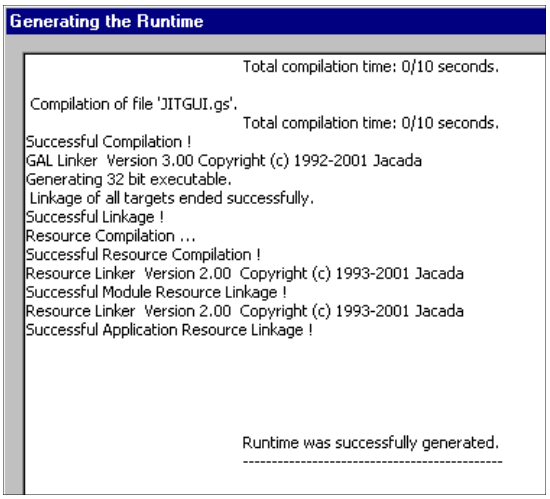
The *Generating the Runtime* dialog box displays information regarding the progress of the compilation and linkage as displayed in the window above. Use the scroll bar to view the various messages.

3. When the compilation ends successfully, the message -`Successful Screen Compilation!`- appears, and the linking process begins.

   While linking is proceeding, the DIL displays messages regarding Phase 1, 2, and 3, which indicate the progression of the linkage. When the linkage ends successfully, the message *"Successful Linkage!"* appears.

   Afterwards the screen compiler is activated, and when this process ends successfully, the message *"Successful Screen Compilation!"* appears.

   Finally, the resource linker is activated, and success is indicated by the message *"Successful Resource Linkage!"*. The following screen displays these messages:



4 At the end of a successful Generate Runtime, the DIL displays the message:

```
Runtime was successfully generated.
```

The *Generating the Runtime* dialog box contains three buttons:

Pause/ Resume Clicking *Pause* suspends the current process. This button then changes to display *Resume*. When you are ready to proceed, choose *Resumse*, which resumes the process. The button changes back to display *Pause*.

Break Clicking *Break* stops the compilation. When you break the following message is displayed:

```
Application processing has been aborted by the
user.
```

Both the *Break* and the *Pause* buttons are disabled, and the *Close* button is activated.

Close Click *Close* to exit Generate Runtime. The *Close* button is disabled when you begin generating runtime. *Close* is activated only after clicking *Break*, or at the end of the *Generate Runtime* process.

Suppressing Error Messages While Compiling

You can configure your system not to display certain messages while compiling. This will increase the speed of the compilation.

- Ignore the message *VGA out of bound* by adding the following to your specific.ini file:

  ```
  [Ignore Messages]
  ```

```
Window is positioned outside the boundaries=1
```

- Ignore the message *Pattern xxx not in section/msc* while loading the KnowledgeBase by adding the following:

```
[Ignore Messages]
10588=1
10589=1
```

Running the Generated Application

When *Generate Runtime* is successful, you are ready to run the generated Application. You can run the generated Application in several ways:

Running Your Application from the Windows Program Manager

Exit ACE. Create an icon in your Windows Program Manager that will run your generated application. Double click on this icon to run the program.

About the Runtime Application

Your runtime Application is a program composed of a runtime environment and the GUI Application you developed in ACE. These two parts are bundled together in ACE, into a distribution that you install on your target systems.

With the runtime Application your end-users can operate the host application through the graphical interface. To do this, you must install the GUI Application on the target system, and setup the target system to operate the host application from a PC. Your target systems communicate with your host application using an emulator.

Creating the Runtime Installation File

From the *Utility* menu select *Create Runtime Installation*.

ACE starts the *Create Runtime Installation* wizard which prompts you for all the information it needs to create a runtime. You can also instruct the wizard to produce a text file listing the files you need to include in an installation you can create manually.The installation is what you distribute to your end users.

Runtime License

Your runtime Application requires a runtime license. This is supplied by your Software GmbH representative. To acquire a license key you must supply Software GmbH with the following information about your Application:

- Name of company
- Name of Application
- Number of sessions. The licensing fee is dependent on the number of sessions.

Note: The default runtime license allow you to run up to 10 concurrent sessions

System Setup

This section deals with the setting up of your system.

Types of Emulators

ACE can work online with your host computer using a “live” emulator (for actually operating your host application), or offline, using a simulated emulator called the File emulator. A live emulator communicates with your host just as a terminal does. The File emulator is used for simulating operation online.

The File Emulator

You can operate your GUI application offline, simulating operation with the host computer, using the File emulator. To run a simulated version of your Application on the target system offline, the PNL files must reside on the target system. These files are not installed on the target system with the runtime Application. To run the GUI Application offline, copy these files from the development system and install them on the target system.

You can simulate a host by creating a PANELS.INI file. In this case, all the *.PNL files, as well as the PANELS.INI file should be located in the Application’s runtime directory:

```
…\<applname>\RT32
```

"Live" Emulators

All of the systems using the runtime Application use a "live" emulator to communicate with the host computer. The "live" emulator runs your host application in the background while ACE runs your converted Application in the foreground. Your converted Application is completely synchronized with your host application. When you enter data on the converted screen it is transferred to the host.

Figure 212 illustrates the logical connection between your host application, the terminal emulator, and your GUI application. Whereas a terminal communicates directly with the host computer, the PC communicates to the host via a terminal emulator.

ACE links and synchronizes between the GUI Application and the emulator.

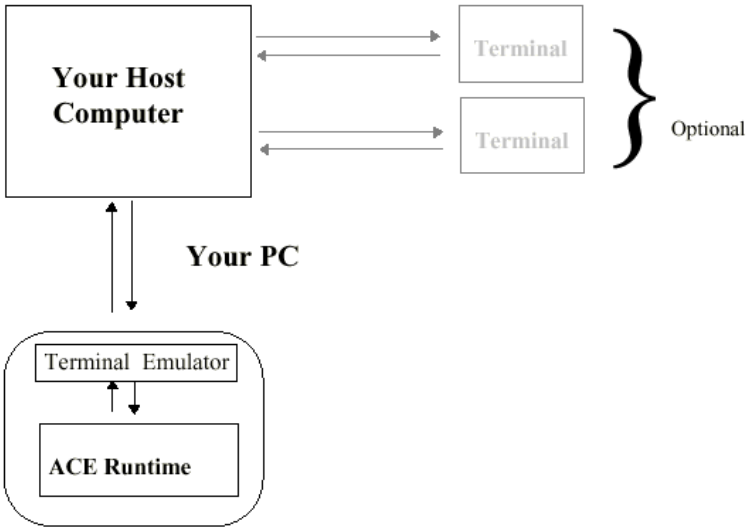


Figure 212. Diagram of connection between host, emulator and GUI

Demos and Testing

You can facilitate the maintenance of applications and help perform demonstrations by using PANELS.INI. For example, when maintaining applications, you may want the runtime to display only a few of the many panels contained in the application. When performing demonstrations you may want to arrange the sequence in which panels are displayed. The PANELS.INI file is generally used when the emulator is set to File mode.

Using PANELS.INI you can configure the runtime to:

- Display specific panels.
- Prompt you to choose a panel from a network or hard drive.
- Automatically stop a demonstration when the last panel is displayed.

- Loop a sequence of panels so that a demonstration automatically begins again.
- Advance to another panel from menus or lists.

Setting Up PANELS.INI

Using a text editor you can configure the PANELS.INI file to change runtime behavior in a number of different ways. After PANELS.INI is configured, save PANELS.INI in the runtime directory located under the application's directory:

`…\applname\RT32`

By pressing a key the panels advance one after another. Different PANELS.INI configurations change the way panels advance during runtime. Editing parameters in the PANELS.INI `[GeneralInfo]` section sets the way your demo runs. The key that is pressed must have functionality in relation to the panel it is advancing from. If *F7* is the key which moves the runtime to the next panel, then the panel you are advancing from must have the *F7* functionality associated with it.

The panels (*.PNL files) used in conjunction with PANELS.INI must also be copied to the runtime directory together with the PANELS.INI file.

Advancing to Specific Panels

There must be a section for each panel. The line that follows the panel name section represents the key pressed to advance to the destination panel. In the example below, panels AAA and CCC are represented by a section within the PANELS.INI file. *Enter* and *F7* represent keys that when pressed, advance the runtime to the next panel. BBB, CCC, DDD, and EEE represent destination panels.

[AAA]	Name of panel.
Enter=BBB	With AAA displayed in runtime, pressing *Enter* advances runtime to panel BBB.
F7=CCC	With AAA displayed in runtime, pressing *F7* advances runtime to panel CCC.
[CCC]	Name of panel.
F7=DDD	With CCC displayed in runtime, pressing *F7* advances runtime to panel DDD.

Enter=EEE With CCC displayed in runtime, pressing *Enter* advances runtime to panel EEE.

Prompting to Choose a Panel from a Directory

In this example pressing the *Enter* key during runtime opens a dialog box prompting you to choose the next panel from a network or hard drive. This happens when demo mode is turned off. `DEMO=1` turns demo mode off.

```
[GeneralInfo]
Demo=1
```

Automatically Stopping a Demonstration

Using this configuration, pressing the *Enter* key advances the runtime, panel by panel, until the last panel is displayed. When the runtime reaches the last panel pressing the *Enter* key has no effect. The demonstration is halted.

- `DEMO=2` turns demo mode on.
- `PanelsDirectory=.` sets the runtime to display panels from the current Application's runtime directory.
- `LoopOnPanels=0` stops the demonstration after the last panel is displayed.

```
[GeneralInfo]
Demo=2
PanelsDirectory=.
LoopOnPanels=0
```

Continuously Looping a Demonstration

Using this configuration, pressing the *Enter* key advances the runtime, panel by panel. Pressing the *Enter* key when the last panel is displayed advances the runtime to the first panel. The sequence of panels can be displayed again by pressing the *Enter* key.

- `DEMO=2` turns demo mode on.
- `PanelsDirectory=.` sets the runtime to display panels from the current Application's runtime directory.
- `LoopOnPanels=1` automatically sets the demo to begin after the last panel is displayed.

```
[GeneralInfo]
Demo=2
PanelsDirectory=.
LoopOnPanels=1
```

Advancing to Specific Panels from a Menu or List

In some host screens such as menus or lists, typing a string and pressing a key moves the program between panels. Often there may be a number of panels associated with a menu or list. The following PANELS.INI line displays the text used to select the item from the menu or list, the key pressed and the destination panel.

```
SwitchedEnter=V;Main
```

Indicates that when you select option “V” and press *Enter*, the program moves to the “Main” screen.

> **Note:** You can use panels from any directory by editing the `PanelsDirectory=` parameter in the `[GeneralInfo]` section. Enter the full path where the panels reside after the equals sign. For example, `PanelsDirectory=C:\My_Appls\Panels`

Index

Symbols

.INI
- files
  - panels 520
  - read data from 220
- in generate runtime 515
- PANELS.INI 520
- subsets 515

A

Accelerators 225
AfterPageUpDown 409
Analysis
- defined 80
- of host screens 58
  - example 71

Analysis view
- color-coding scheme 113
- editing pattern definition properties 147
- editing pattern definitions 136
- ignoring a pattern definition 117
- modifying location or dimension 117
- moving through a pattern definition 114
- operations 111
- pattern definitions 121
  - modifying 118
- screen image 112

Apply 193
Attaching
- methods
  - to floating menu items 300

Attributes 52
Auto Ident option 476
Automatic menu 303

B

Basic pattern definitions 492
Batch
- dialog box 499
- log batch changes 500
- log subapplication changes 500
- mode
  - query 506
- subapplication log 502

Bitmaps 228
BMS
- BMS/MFS
  - creating screen images 40
- file format 30
- filter sections 106
- workflow 39

Break (method line type) 369
Button 226

C

Changes
- undoing 193, 473

Character set 66
- described 66
- Parameters tab 148
- using efficiently 68

Check box 231
- formatting 278

Child patterns
- adding to new pattern definitions 146
- creating and positioning 140

Color
- color control 308
  - overview 308
- color-coding in Analysis view 113
- fixed 308, 311
- in Runtime Field Info view 487
- in Runtime Screen ID view 467
- setting up on Microsoft Windows 312
- setup 309, 312
- system 308
- table 308, 314

Combo box 233
- formatting 281

Comment (method line type) 369
Compiling
- suppressing error messages 517

Components
- deleting 188
- renaming 189

Compound pattern definitions 62, 491, 493
Controls
accelerators 225
adjust to text 179
advanced editing 180
alignment 180
attaching functionality to 197
button 226
associating with an image 228
center 178
check box 231
formatting 278
combo box 233, 281
HostBasedFormatValues 242
control styles 225
date field 236
deleting 188
dynamic group 237
frame 238
General user-triggered method 240
group box 240
leading control 176
line 246
link 247
list box 281
menu item 249
picture button 226
prompt 250
properties, modification of 197
radio group 252, 281
repositioning 191
resizing 192
selecting a group of 174
selecting controls in Design view 173
selecting fonts for 318
selecting individual controls 173
spin 254
static 257
subwindow 260
tab 267
table 262
text box 270
undoing local modifications 193
variable 268
window 268
Conversion process 512
Copying
text to the clipboard 119
Creating
BMS/MFS screen images 40
DDS screen images 35
screen images 42
for mainframe SDF applications 38
from screen captures 41

D

Data validity checks 222
Date field 236
DDS
files
creating DDSscreen images 35
filter sections 107
indicators 36
record formats 36
workflow 32
diagram 32
shared folders 32
step-by-step procedure 33
Decomposed screen
modifying 494
modifying pattern def.'s dimension 492
modifying pattern def.'s location 490
Decomposition properties
fixed data 488
lock selection 489
Delete captured screen image 42
Demos 520
Design
modifications 161
Design view 162
color and font 309
restart 194
selecting controls 173
toolbar 166
View menu 166
working in 166
Dictionaries 284
Do (method line type) 370, 383
DoMethods 383
commonly used methods list 392
list of available 429
parameters 390
pressing a host key 400
receiver 383
return types 430

scope 383
selection 389
syntax and terminology 384
Dynamic group 237
Parameters tab 149
Dynamic pattern definitions 73

E

Elimination (RT Screen Identification) 466
Else (method line type) 375
Emulator
file emulator 519
types 519
working with 520
EndDo (method line type) 370
Endif (method line type) 375
Enter method 338, 393
Error messages
supressing during runtime generation 517
Events 341
Executable file
producing 516
Expression
method line type 370
Extended Info tab 157
Extra fields 38

F

F_key method 338
File formats
BMS 30
IND 31
MFS 30
PNL 31
SDF 31
SDI 31
Filter section 83
BMS 106
DDS 107
defined 83
displaying 105
operations 105
submenu 109
toggling filter section display 105
Filtering the KnowledgeBase window 129
Fingerprint
location 472
runtime identification 467
runtime screen ID 466
runtime screen identification 471
Fixed characters
in runtime screen ID 468
marking in runtime ID screen 469
Fixed color 311
defined 308
Floating menus 297
attaching methods 300
attaching to a right click 299
attaching to controls 299
Floating representation
components
creating 206
Focus order 323
Fonts
absolute pixels 319
changes in control 319
changing font size 319
dialog units 319
font designing 307
in ACE runtime 318
selecting 318
Format
text format definition 277
Format tab 277
KnowledgeBase definitions window 277
Formatting
check box 278
combo box 281
list box 281
radio group 281
text 282
user dictionary 284
Frame control 238
FromWizard 153

G

GAL (Graphical Application Language) 511
General user-triggered method 240
Generate runtime
command 516
configuring 512
dialog box 516
subsets 513

suppressing error messages 517
GetToBottomOfList 409
Gettotopoflist 410
Group
dynamic 237
Group box 240

H

Hidden fields 38
Horizontal
group 492
iteration 493
Horizontal group 62
Parameters tab 150
using efficiently 65
Horizontal iteration
Parameters tab 151
Host key
pressing 400
Host screen analysis 58, 80
defined 80
example 71
Host screens
screen model type 33, 39
Host session window 90
Host view 51
HostBasedFormatValues 242
Hotkeys 295
assigning hotkeys to menu items 295

I

Identification
of screens 484
Identification definition 475
Ignore
ignoring a pattern definition 117
Images
associating with a button 228
IMS, no response 458
IND
file format 31
Indicators 36
Input focus order 323
Iterations 66
overview 66
properties of 70
using, in pattern definitions 70

K

KnowledgeBase
Definitions window 123, 203
filtering displayed patterns 129
navigating 128
pane operations and views 124
properties tabs 147
screen tab options 214
style tab 221
definitions window
properties pane 158
set/change location tab 158
rules 56
Screen tab 212
validation 136

L

Layout
menu 108
operations 102
Layout view 79
interface 90
operating instructions for 93
operations 102
Layouts
applying to screen image 102
arranging 102
creating 103
defined 79
editing 103
new 103
reading from INI files 104
removing from screen image 102
using 86
Leading control 176
Line control 246
Link control 247
ListFromWizard
type of pattern definition 136
Lists
commonly used methods list 392
list box
formatting 281
list columns 152

list of DoMethods 429
list of system-triggered methods 347
list sections 136
list sections by wizard 97
list with parameters 152
ListFromWizard 153
Local modifications
performing 114
Location
location range
fingerprint 472
location tab 158
Location parameter 60
Lock selection
Runtime Field Information view 489
Lockup 458

M

Mainframe
applications
creating screen images with SDF 38
locks up 458
Manager tab
KnowledgeBase definitions window 215
Masks, masking 273
Meaningful Patterns 468, 484, 487
Menus
creating automatically 304
default menu
setting 303
editing in design view 287
editor
editing menu items 290
floating menus 297
right click 297
subapplication level 289
layout 108
menu item 249, 303
list box 290
menu options
configuring 287
menu section by wizard 97
menu structure
item 288
popup Item 288
separator 288
primary/filter section 109
setting the default 303
View menu 166
menus
View 166
Message areas
in runtime screen ID 470
marking 470
Message definition 470
creating 147
designating 147
viewing 134
Methods
attaching to floating menu items 300
Do (method line type) 370, 383
DoMethod 383, 392
DoMethods return types 430
Else (method line type) 375
EndDo (method line type) 370
Endif (method line type) 375
Enter 338, 393
events 341
examples 397
Expression (method line type) 370
F_key method 338
general user-triggered method 240
GeneralUTMethod 240
HostType (method line type) 373
If (method line type) 375
list of available DoMethods 429
method line types 366
Break 369
Comment 369
Do 370
DoMethod 383
DoMethods, return types 430
Else 375
EndDo 370
Endif 375
Expression 370
HostType 373
If 375
list of available DoMethods 429
Msgbox 375
Return 378
Update 381
Msgbox (method line type) 375
parameters 385
pressing a host key 400

referencing
method lines 394
Return (method line type) 378
system-triggered methods 347
user-triggered methods 337
writing method lines 368
MethodsSelectMenuOption 339
MFS
file format 30
workflow 39
Microsoft Windows
system elements,table of 312
Modifications
undoing 193, 473
Moving sections on screen image 93
Multiple sessions
run application 518

O

OneOf
overview 64
Parameters tab 153
pattern definition 214, 215
using efficiently 65

P

PageDown 410
PageUp 411
Palettes
window components 184
Parameters
in methods 385
Parameters tab
dynamic group 149
horizontal iteration 151
list column 152
OneOf 153
Parent pattern 55
Partial generate runtime
configuring the test subset 514
Pattern definition
creating a new child pattern 146
display criteria
new 132
location 60
name 488
OneOfs 64
Properties dialog box 115
set/change location of 158
tree 212
Pattern definitions 55, 56
analysis with 58
character set 66, 148
child patterns
creating new 140
replacing 139
compound 62, 491, 493
concepts 55
creating new 144
deleting 139
display criteria
deleting 134
modifying 133
displaying selectively 129
duplicating 143
dynamic 73
dynamic group 149
dynamic iteration 149
global modifications of 121
graphic representation of 212
hierarchy
expanding and collapsing 127
viewing 125
horizontal group 62, 150
horizontal iteration 151
ignoring 117
iterations 66
list column 152
list with parameters 152
ListFromWizard type 136
modifying 118
modifying display criteria 133
moving through the hierarchy 114
OneOf 64, 153
ordering 82, 100
parent pattern
displaying 127
parts 56
popup border parameters tab 153
primary patterns 157
priority of 82, 100
scattered group 154
search order 59, 100
establishing 140

search order of 82
selecting by name 128
set as example 118
statistics on use of 135
string 57, 155
symbols for 126
tree
options 214
types
OneOf 214, 215
vertical group 155
vertical iteration 156
viewing 121, 125
Pattern display criteria
new 132
PNL file format 31
Popup border
Parameters tab 153
Pressing a host key 400
Primary pattern 60
definition 60
Primary patterns 157
Primary section submenu 109
Prompt control 250
Protected fields 38
Pull-down menu 303
structure 288

Q

Query
creating a subapplication group 509
overview 506
performing 508
Query dialog box 507
querying criteria 507
saving a group 509
selecting subapplications 507

R

Radio group
formatting 281
Radio group control 252
Receiver 383
Record formats 36
Redo 193, 473
Remove protections 38
Representation components
accelerators 225
button 226
check box 231
combo box 233
HostBasedFormatValues 242
date field 236
Format tab 212
frame 238
GeneralUTMethod 240
group box 240
line 246
Manager tab 212
menu item 249
prompt 250
properties 211
global editing of 211
radio group 252
Screen tab 212
spin control 254
static 257
Style tab 212
subwindow 260
tab 267
table 262
text box 270
variable 268
window 268
Representation definition
automatic menu 303
components 201
creating 205
deleting 207
creating 205
deleting 205
editing 199
example 163
menu item 303
Resizing controls 192
Resizing sections on screen image 94
Restart
design 194
Return types of DoMethods 430
Right mouse button
menu editor 297
Runtime
data flow example 216
Runtime application 511

invoking 518
license 519
system setup 519
Runtime demos 520
Runtime events 341
Runtime Field Information view 483
variable patterns 485
Runtime field information view
accessing 487
Runtime screen identification 483
automatic 475
elimination process 466
fingerprint 471
location 472
menu 466
modifying (Example) 477
screen 467
fingerprint 471
fixed and variable characters 469
fixed/variable characters 468
message areas 470
view 465
Runtime Screen Identification view
capturing a host screen image 480
comparing host screen and screen image 479
comparing two screen images 480

S

Scattered group
Parameters tab 154
Screen captures
and SDF 42
creating screen images 41
Screen decomposition
Runtime Field Information view 487
Screen identification 484
Screen image editor 43
Screen images
creating 29
using screen definition files 29
edit 43
maintaining 42
Screen model type 33, 39
model 3270 type 2 39
model 3270 type 3 39
model 3270 type 4 39
model 3270 type 5 39
model 5250 type 2 33
model 5250 type 5 34
specifying 34, 40
Screen tab 212
SDF
file format 31
screen captures 42
SDI
file format 31
Search order of pattern definitions
establishing 140
Section explanation panel 90
Sections 60, 78, 79
and analysis 80
applying to screen image 94
benefits of using 80
copying 99
defined 78, 80
deleting 100
dragging 90, 94
editing 97
for lists 136
marking menu or list 97
moving on screen image 93
new 99
ordering 83
overlapping 83
overview 60
priority of 83
removing from screen image 96
replacing on screen image 96
resizing on screen image 94
saving as 99
search order of pattern definitions 100
SelectMenuOption method 339
Send
SendAIDKeyByString 400
SendASpecificKey 400
SendKeyByString 400
SendKeysWithoutReset 400
Set
set as example 118
Set location tab 158
Spin control 254
Static control 257
Statistics
on use of pattern definitions 135
String

Parameters tab 155
Strings 57, 492
overview 57
uses of 67
Style tab
KnowledgeBase definitions window 221
Subapplications
changing default menu 303
features 497
menu editor 289
editing menu items 290
query 509
Subsets 498
in generate runtime 513, 514
.INI file settings 515
SubWindow control 260
Suggested window
leading control 176
System
color 308, 312
lockup 458
System-triggered methods 341

T

Tab control 267
Tabbing
leaving modify tabbing order mode 331
modifying tabbing order 326
resetting tabbing order 329
setting tabbing order 323
tabbing order 323
Table control 262
TableChangedSelection 411
Tabs
change location 158
Extended Info tab 157
Manager tab
KnowledgeBase definitions window 215
Screen
KnowledgeBase definitions window 212
Text
adjust control size by 179
formatting 282
Text box control 270
Toolbars
Design view 166
Triggers 337

U

Undo 193, 473
Update (method line type) 381
Updating
variables 381
User
color setup 312
dictionary 284
UserAcceptScreen 412
UserAfterRefreshSubapplication 413
UserAftertabFolderChanged 414
UserAfterTableAction 415
UserAttentionRequest 415
UserBack 416
UserBeforeTabFolderChanged 416
UserBeforeTableAction 417
UserCloseSubApplWindow 417
UserDestroyApplication 418
UserDestroySubapplication 419
UserHostHelpRequest 419
UserHostMessageHelpRequest 420
UserInitApplication 421
UserInitBeforeFirstSubappl 421
UserInitSubapplication 422
UserIsRealMessage 422
UserMoveToDependentScreen 423
UserMWIRequest 424
UserPreUpdateInitSubAppl 424
UserRefreshSubApplication 425
UserServerDataReady 426
UserShouldCloseSubApplWindow 426
UserShouldWindowBeBuilt 427
UserSkipSubApplication 427
User-triggered method 240
User-triggered methods 337
Using
system-triggered methods 347

V

Validity checks
external validity checks 224
internal validity checks 222
Variable characters
in runtime ID screen 468
marking in runtime ID screen 470
Variable control 268
Variables

attributes
updating 382
explicit name 219
name 218
patterns
Runtime Field Information view 485
updating 381
Vertical group 155
Vertical iteration 156
View menu 166
View runtime application
from compiler 518
Views
host 51
layout 79

W

Window
repositioning controls 191
Window components
deleting 188
renaming 189
Window components palette 184
Window control 268
Window menus
menu editor 289
Workflow for BMS/MFS
creating screen images 39